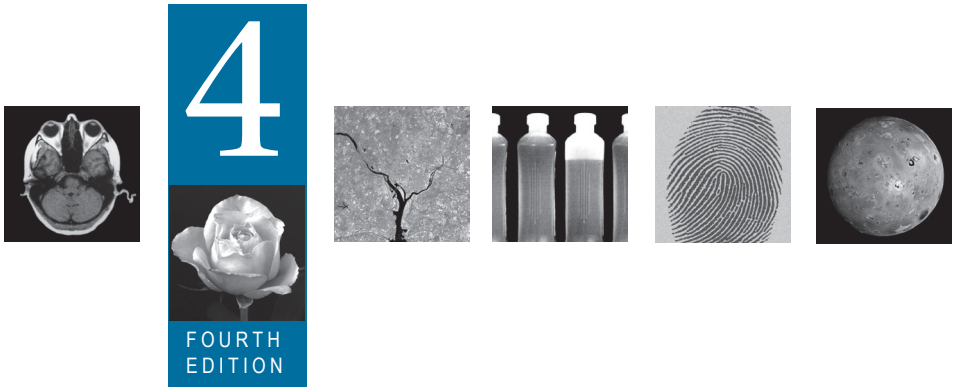
Digital Image Processing



Rafael C. Gonzalez
University of Tennessee

Richard E. Woods
Interapptics

Pearson
330 Hudson Street, New York, NY 10013

Senior Vice President Courseware Portfolio Management: Marcia J. Horton
Director, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Portfolio Manager: Julie Bai
Field Marketing Manager: Demetrius Hall
Product Marketing Manager: Yvonne Vannatta
Marketing Assistant: Jon Bryant
Content Managing Producer, ECS and Math: Scott Disanno
Content Producer: Michelle Bayman
Project Manager: Rose Kernan
Operations Specialist: Maura Zaldivar-Garcia
Manager, Rights and Permissions: Ben Ferrini
Cover Designer: Black Horse Designs
Cover Photo: **MRI image**—Author supplied; **Rose**—Author supplied; **Satellite image of Washington, D.C.**—Courtesy of NASA; **Bottles**—Author supplied; **Fingerprint**—Courtesy of the National Institute of Standards and Technology; **Moon IO of Jupiter**—Courtesy of NASA
Composition: Richard E. Woods

Pearson Education Ltd., London
Pearson Education Singapore, Pte. Ltd
Pearson Education Canada, Inc.
Pearson Education Japan
Pearson Education Australia PTY, Ltd
Pearson Education North Asia, Ltd., Hong Kong
Pearson Education de Mexico, S.A. de C.V.
Pearson Education Malaysia, Pte. Ltd.

Pearson Education, Inc., Hoboken

Library of Congress Cataloging-in-Publication Data

Names: Gonzalez, Rafael C., author. | Woods, Richard E. (Richard Eugene), author.
Title: Digital Image Processing / Rafael C. Gonzalez, University of Tennessee, Richard E. Woods, Interapptics.
Description: New York, NY : Pearson, [2018] | Includes bibliographical references and index.
Identifiers: LCCN 2017001581 | ISBN 9780133356724
Subjects: LCSH: Image processing--Digital techniques.
Classification: LCC TA1632 .G66 2018 | DDC 621.36/7--dc23
LC record available at https://lccn.loc.gov/2017001581

ISBN-10: 0-13-335672-8
1 16 ISBN-13: 9780133356724

Pearson www.pearsonhighered.com

To Connie, Ralph, and Rob
and
To Janice, David, and Jonathan

Contents

Preface

When something can be read without effort, great effort has gone into its writing.
Enrique Jardiel Poncela

This edition of *Digital Image Processing* is a major revision of the book. As in the 1977 and 1987 editions by Gonzalez and Wintz, and the 1992, 2002, and 2008 editions by Gonzalez and Woods, this sixth-generation edition was prepared with students and instructors in mind. The principal objectives of the book continue to be to provide an introduction to basic concepts and methodologies applicable to digital image processing, and to develop a foundation that can be used as the basis for further study and research in this field. To achieve these objectives, we focused again on material that we believe is fundamental and whose scope of application is not limited to the solution of specialized problems. The mathematical complexity of the book remains at a level well within the grasp of college seniors and first-year graduate students who have introductory preparation in mathematical analysis, vectors, matrices, probability, statistics, linear systems, and computer programming. The book website provides tutorials to support readers needing a review of this background material.

One of the principal reasons this book has been the world leader in its field for 40 years is the level of attention we pay to the changing educational needs of our readers. The present edition is based on an extensive survey that involved faculty, students, and independent readers of the book in 150 institutions from 30 countries. The survey revealed a need for coverage of new material that has matured since the last edition of the book. The principal findings of the survey indicated a need for:

- New material related to histogram matching.
- Expanded coverage of the fundamentals of spatial filtering.
- A more comprehensive and cohesive coverage of image transforms.
- A more complete presentation of finite differences, with a focus on edge detection.
- A discussion of clustering, superpixels, and their use in region segmentation.
- New material on active contours that includes snakes and level sets, and their use in image segmentation.
- Coverage of maximally stable extremal regions.
- Expanded coverage of feature extraction to include the Scale Invariant Feature Transform (SIFT).
- Expanded coverage of neural networks to include deep neural networks, backpropagation, deep learning, and, especially, deep convolutional neural networks.
- More homework problems at the end of the chapters.
- MATLAB computer projects.

The new and reorganized material that resulted in the present edition is our attempt at providing a reasonable balance between rigor, clarity of presentation,

and the findings of the survey. In addition to new material, earlier portions of the text were updated and clarified. This edition contains 425 new images, 135 new drawings, and 220 new exercises. For the first time, we have included MATLAB projects at the end of every chapter. In total there are 120 new MATLAB projects that cover a broad range of the material in the book. Although the solutions we provide are in MATLAB, the projects themselves are written in such a way that they can be implemented in other languages. Projects are an important addition because they will allow students to experiment with material they learn in the classroom. A large database of digital images is provided for this purpose.

New to This Edition

The highlights of this edition are as follows.

Chapter 1: Some figures were updated, and parts of the text were rewritten to correspond to changes in later chapters.

Chapter 2: Many of the sections and examples were rewritten for clarity. We added a new section dealing with random numbers and probability, with an emphasis on their application to image processing. We included 12 new examples, 31 new images, 22 new drawings, 32 new exercises, and 10 new MATLAB projects.

Chapter 3: Major revisions of the topics in this chapter include a new section on exact histogram matching. Fundamental concepts of spatial filtering were rewritten to include a discussion on separable filter kernels, expanded coverage of the properties of lowpass Gaussian kernels, and expanded coverage of highpass, bandreject, and bandpass filters, including numerous new examples that illustrate their use. In addition to revisions in the text, including 6 new examples, the chapter has 67 new images, 18 new line drawings, 31 new exercises, and 10 new MATLAB projects.

Chapter 4: Several of the sections of this chapter were revised to improve the clarity of presentation. We replaced dated graphical material with 35 new images and 4 new line drawings. We added 25 new exercises and 10 new MATLAB projects.

Chapter 5: Revisions to this chapter were limited to clarifications and a few corrections in notation. We added 6 new images, 17 new exercises, and 10 new MATLAB projects.

Chapter 6: This is a new chapter that brings together wavelets, several new transforms, and many of the image transforms that were scattered throughout the book. The emphasis of this new chapter is on the presentation of these transforms from a unified point of view. We added 24 new images, 20 new drawings, 25 new exercises and 10 new MATLAB projects.

Chapter 7: The material dealing with color image processing was moved to this chapter. Several sections were clarified, and the explanation of the CMY and CMYK color models was expanded. We added 2 new images and 10 new MATLAB projects.

Chapter 8: In addition to numerous clarifications and minor improvements to the presentation, we added 10 new MATLAB projects to this chapter.

Chapter 9: Revisions of this chapter included a complete rewrite of several sections, including redrafting of several line drawings. We added 18 new exercises and 10 new MATLAB projects.

Chapter 10: Several of the sections were rewritten for clarity. We updated the chapter by adding coverage of finite differences, K-means clustering, superpixels, and graph cuts. The new topics are illustrated with 4 new examples. In total, we added 31 new images, 3 new drawings, 8 new exercises, and 10 new MATLAB projects.

Chapter 11: This is a new chapter dealing with active contours for image segmentation, including snakes and level sets. An important feature in this chapter is that it presents a derivation of the fundamental snake equation. Similarly, we provide a derivation of the level set equation. Both equations are derived starting from basic principles, and the methods are illustrated with numerous examples. The strategy when we prepared this chapter was to bring this material to a level that could be understood by beginners in our field. To that end, we complemented the text material with 17 new examples, 141 new images, 19 new drawings, 37 new problems, and 10 new MATLAB projects.

Chapter 12: This is the chapter on feature extraction, which was moved from its 11th position in the previous edition. The chapter was updated with numerous topics, beginning with a more detailed classification of feature types and their uses. In addition to improvements in the clarity of presentation, we added coverage of slope change codes, expanded the explanation of skeletons, medial axes, and the distance transform, and added several new basic descriptors of compactness, circularity, and eccentricity. New material includes coverage of the Harris-Stephens corner detector, and a presentation of maximally stable extremal regions. A major addition to the chapter is a comprehensive discussion dealing with the Scale-Invariant Feature Transform (SIFT). The new material is complemented by 65 new images, 15 new drawings, 4 new examples, and 15 new exercises. We also added 10 new MATLAB projects.

Chapter 13: This is the image pattern classification chapter that was Chapter 12 in the previous edition. This chapter underwent a major revision to include an extensive rewrite of neural networks and deep learning, an area that has grown significantly since the last edition of the book. We added a comprehensive discussion on fully connected, deep neural networks that includes derivation of backpropagation starting from basic principles. The equations of backpropagation were expressed in "traditional" scalar terms, and then generalized into a compact set of matrix equations ideally suited for implementation of deep neural nets. The effectiveness of fully connected networks was demonstrated with several examples that included a comparison with the Bayes classifier. One of the most-requested topics in the survey was coverage of deep convolutional neural networks. We added an extensive section on this, following the same blueprint we used for deep, fully connected nets. That is, we derived the equations of backpropagation for convolutional nets, and showed how they are different from "traditional" backpropagation. We then illustrated the use of convolutional networks with simple images, and applied them to large image

databases of numerals and natural scenes. The written material is complemented by 23 new images, 28 new drawings, and 12 new exercises. We also included 10 new MATLAB projects.

Also for the first time, we have created student and faculty support packages that can be downloaded from the book website. The *Student Support Package* contains all the original images in the book, answers to selected exercises, detailed answers (including MATLAB code) to selected MATLAB projects, and instructions for using a set of utility functions that complement the projects. The *Faculty Support Package* contains solutions to all exercises and projects, teaching suggestions, and all the art in the book in the form of modifiable PowerPoint slides. One support package is made available with every new book, free of charge.

MATLAB projects are structured in a unique way that gives instructors significant flexibility in how projects are assigned. The MATLAB functions required to solve all the projects in the book are provided in executable, p-code format. These functions run just like the original functions, but the source code is not visible, and the files cannot be modified. The availability of these functions as a complete package makes it possible for projects to be assigned solely for the purpose of experimenting with image processing concepts, without having to write a single line of code. In other words, the complete set of MATLAB functions is available as a stand-alone p-code toolbox, ready to use without further development. When instructors elect to assign projects that involve MATLAB code development, we provide students enough answers to form a good base that they can expand, thus gaining experience with developing software solutions to image processing problems. Instructors have access to detailed answers to all projects.

The book website, established during the launch of the 2002 edition, continues to be a success, attracting more than 25,000 visitors each month. The site was upgraded for the launch of this edition. For more details on site features and content, see *The Book Website*, following the *Acknowledgments* section.

This edition of *Digital Image Processing* is a reflection of how the educational needs of our readers have changed since 2008. As is usual in an endeavor such as this, progress in the field continues after work on the manuscript stops. One of the reasons why this book has been so well accepted since it first appeared in 1977 is its continued emphasis on fundamental concepts that retain their relevance over time. This approach, among other things, attempts to provide a measure of stability in a rapidly evolving body of knowledge. We have tried to follow the same principle in preparing this edition of the book.

R.C.G.
R.E.W.

Acknowledgments

We are indebted to a number of individuals in academic circles, industry, and government who have contributed to this edition of the book. In particular, we wish to extend our appreciation to Hairong Qi and her students, Zhifei Zhang and Chengcheng Li, for their valuable review of the material on neural networks, and for their help in generating examples for that material. We also want to thank Ernesto Bribiesca Correa for providing and reviewing material on slope chain codes, and Dirk Padfield for his many suggestions and review of several chapters in the book. We appreciate Michel Kocher's many thoughtful comments and suggestions over the years on how to improve the book. Thanks also to Steve Eddins for his suggestions on MATLAB and related software issues, and to Dino Colcuc for his review of the material on exact histogram specification.

Numerous individuals have contributed to material carried over from the previous to the current edition of the book. Their contributions have been important in so many different ways that we find it difficult to acknowledge them in any other way but alphabetically. We thank Mongi A. Abidi, Yongmin Kim, Bryan Morse, Andrew Oldroyd, Ali M. Reza, Edgardo Felipe Riveron, Jose Ruiz Shulcloper, and Cameron H.G. Wright for their many suggestions on how to improve the presentation and/or the scope of coverage in the book. We are also indebted to Naomi Fernandes at the MathWorks for providing us with MATLAB software and support that were important in our ability to create many of the examples and experimental results included in this edition of the book.

A significant percentage of the new images used in this edition (and in some cases their history and interpretation) were obtained through the efforts of individuals whose contributions are sincerely appreciated. In particular, we wish to acknowledge the efforts of Serge Beucher, Uwe Boos, Michael E. Casey, Michael W. Davidson, Susan L. Forsburg, Thomas R. Gest, Daniel A. Hammer, Zhong He, Roger Heady, Juan A. Herrera, John M. Hudak, Michael Hurwitz, Chris J. Johannsen, Rhonda Knighton, Don P. Mitchell, A. Morris, Curtis C. Ober, David. R. Pickens, Michael Robinson, Michael Shaffer, Pete Sites, Sally Stowe, Craig Watson, David K. Wehe, and Robert A. West. We also wish to acknowledge other individuals and organizations cited in the captions of numerous figures throughout the book for their permission to use that material.

We also thank Scott Disanno, Michelle Bayman, Rose Kernan, and Julie Bai for their support and significant patience during the production of the book.

R.C.G.
R.E.W.

The Book Website

www.ImageProcessingPlace.com

Digital Image Processing is a completely self-contained book. However, the companion website offers additional support in a number of important areas.

For the Student or Independent Reader the site contains

- Reviews in areas such as probability, statistics, vectors, and matrices.
- A Tutorials section containing dozens of tutorials on topics relevant to the material in the book.
- An image database containing all the images in the book, as well as many other image databases.

For the Instructor the site contains

- An Instructor's Manual with complete solutions to all the problems and MATLAB projects in the book, as well as course and laboratory teaching guidelines. The manual is available free of charge to instructors who have adopted the book for classroom use.
- Classroom presentation materials in PowerPoint format.
- Material removed from previous editions, downloadable in convenient PDF format.
- Numerous links to other educational resources.

For the Practitioner the site contains additional specialized topics such as

- Links to commercial sites.
- Selected new references.
- Links to commercial image databases.

The website is an ideal tool for keeping the book current between editions by including new topics, digital images, and other relevant material that has appeared after the book was published. Although considerable care was taken in the production of the book, the website is also a convenient repository for any errors discovered between printings.

The DIP4E Support Packages

In this edition, we created support packages for students and faculty to organize all the classroom support materials available for the new edition of the book into one easy download. The Student Support Package contains all the original images in the book, answers to selected exercises, detailed answers (including MATLAB code) to selected MATLAB projects, and instructions for using a set of utility functions that complement the projects. The Faculty Support Package contains solutions to all exercises and projects, teaching suggestions, and all the art in the book in modifiable PowerPoint slides. One support package is made available with every new book, free of charge. Applications for the support packages are submitted at the book website.

About the Authors

RAFAEL C. GONZALEZ

R. C. Gonzalez received the B.S.E.E. degree from the University of Miami in 1965 and the M.E. and Ph.D. degrees in electrical engineering from the University of Florida, Gainesville, in 1967 and 1970, respectively. He joined the Electrical and Computer Science Department at the University of Tennessee, Knoxville (UTK) in 1970, where he became Associate Professor in 1973, Professor in 1978, and Distinguished Service Professor in 1984. He served as Chairman of the department from 1994 through 1997. He is currently a Professor Emeritus at UTK.

Gonzalez is the founder of the Image & Pattern Analysis Laboratory and the Robotics & Computer Vision Laboratory at the University of Tennessee. He also founded Perceptics Corporation in 1982 and was its president until 1992. The last three years of this period were spent under a full-time employment contract with Westinghouse Corporation, who acquired the company in 1989.

Under his direction, Perceptics became highly successful in image processing, computer vision, and laser disk storage technology. In its initial ten years, Perceptics introduced a series of innovative products, including: The world's first commercially available computer vision system for automatically reading license plates on moving vehicles; a series of large-scale image processing and archiving systems used by the U.S. Navy at six different manufacturing sites throughout the country to inspect the rocket motors of missiles in the Trident II Submarine Program; the market-leading family of imaging boards for advanced Macintosh computers; and a line of trillion-byte laser disk products.

He is a frequent consultant to industry and government in the areas of pattern recognition, image processing, and machine learning. His academic honors for work in these fields include the 1977 UTK College of Engineering Faculty Achievement Award; the 1978 UTK Chancellor's Research Scholar Award; the 1980 Magnavox Engineering Professor Award; and the 1980 M.E. Brooks Distinguished Professor Award. In 1981 he became an IBM Professor at the University of Tennessee and in 1984 he was named a Distinguished Service Professor there. He was awarded a Distinguished Alumnus Award by the University of Miami in 1985, the Phi Kappa Phi Scholar Award in 1986, and the University of Tennessee's Nathan W. Dougherty Award for Excellence in Engineering in 1992.

Honors for industrial accomplishment include the 1987 IEEE Outstanding Engineer Award for Commercial Development in Tennessee; the 1988 Albert Rose National Award for Excellence in Commercial Image Processing; the 1989 B. Otto Wheeley Award for Excellence in Technology Transfer; the 1989 Coopers and Lybrand Entrepreneur of the Year Award; the 1992 IEEE Region 3 Outstanding Engineer Award; and the 1993 Automated Imaging Association National Award for Technology Development.

Gonzalez is author or co-author of over 100 technical articles, two edited books, and four textbooks in the fields of pattern recognition, image processing, and robotics. His books are used in over 1000 universities and research institutions throughout

the world. He is listed in the prestigious *Marquis Who's Who in America*, *Marquis Who's Who in Engineering*, *Marquis Who's Who in the World*, and in 10 other national and international biographical citations. He is the co-holder of two U.S. Patents, and has been an associate editor of the *IEEE Transactions on Systems, Man and Cybernetics*, and the *International Journal of Computer and Information Sciences*. He is a member of numerous professional and honorary societies, including Tau Beta Pi, Phi Kappa Phi, Eta Kappa Nu, and Sigma Xi. He is a Fellow of the IEEE.

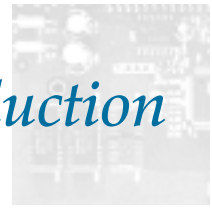
RICHARD E. WOODS

R. E. Woods earned his B.S., M.S., and Ph.D. degrees in Electrical Engineering from the University of Tennessee, Knoxville in 1975, 1977, and 1980, respectively. He became an Assistant Professor of Electrical Engineering and Computer Science in 1981 and was recognized as a Distinguished Engineering Alumnus in 1986.

A veteran hardware and software developer, Dr. Woods has been involved in the founding of several high-technology startups, including Perceptics Corporation, where he was responsible for the development of the company's quantitative image analysis and autonomous decision-making products; MedData Interactive, a high-technology company specializing in the development of handheld computer systems for medical applications; and Interapptics, an internet-based company that designs desktop and handheld computer applications.

Dr. Woods currently serves on several nonprofit educational and media-related boards, including Johnson University, and was recently a summer English instructor at the Beijing Institute of Technology. He is the holder of a U.S. Patent in the area of digital image processing and has published two textbooks, as well as numerous articles related to digital signal processing. Dr. Woods is a member of several professional societies, including Tau Beta Pi, Phi Kappa Phi, and the IEEE.

1 Introduction

One picture is worth more than ten thousand words.

Anonymous

Preview

Interest in digital image processing methods stems from two principal application areas: improvement of pictorial information for human interpretation, and processing of image data for tasks such as storage, transmission, and extraction of pictorial information. This chapter has several objectives: (1) to define the scope of the field that we call image processing; (2) to give a historical perspective of the origins of this field; (3) to present an overview of the state of the art in image processing by examining some of the principal areas in which it is applied; (4) to discuss briefly the principal approaches used in digital image processing; (5) to give an overview of the components contained in a typical, general-purpose image processing system; and (6) to provide direction to the literature where image processing work is reported. The material in this chapter is extensively illustrated with a range of images that are representative of the images we will be using throughout the book.

Upon completion of this chapter, readers should:

- Understand the concept of a digital image.
- Have a broad overview of the historical underpinnings of the field of digital image processing.
- Understand the definition and scope of digital image processing.
- Know the fundamentals of the electromagnetic spectrum and its relationship to image generation.
- Be aware of the different fields in which digital image processing methods are applied.
- Be familiar with the basic processes involved in image processing.
- Be familiar with the components that make up a general-purpose digital image processing system.
- Be familiar with the scope of the literature where image processing work is reported.

1.1 WHAT IS DIGITAL IMAGE PROCESSING?

An image may be defined as a two-dimensional function, $f(x, y)$, where x and y are *spatial* (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the *intensity* or *gray level* of the image at that point. When x, y, and the intensity values of f are all finite, discrete quantities, we call the image a *digital image*. The field of *digital image processing* refers to processing digital images by means of a digital computer. Note that a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are called *picture elements*, *image elements*, *pels*, and *pixels*. *Pixel* is the term used most widely to denote the elements of a digital image. We will consider these definitions in more formal terms in Chapter 2.

Vision is the most advanced of our senses, so it is not surprising that images play the single most important role in human perception. However, unlike humans, who are limited to the visual band of the electromagnetic (EM) spectrum, imaging machines cover almost the entire EM spectrum, ranging from gamma to radio waves. They can operate on images generated by sources that humans are not accustomed to associating with images. These include ultrasound, electron microscopy, and computer-generated images. Thus, digital image processing encompasses a wide and varied field of applications.

There is no general agreement among authors regarding where image processing stops and other related areas, such as *image analysis* and *computer vision*, start. Sometimes, a distinction is made by defining image processing as a discipline in which both the input and output of a process are images. We believe this to be a limiting and somewhat artificial boundary. For example, under this definition, even the trivial task of computing the average intensity of an image (which yields a single number) would not be considered an image processing operation. On the other hand, there are fields such as computer vision whose ultimate goal is to use computers to emulate human vision, including learning and being able to make inferences and take actions based on visual inputs. This area itself is a branch of *artificial intelligence* (AI) whose objective is to emulate human intelligence. The field of AI is in its earliest stages of infancy in terms of development, with progress having been much slower than originally anticipated. The area of image analysis (also called *image understanding*) is in between image processing and computer vision.

There are no clear-cut boundaries in the continuum from image processing at one end to computer vision at the other. However, one useful paradigm is to consider three types of computerized processes in this continuum: low-, mid-, and high-level processes. Low-level processes involve primitive operations such as image preprocessing to reduce noise, contrast enhancement, and image sharpening. A low-level process is characterized by the fact that both its inputs and outputs are images. Mid-level processing of images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes extracted from those images (e.g., edges, contours, and the identity of individual objects). Finally, higher-level processing

involves "making sense" of an ensemble of recognized objects, as in image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with human vision.

Based on the preceding comments, we see that a logical place of overlap between image processing and image analysis is the area of recognition of individual regions or objects in an image. Thus, what we call in this book *digital image processing* encompasses processes whose inputs and outputs are images and, in addition, includes processes that extract attributes from images up to, and including, the recognition of individual objects. As an illustration to clarify these concepts, consider the area of automated analysis of text. The processes of acquiring an image of the area containing the text, preprocessing that image, extracting (segmenting) the individual characters, describing the characters in a form suitable for computer processing, and recognizing those individual characters are in the scope of what we call digital image processing in this book. Making sense of the content of the page may be viewed as being in the domain of image analysis and even computer vision, depending on the level of complexity implied by the statement "making sense of." As will become evident shortly, digital image processing, as we have defined it, is used routinely in a broad range of areas of exceptional social and economic value. The concepts developed in the following chapters are the foundation for the methods used in those application areas.

1.2 THE ORIGINS OF DIGITAL IMAGE PROCESSING

One of the earliest applications of digital images was in the newspaper industry, when pictures were first sent by submarine cable between London and New York. Introduction of the Bartlane cable picture transmission system in the early 1920s reduced the time required to transport a picture across the Atlantic from more than a week to less than three hours. Specialized printing equipment coded pictures for cable transmission, then reconstructed them at the receiving end. Figure 1.1 was transmitted in this way and reproduced on a telegraph printer fitted with typefaces simulating a halftone pattern.

Some of the initial problems in improving the visual quality of these early digital pictures were related to the selection of printing procedures and the distribution of

FIGURE 1.1 A digital picture produced in 1921 from a coded tape by a telegraph printer with special typefaces. (McFarlane.) [References in the bibliography at the end of the book are listed in alphabetical order by authors' last names.]

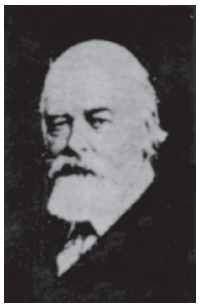


FIGURE 1.2
A digital picture made in 1922 from a tape punched after the signals had crossed the Atlantic twice. (McFarlane.)

intensity levels. The printing method used to obtain Fig. 1.1 was abandoned toward the end of 1921 in favor of a technique based on photographic reproduction made from tapes perforated at the telegraph receiving terminal. Figure 1.2 shows an image obtained using this method. The improvements over Fig. 1.1 are evident, both in tonal quality and in resolution.

The early Bartlane systems were capable of coding images in five distinct levels of gray. This capability was increased to 15 levels in 1929. Figure 1.3 is typical of the type of images that could be obtained using the 15-tone equipment. During this period, introduction of a system for developing a film plate via light beams that were modulated by the coded picture tape improved the reproduction process considerably.

Although the examples just cited involve digital images, they are not considered digital image processing results in the context of our definition, because digital computers were not used in their creation. Thus, the history of digital image processing is intimately tied to the development of the digital computer. In fact, digital images require so much storage and computational power that progress in the field of digital image processing has been dependent on the development of digital computers and of supporting technologies that include data storage, display, and transmission.

FIGURE 1.3
Unretouched cable picture of Generals Pershing (right) and Foch, transmitted in 1929 from London to New York by 15-tone equipment. (McFarlane.)

The concept of a computer dates back to the invention of the abacus in Asia Minor, more than 5000 years ago. More recently, there have been developments in the past two centuries that are the foundation of what we call a computer today. However, the basis for what we call a *modern* digital computer dates back to only the 1940s, with the introduction by John von Neumann of two key concepts: (1) a memory to hold a stored program and data, and (2) conditional branching. These two ideas are the foundation of a central processing unit (CPU), which is at the heart of computers today. Starting with von Neumann, there were a series of key advances that led to computers powerful enough to be used for digital image processing. Briefly, these advances may be summarized as follows: (1) the invention of the transistor at Bell Laboratories in 1948; (2) the development in the 1950s and 1960s of the high-level programming languages COBOL (Common Business-Oriented Language) and FORTRAN (Formula Translator); (3) the invention of the integrated circuit (IC) at Texas Instruments in 1958; (4) the development of operating systems in the early 1960s; (5) the development of the microprocessor (a single chip consisting of a CPU, memory, and input and output controls) by Intel in the early 1970s; (6) the introduction by IBM of the personal computer in 1981; and (7) progressive miniaturization of components, starting with large-scale integration (LI) in the late 1970s, then very-large-scale integration (VLSI) in the 1980s, to the present use of ultra-large-scale integration (ULSI) and experimental nonotechnologies. Concurrent with these advances were developments in the areas of mass storage and display systems, both of which are fundamental requirements for digital image processing.

The first computers powerful enough to carry out meaningful image processing tasks appeared in the early 1960s. The birth of what we call digital image processing today can be traced to the availability of those machines, and to the onset of the space program during that period. It took the combination of those two developments to bring into focus the potential of digital image processing for solving problems of practical significance. Work on using computer techniques for improving images from a space probe began at the Jet Propulsion Laboratory (Pasadena, California) in 1964, when pictures of the moon transmitted by *Ranger 7* were processed by a computer to correct various types of image distortion inherent in the on-board television camera. Figure 1.4 shows the first image of the moon taken by *Ranger 7* on July 31, 1964 at 9:09 A.M. Eastern Daylight Time (EDT), about 17 minutes before impacting the lunar surface (the markers, called *reseau marks*, are used for geometric corrections, as discussed in Chapter 2).This also is the first image of the moon taken by a U.S. spacecraft. The imaging lessons learned with *Ranger 7* served as the basis for improved methods used to enhance and restore images from the Surveyor missions to the moon, the *Mariner* series of flyby missions to Mars, the *Apollo* manned flights to the moon, and others.

In parallel with space applications, digital image processing techniques began in the late 1960s and early 1970s to be used in medical imaging, remote Earth resources observations, and astronomy. The invention in the early 1970s of *computerized axial tomography* (CAT), also called *computerized tomography* (CT) for short, is one of the most important events in the application of image processing in medical diagnosis. Computerized axial tomography is a process in which a ring of detectors

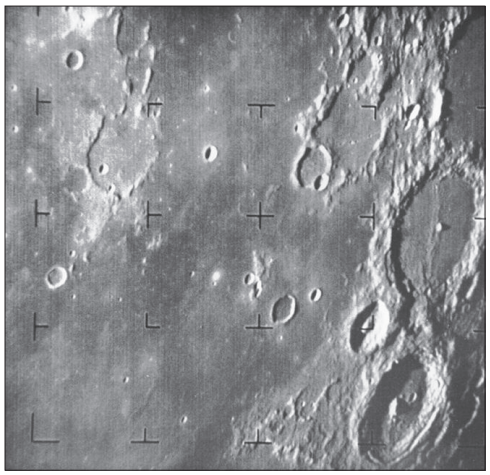


FIGURE 1.4
The first picture of the moon by a U.S. spacecraft. *Ranger 7* took this image on July 31, 1964 at 9:09 A.M. EDT, about 17 minutes before impacting the lunar surface. (Courtesy of NASA.)

encircles an object (or patient) and an X-ray source, concentric with the detector ring, rotates about the object. The X-rays pass through the object and are collected at the opposite end by the corresponding detectors in the ring. This procedure is repeated the source rotates. Tomography consists of algorithms that use the sensed data to construct an image that represents a "slice" through the object. Motion of the object in a direction perpendicular to the ring of detectors produces a set of such slices, which constitute a three-dimensional (3-D) rendition of the inside of the object. Tomography was invented independently by Sir Godfrey N. Hounsfield and Professor Allan M. Cormack, who shared the 1979 Nobel Prize in Medicine for their invention. It is interesting to note that X-rays were discovered in 1895 by Wilhelm Conrad Roentgen, for which he received the 1901 Nobel Prize for Physics. These two inventions, nearly 100 years apart, led to some of the most important applications of image processing today.

From the 1960s until the present, the field of image processing has grown vigorously. In addition to applications in medicine and the space program, digital image processing techniques are now used in a broad range of applications. Computer procedures are used to enhance the contrast or code the intensity levels into color for easier interpretation of X-rays and other images used in industry, medicine, and the biological sciences. Geographers use the same or similar techniques to study pollution patterns from aerial and satellite imagery. Image enhancement and restoration procedures are used to process degraded images of unrecoverable objects, or experimental results too expensive to duplicate. In archeology, image processing methods have successfully restored blurred pictures that were the only available records of rare artifacts lost or damaged after being photographed. In physics and related fields, computer techniques routinely enhance images of experiments in areas such as high-energy plasmas and electron microscopy. Similarly successful applications of image processing concepts can be found in astronomy, biology, nuclear medicine, law enforcement, defense, and industry.

These examples illustrate processing results intended for human interpretation. The second major area of application of digital image processing techniques mentioned at the beginning of this chapter is in solving problems dealing with machine perception. In this case, interest is on procedures for extracting information from an image, in a form suitable for computer processing. Often, this information bears little resemblance to visual features that humans use in interpreting the content of an image. Examples of the type of information used in machine perception are statistical moments, Fourier transform coefficients, and multidimensional distance measures. Typical problems in machine perception that routinely utilize image processing techniques are automatic character recognition, industrial machine vision for product assembly and inspection, military recognizance, automatic processing of fingerprints, screening of X-rays and blood samples, and machine processing of aerial and satellite imagery for weather prediction and environmental assessment. The continuing decline in the ratio of computer price to performance, and the expansion of networking and communication bandwidth via the internet, have created unprecedented opportunities for continued growth of digital image processing. Some of these application areas will be illustrated in the following section.

1.3 EXAMPLES OF FIELDS THAT USE DIGITAL IMAGE PROCESSING

Today, there is almost no area of technical endeavor that is not impacted in some way by digital image processing. We can cover only a few of these applications in the context and space of the current discussion. However, limited as it is, the material presented in this section will leave no doubt in your mind regarding the breadth and importance of digital image processing. We show in this section numerous areas of application, each of which routinely utilizes the digital image processing techniques developed in the following chapters. Many of the images shown in this section are used later in one or more of the examples given in the book. Most images shown are digital images.

The areas of application of digital image processing are so varied that some form of organization is desirable in attempting to capture the breadth of this field. One of the simplest ways to develop a basic understanding of the extent of image processing applications is to categorize images according to their source (e.g., X-ray, visual, infrared, and so on).The principal energy source for images in use today is the electromagnetic energy spectrum. Other important sources of energy include acoustic, ultrasonic, and electronic (in the form of electron beams used in electron microscopy). Synthetic images, used for modeling and visualization, are generated by computer. In this section we will discuss briefly how images are generated in these various categories, and the areas in which they are applied. Methods for converting images into digital form will be discussed in the next chapter.

Images based on radiation from the EM spectrum are the most familiar, especially images in the X-ray and visual bands of the spectrum. Electromagnetic waves can be conceptualized as propagating sinusoidal waves of varying wavelengths, or they can be thought of as a stream of massless particles, each traveling in a wavelike pattern and moving at the speed of light. Each massless particle contains a certain amount (or bundle) of energy. Each bundle of energy is called a *photon*. If spectral

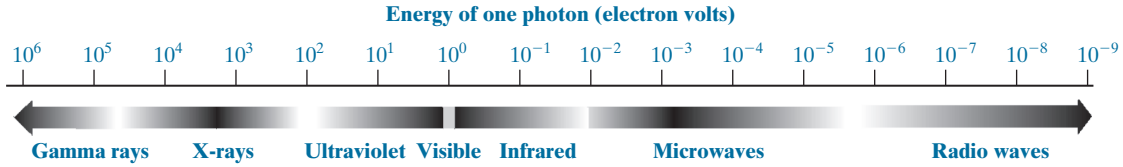


FIGURE 1.5 The electromagnetic spectrum arranged according to energy per photon.

bands are grouped according to energy per photon, we obtain the spectrum shown in Fig. 1.5, ranging from gamma rays (highest energy) at one end to radio waves (lowest energy) at the other. The bands are shown shaded to convey the fact that bands of the EM spectrum are not distinct, but rather transition smoothly from one to the other.

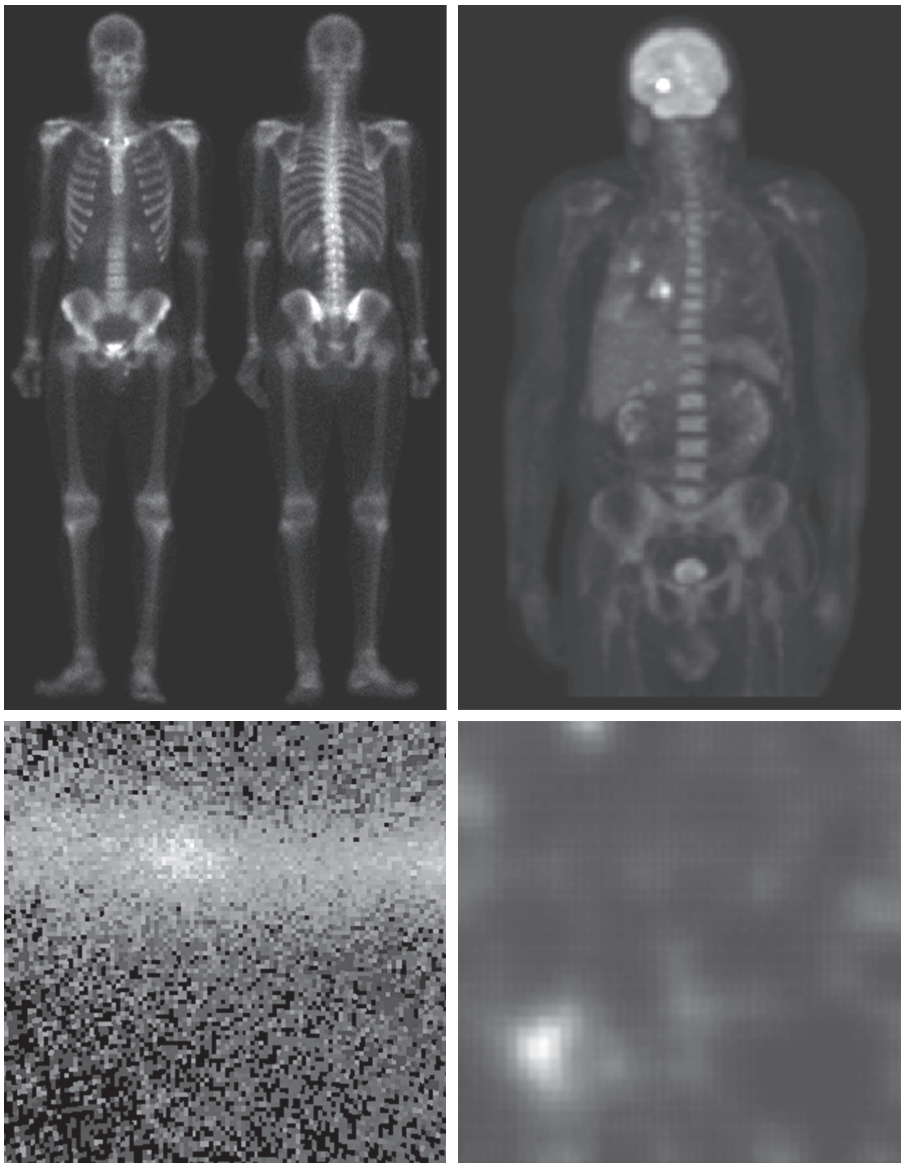
GAMMA-RAY IMAGING

Major uses of imaging based on gamma rays include nuclear medicine and astronomical observations. In nuclear medicine, the approach is to inject a patient with a radioactive isotope that emits gamma rays as it decays. Images are produced from the emissions collected by gamma-ray detectors. Figure 1.6(a) shows an image of a complete bone scan obtained by using gamma-ray imaging. Images of this sort are used to locate sites of bone pathology, such as infections or tumors. Figure 1.6(b) shows another major modality of nuclear imaging called *positron emission tomography* (PET). The principle is the same as with X-ray tomography, mentioned briefly in Section 1.2. However, instead of using an external source of X-ray energy, the patient is given a radioactive isotope that emits positrons as it decays. When a positron meets an electron, both are annihilated and two gamma rays are given off. These are detected and a tomographic image is created using the basic principles of tomography. The image shown in Fig. 1.6(b) is one sample of a sequence that constitutes a 3-D rendition of the patient. This image shows a tumor in the brain and another in the lung, easily visible as small white masses.

A star in the constellation of Cygnus exploded about 15,000 years ago, generating a superheated, stationary gas cloud (known as the Cygnus Loop) that glows in a spectacular array of colors. Figure 1.6(c) shows an image of the Cygnus Loop in the gamma-ray band. Unlike the two examples in Figs. 1.6(a) and (b), this image was obtained using the natural radiation of the object being imaged. Finally, Fig. 1.6(d) shows an image of gamma radiation from a valve in a nuclear reactor. An area of strong radiation is seen in the lower left side of the image.

X-RAY IMAGING

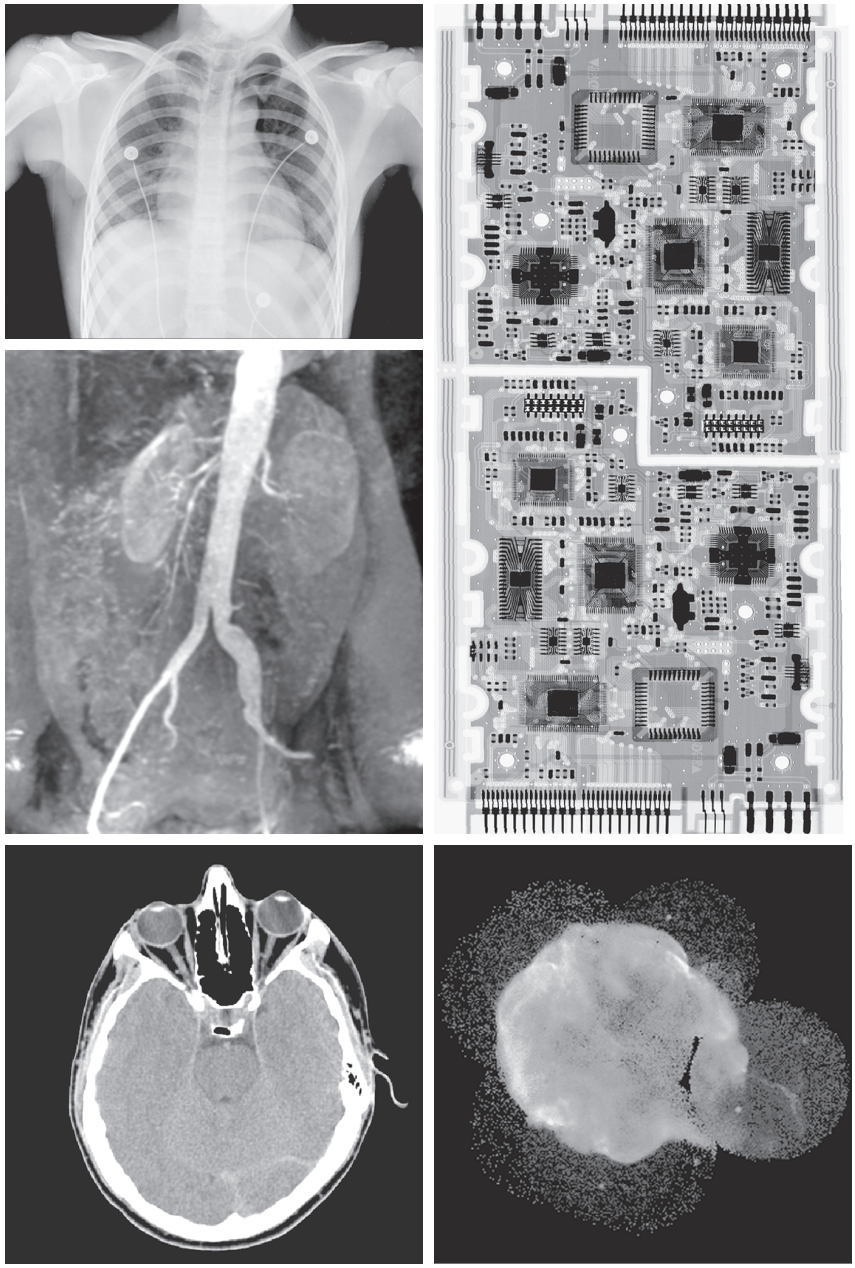
X-rays are among the oldest sources of EM radiation used for imaging. The best known use of X-rays is medical diagnostics, but they are also used extensively in industry and other areas, such as astronomy. X-rays for medical and industrial imaging are generated using an X-ray tube, which is a vacuum tube with a cathode and anode. The cathode is heated, causing free electrons to be released. These electrons flow at high speed to the positively charged anode. When the electrons strike a



a b
c d

FIGURE 1.6
Examples of gamma-ray imaging. (a) Bone scan. (b) PET image. (c) Cygnus Loop. (d) Gamma radiation (bright spot) from a reactor valve. (Images courtesy of (a) G.E. Medical Systems; (b) Dr. Michael E. Casey, CTI PET Systems; (c) NASA; (d) Professors Zhong He and David K. Wehe, University of Michigan.)

nucleus, energy is released in the form of X-ray radiation. The energy (penetrating power) of X-rays is controlled by a voltage applied across the anode, and by a current applied to the filament in the cathode. Figure 1.7(a) shows a familiar chest X-ray generated simply by placing the patient between an X-ray source and a film sensitive to X-ray energy. The intensity of the X-rays is modified by absorption as they pass through the patient, and the resulting energy falling on the film develops it, much in the same way that light develops photographic film. In digital radiography,



a d
c
b e

FIGURE 1.7
Examples of X-ray imaging. (a) Chest X-ray. (b) Aortic angiogram. (c) Head CT. (d) Circuit boards. (e) Cygnus Loop. (Images courtesy of (a) and (c) Dr. David R. Pickens, Dept. of Radiology & Radiological Sciences, Vanderbilt University Medical Center; (b) Dr. Thomas R. Gest, Division of Anatomical Sciences, Univ. of Michigan Medical School; (d) Mr. Joseph E. Pascente, Lixi, Inc.; and (e) NASA.)

digital images are obtained by one of two methods: (1) by digitizing X-ray films; or; (2) by having the X-rays that pass through the patient fall directly onto devices (such as a phosphor screen) that convert X-rays to light. The light signal in turn is captured by a light-sensitive digitizing system. We will discuss digitization in more detail in Chapters 2 and 4.

Angiography is another major application in an area called contrast enhancement radiography. This procedure is used to obtain images of blood vessels, called *angiograms*. A catheter (a small, flexible, hollow tube) is inserted, for example, into an artery or vein in the groin. The catheter is threaded into the blood vessel and guided to the area to be studied. When the catheter reaches the site under investigation, an X-ray contrast medium is injected through the tube. This enhances the contrast of the blood vessels and enables a radiologist to see any irregularities or blockages. Figure 1.7(b) shows an example of an aortic angiogram. The catheter can be seen being inserted into the large blood vessel on the lower left of the picture. Note the high contrast of the large vessel as the contrast medium flows up in the direction of the kidneys, which are also visible in the image. As we will discuss further in Chapter 2, angiography is a major area of digital image processing, where image subtraction is used to further enhance the blood vessels being studied.

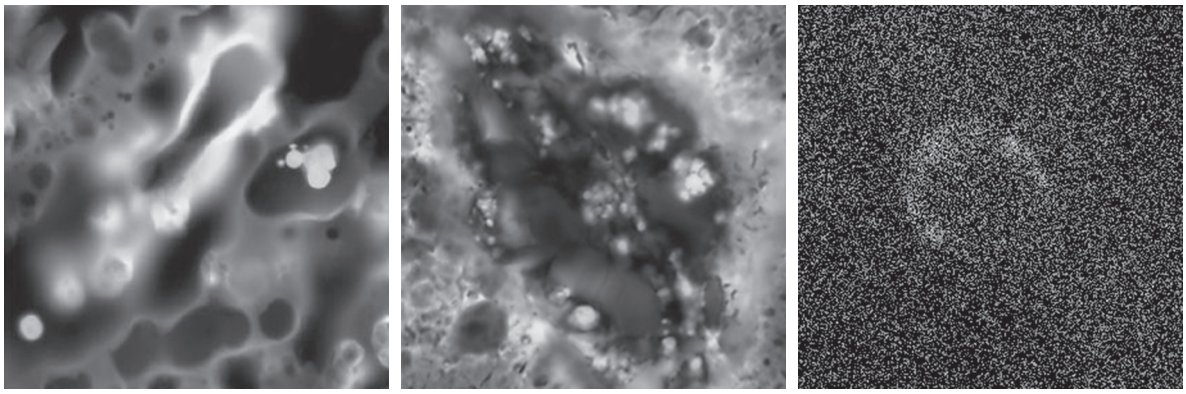
Another important use of X-rays in medical imaging is computerized axial tomography (CAT). Due to their resolution and 3-D capabilities, CAT scans revolutionized medicine from the moment they first became available in the early 1970s. As noted in Section 1.2, each CAT image is a "slice" taken perpendicularly through the patient. Numerous slices are generated as the patient is moved in a longitudinal direction. The ensemble of such images constitutes a 3-D rendition of the inside of the body, with the longitudinal resolution being proportional to the number of slice images taken. Figure 1.7(c) shows a typical CAT slice image of a human head.

Techniques similar to the ones just discussed, but generally involving higher energy X-rays, are applicable in industrial processes. Figure 1.7(d) shows an X-ray image of an electronic circuit board. Such images, representative of literally hundreds of industrial applications of X-rays, are used to examine circuit boards for flaws in manufacturing, such as missing components or broken traces. Industrial CAT scans are useful when the parts can be penetrated by X-rays, such as in plastic assemblies, and even large bodies, such as solid-propellant rocket motors. Figure 1.7(e) shows an example of X-ray imaging in astronomy. This image is the Cygnus Loop of Fig. 1.6(c), but imaged in the X-ray band.

IMAGING IN THE ULTRAVIOLET BAND

Applications of ultraviolet "light" are varied. They include lithography, industrial inspection, microscopy, lasers, biological imaging, and astronomical observations. We illustrate imaging in this band with examples from microscopy and astronomy.

Ultraviolet light is used in fluorescence microscopy, one of the fastest growing areas of microscopy. Fluorescence is a phenomenon discovered in the middle of the nineteenth century, when it was first observed that the mineral fluorspar fluoresces when ultraviolet light is directed upon it. The ultraviolet light itself is not visible, but when a photon of ultraviolet radiation collides with an electron in an atom of a fluorescent material, it elevates the electron to a higher energy level. Subsequently, the excited electron relaxes to a lower level and emits light in the form of a lower-energy photon in the visible (red) light region. Important tasks performed with a fluorescence microscope are to use an excitation light to irradiate a prepared specimen, and then to separate the much weaker radiating fluorescent light from the brighter



a b c

FIGURE 1.8 Examples of ultraviolet imaging. (a) Normal corn. (b) Corn infected by smut. (c) Cygnus Loop. (Images (a) and (b) courtesy of Dr. Michael W. Davidson, Florida State University, (c) NASA.)

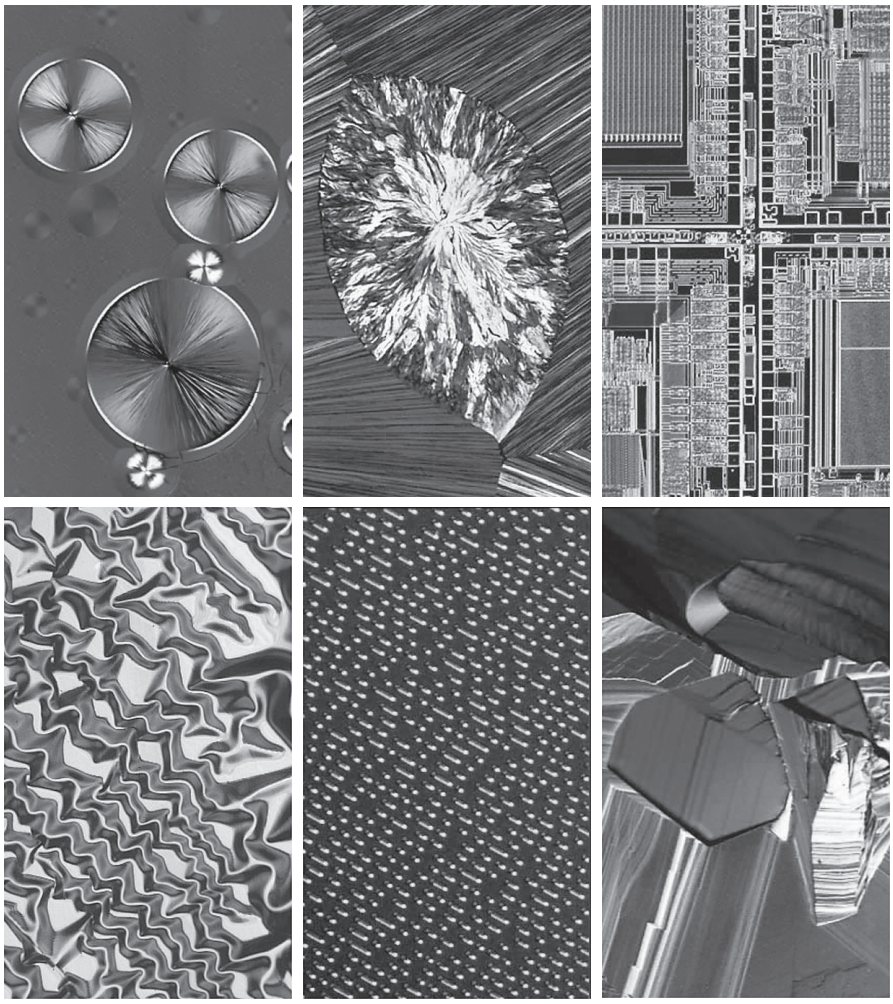
excitation light. Thus, only the emission light reaches the eye or other detector. The resulting fluorescing areas shine against a dark background with sufficient contrast to permit detection. The darker the background of the nonfluorescing material, the more efficient the instrument.

Fluorescence microscopy is an excellent method for studying materials that can be made to fluoresce, either in their natural form (primary fluorescence) or when treated with chemicals capable of fluorescing (secondary fluorescence). Figures 1.8(a) and (b) show results typical of the capability of fluorescence microscopy. Figure 1.8(a) shows a fluorescence microscope image of normal corn, and Fig. 1.8(b) shows corn infected by "smut," a disease of cereals, corn, grasses, onions, and sorghum that can be caused by any one of more than 700 species of parasitic fungi. Corn smut is particularly harmful because corn is one of the principal food sources in the world. As another illustration, Fig. 1.8(c) shows the Cygnus Loop imaged in the high-energy region of the ultraviolet band.

IMAGING IN THE VISIBLE AND INFRARED BANDS

Considering that the visual band of the electromagnetic spectrum is the most familiar in all our activities, it is not surprising that imaging in this band outweighs by far all the others in terms of breadth of application. The infrared band often is used in conjunction with visual imaging, so we have grouped the visible and infrared bands in this section for the purpose of illustration. We consider in the following discussion applications in light microscopy, astronomy, remote sensing, industry, and law enforcement.

Figure 1.9 shows several examples of images obtained with a light microscope. The examples range from pharmaceuticals and microinspection to materials characterization. Even in microscopy alone, the application areas are too numerous to detail here. It is not difficult to conceptualize the types of processes one might apply to these images, ranging from enhancement to measurements.



a b c
d e f

FIGURE 1.9
Examples of light microscopy images. (a) Taxol (anticancer agent), magnified 250×. (b) Cholesterol—40×. (c) Microprocessor—60×. (d) Nickel oxide thin film—600×. (e) Surface of audio CD—1750×. (f) Organic superconductor— 450×. (Images courtesy of Dr. Michael W. Davidson, Florida State University.)

Another major area of visual processing is remote sensing, which usually includes several bands in the visual and infrared regions of the spectrum. Table 1.1 shows the so-called *thematic bands* in NASA's LANDSAT satellites. The primary function of LANDSAT is to obtain and transmit images of the Earth from space, for purposes of monitoring environmental conditions on the planet. The bands are expressed in terms of wavelength, with 1μm being equal to 10^{-6} m (we will discuss the wavelength regions of the electromagnetic spectrum in more detail in Chapter 2). Note the characteristics and uses of each band in Table 1.1.

In order to develop a basic appreciation for the power of this type of multispectral imaging, consider Fig. 1.10, which shows one image for each of the spectral bands in Table 1.1. The area imaged is Washington D.C., which includes features such as buildings, roads, vegetation, and a major river (the Potomac) going though the city.

TABLE 1.1
Thematic bands of NASA's LANDSAT satellite.

Band No.	Name	Wavelength (μm)	Characteristics and Uses
1	Visible blue	0.45–0.52	Maximum water penetration
2	Visible green	0.53–0.61	Measures plant vigor
3	Visible red	0.63–0.69	Vegetation discrimination
4	Near infrared	0.78–0.90	Biomass and shoreline mapping
5	Middle infrared	1.55–1.75	Moisture content: soil/vegetation
6	Thermal infrared	10.4–12.5	Soil moisture; thermal mapping
7	Short-wave infrared	2.09–2.35	Mineral mapping

Images of population centers are used over time to assess population growth and shift patterns, pollution, and other factors affecting the environment. The differences between visual and infrared image features are quite noticeable in these images. Observe, for example, how well defined the river is from its surroundings in Bands 4 and 5.

Weather observation and prediction also are major applications of multispectral imaging from satellites. For example, Fig. 1.11 is an image of Hurricane Katrina, one of the most devastating storms in recent memory in the Western Hemisphere. This image was taken by a National Oceanographic and Atmospheric Administration (NOAA) satellite using sensors in the visible and infrared bands. The eye of the hurricane is clearly visible in this image.

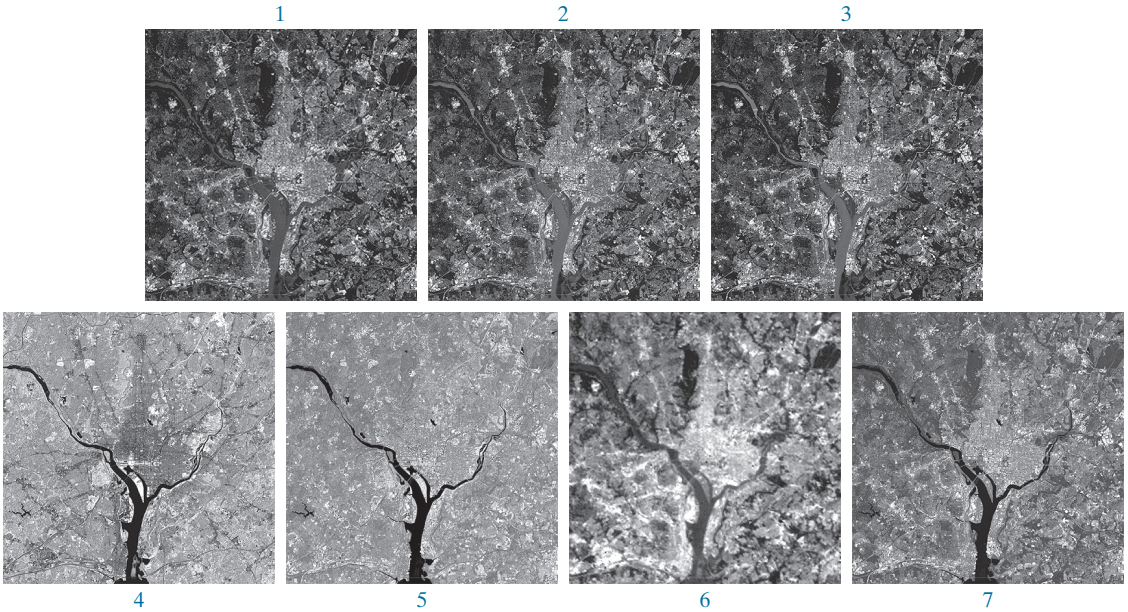


FIGURE 1.10 LANDSAT satellite images of the Washington, D.C. area. The numbers refer to the thematic bands in Table 1.1. (Images courtesy of NASA.)

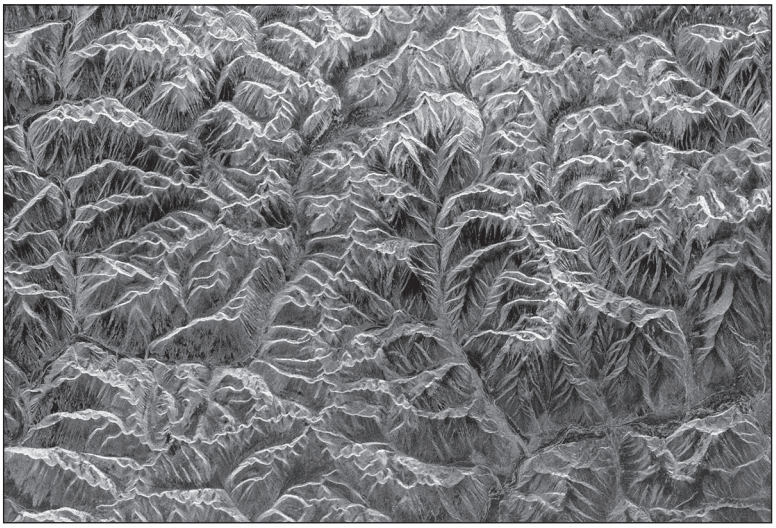
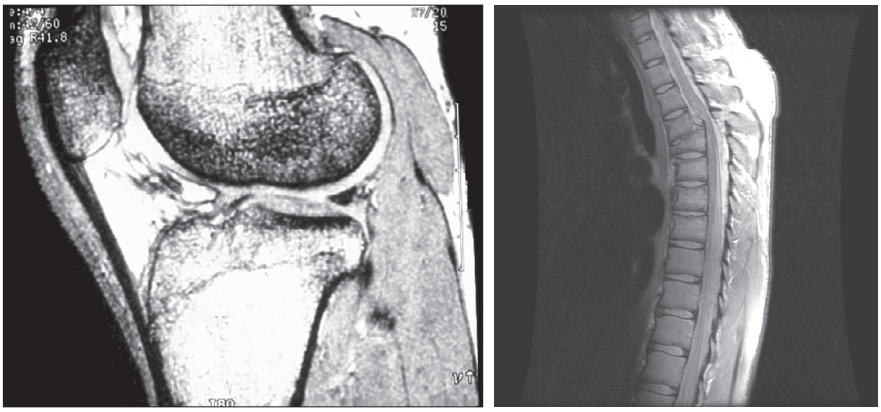


FIGURE 1.16 Spaceborne radar image of mountainous region in southeast Tibet. (Courtesy of NASA.)

in this section acoustic imaging, electron microscopy, and synthetic (computer-generated) imaging.

Imaging using "sound" finds application in geological exploration, industry, and medicine. Geological applications use sound in the low end of the sound spectrum (hundreds of Hz) while imaging in other areas use ultrasound (millions of Hz). The most important commercial applications of image processing in geology are in mineral and oil exploration. For image acquisition over land, one of the main approaches is to use a large truck and a large flat steel plate. The plate is pressed on the ground by



a b

FIGURE 1.17 MRI images of a human (a) knee, and (b) spine. (Figure (a) courtesy of Dr. Thomas R. Gest, Division of Anatomical Sciences, University of Michigan Medical School, and (b) courtesy of Dr. David R. Pickens, Department of Radiology and Radiological Sciences, Vanderbilt University Medical Center.)

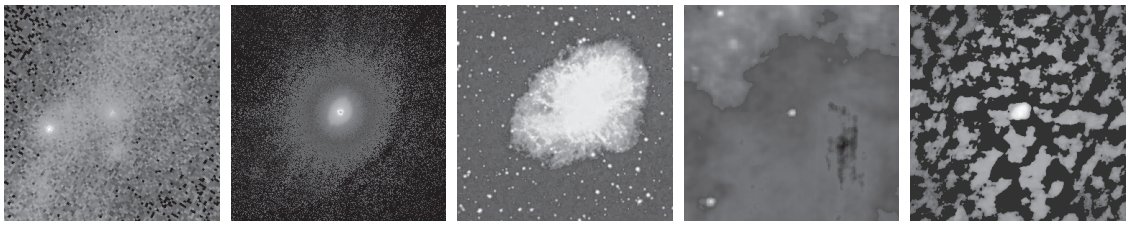


FIGURE 1.18 Images of the Crab Pulsar (in the center of each image) covering the electromagnetic spectrum. (Courtesy of NASA.)

the truck, and the truck is vibrated through a frequency spectrum up to 100 Hz. The strength and speed of the returning sound waves are determined by the composition of the Earth below the surface. These are analyzed by computer, and images are generated from the resulting analysis.

For marine image acquisition, the energy source consists usually of two air guns towed behind a ship. Returning sound waves are detected by hydrophones placed in cables that are either towed behind the ship, laid on the bottom of the ocean, or hung from buoys (vertical cables). The two air guns are alternately pressurized to ~2000 psi and then set off. The constant motion of the ship provides a transversal direction of motion that, together with the returning sound waves, is used to generate a 3-D map of the composition of the Earth below the bottom of the ocean.

Figure 1.19 shows a cross-sectional image of a well-known 3-D model against which the performance of seismic imaging algorithms is tested. The arrow points to a hydrocarbon (oil and/or gas) trap. This target is brighter than the surrounding layers because the change in density in the target region is larger. Seismic interpreters look for these "bright spots" to find oil and gas. The layers above also are bright, but their brightness does not vary as strongly across the layers. Many seismic reconstruction algorithms have difficulty imaging this target because of the faults above it.

Although ultrasound imaging is used routinely in manufacturing, the best known applications of this technique are in medicine, especially in obstetrics, where fetuses are imaged to determine the health of their development. A byproduct of this

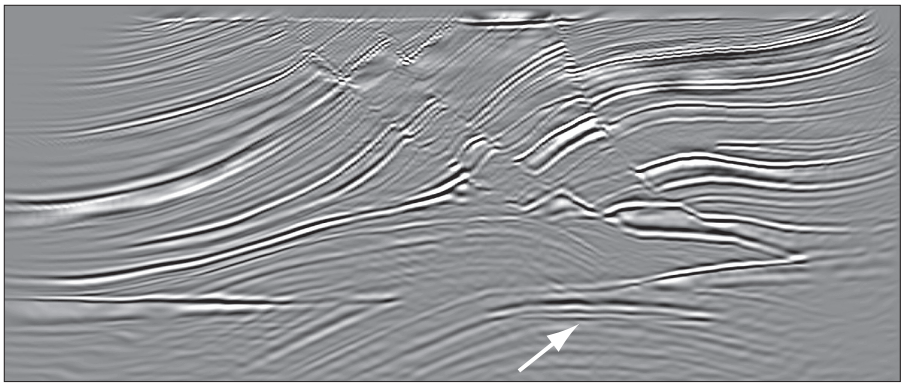


FIGURE 1.19 Cross-sectional image of a seismic model. The arrow points to a hydrocarbon (oil and/or gas) trap. (Courtesy of Dr. Curtis Ober, Sandia National Laboratories.)

1.4 FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING

It is helpful to divide the material covered in the following chapters into the two broad categories defined in Section 1.1: methods whose input and output are images, and methods whose inputs may be images, but whose outputs are attributes extracted from those images. This organization is summarized in Fig. 1.23. The diagram does not imply that every process is applied to an image. Rather, the intention is to convey an idea of all the methodologies that can be applied to images for different purposes, and possibly with different objectives. The discussion in this section may be viewed as a brief overview of the material in the remainder of the book.

Image acquisition is the first process in Fig. 1.23. The discussion in Section 1.3 gave some hints regarding the origin of digital images. This topic will be considered in much more detail in Chapter 2, where we also introduce a number of basic digital image concepts that are used throughout the book. Acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling.

Image enhancement is the process of manipulating an image so the result is more suitable than the original for a specific application. The word *specific* is important here, because it establishes at the outset that enhancement techniques are problem oriented. Thus, for example, a method that is quite useful for enhancing X-ray images may not be the best approach for enhancing satellite images taken in the infrared band of the electromagnetic spectrum.

There is no general "theory" of image enhancement. When an image is processed for visual interpretation, the viewer is the ultimate judge of how well a particular

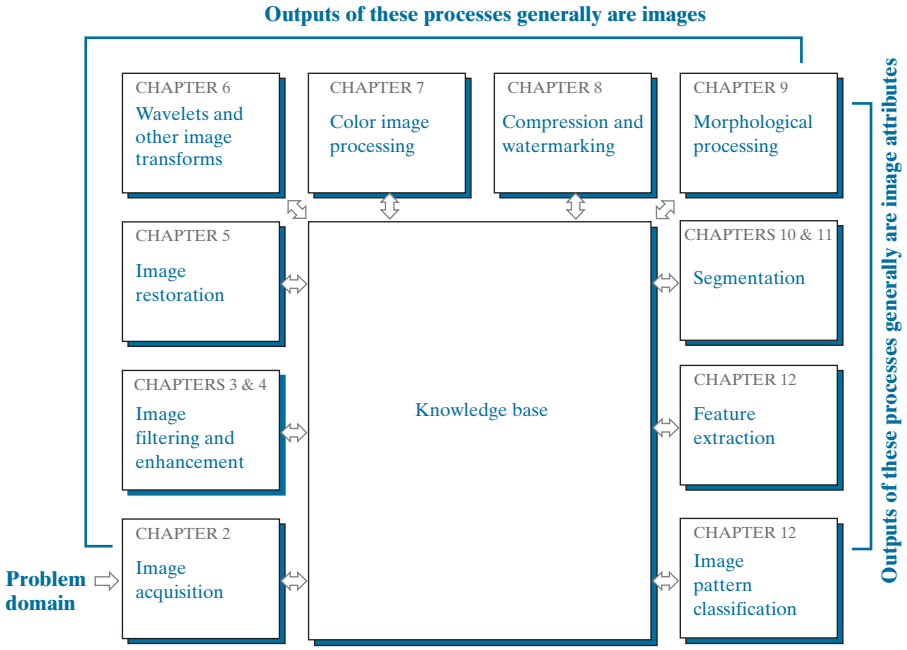


FIGURE 1.23 Fundamental steps in digital image processing. The chapter(s) indicated in the boxes is where the material described in the box is discussed.

method works. Enhancement techniques are so varied, and use so many different image processing approaches, that it is difficult to assemble a meaningful body of techniques suitable for enhancement in one chapter without extensive background development. For this reason, and also because beginners in the field of image processing generally find enhancement applications visually appealing, interesting, and relatively simple to understand, we will use image enhancement as examples when introducing new concepts in parts of Chapter 2 and in Chapters 3 and 4. The material in the latter two chapters span many of the methods used traditionally for image enhancement. Therefore, using examples from image enhancement to introduce new image processing methods developed in these early chapters not only saves having an extra chapter in the book dealing with image enhancement but, more importantly, is an effective approach for introducing newcomers to the details of processing techniques early in the book. However, as you will see in progressing through the rest of the book, the material developed in Chapters 3 and 4 is applicable to a much broader class of problems than just image enhancement.

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation. Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a "good" enhancement result.

Wavelets are the foundation for representing images in various degrees of resolution. In particular, this material is used in the book for image data compression and for pyramidal representation, in which images are subdivided successively into smaller regions. The material in Chapters 4 and 5 is based mostly on the Fourier transform. In addition to wavelets, we will also discuss in Chapter 6 a number of other transforms that are used routinely in image processing.

Color image processing is an area that has been gaining in importance because of the significant increase in the use of digital images over the internet. Chapter 7 covers a number of fundamental concepts in color models and basic color processing in a digital domain. Color is used also as the basis for extracting features of interest in an image.

Compression, as the name implies, deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it. Although storage technology has improved significantly over the past decade, the same cannot be said for transmission capacity. This is true particularly in uses of the internet, which are characterized by significant pictorial content. Image compression is familiar (perhaps inadvertently) to most users of computers in the form of image file extensions, such as the jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape. The material in this chapter begins a transition from processes that output images to processes that output image attributes, as indicated in Section 1.1.

Segmentation partitions an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image

processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually. On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely automated object classification is to succeed.

Feature extraction almost always follows the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself. Feature extraction consists of feature detection and feature description. *Feature detection* refers to finding the features in an image, region, or boundary. *Feature description* assigns quantitative attributes to the detected features. For example, we might detect corners in a region, and describe those corners by their orientation and location; both of these descriptors are quantitative attributes. Feature processing methods discussed in this chapter are subdivided into three principal categories, depending on whether they are applicable to boundaries, regions, or whole images. Some features are applicable to more than one category. Feature descriptors should be as insensitive as possible to variations in parameters such as scale, translation, rotation, illumination, and viewpoint.

Image pattern classification is the process that assigns a label (e.g., "vehicle") to an object based on its feature descriptors. In the last chapter of the book, we will discuss methods of image pattern classification ranging from "classical" approaches such as *minimum-distance*, *correlation*, and *Bayes classifiers*, to more modern approaches implemented using *deep neural networks*. In particular, we will discuss in detail *deep convolutional neural networks*, which are ideally suited for image processing work.

So far, we have said nothing about the need for prior knowledge or about the interaction between the knowledge base and the processing modules in Fig. 1.23. *Knowledge* about a problem domain is coded into an image processing system in the form of a knowledge database. This knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base can also be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem, or an image database containing high-resolution satellite images of a region in connection with change-detection applications. In addition to guiding the operation of each processing module, the knowledge base also controls the interaction between modules. This distinction is made in Fig. 1.23 by the use of double-headed arrows between the processing modules and the knowledge base, as opposed to single-headed arrows linking the processing modules.

Although we do not discuss image display explicitly at this point, it is important to keep in mind that viewing the results of image processing can take place at the output of any stage in Fig. 1.23. We also note that not all image processing applications require the complexity of interactions implied by Fig. 1.23. In fact, not even all those modules are needed in many cases. For example, image enhancement for human visual interpretation seldom requires use of any of the other stages in Fig. 1.23. In general, however, as the complexity of an image processing task increases, so does the number of processes required to solve the problem.

1.5 COMPONENTS OF AN IMAGE PROCESSING SYSTEM

As recently as the mid-1980s, numerous models of image processing systems being sold throughout the world were rather substantial peripheral devices that attached to equally substantial host computers. Late in the 1980s and early in the 1990s, the market shifted to image processing hardware in the form of single boards designed to be compatible with industry standard buses and to fit into engineering workstation cabinets and personal computers. In the late 1990s and early 2000s, a new class of add-on boards, called graphics processing units (GPUs) were introduced for work on 3-D applications, such as games and other 3-D graphics applications. It was not long before GPUs found their way into image processing applications involving large-scale matrix implementations, such as training deep convolutional networks. In addition to lowering costs, the market shift from substantial peripheral devices to add-on processing boards also served as a catalyst for a significant number of new companies specializing in the development of software written specifically for image processing.

The trend continues toward miniaturizing and blending of general-purpose small computers with specialized image processing hardware and software. Figure 1.24 shows the basic components comprising a typical general-purpose system used for digital image processing. The function of each component will be discussed in the following paragraphs, starting with image sensing.

Two subsystems are required to acquire digital images. The first is a physical *sensor* that responds to the energy radiated by the object we wish to image. The second, called a *digitizer*, is a device for converting the output of the physical sensing device into digital form. For instance, in a digital video camera, the sensors (CCD chips) produce an electrical output proportional to light intensity. The digitizer converts these outputs to digital data. These topics will be covered in Chapter 2.

Specialized image processing hardware usually consists of the digitizer just mentioned, plus hardware that performs other primitive operations, such as an *arithmetic logic unit* (ALU), that performs arithmetic and logical operations in parallel on entire images. One example of how an ALU is used is in averaging images as quickly as they are digitized, for the purpose of noise reduction. This type of hardware sometimes is called a *front-end subsystem*, and its most distinguishing characteristic is speed. In other words, this unit performs functions that require fast data throughputs (e.g., digitizing and averaging video images at 30 frames/s) that the typical main computer cannot handle. One or more GPUs (see above) also are common in image processing systems that perform intensive matrix operations.

The *computer* in an image processing system is a general-purpose computer and can range from a PC to a supercomputer. In dedicated applications, sometimes custom computers are used to achieve a required level of performance, but our interest here is on general-purpose image processing systems. In these systems, almost any well-equipped PC-type machine is suitable for off-line image processing tasks.

Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules. More sophisticated

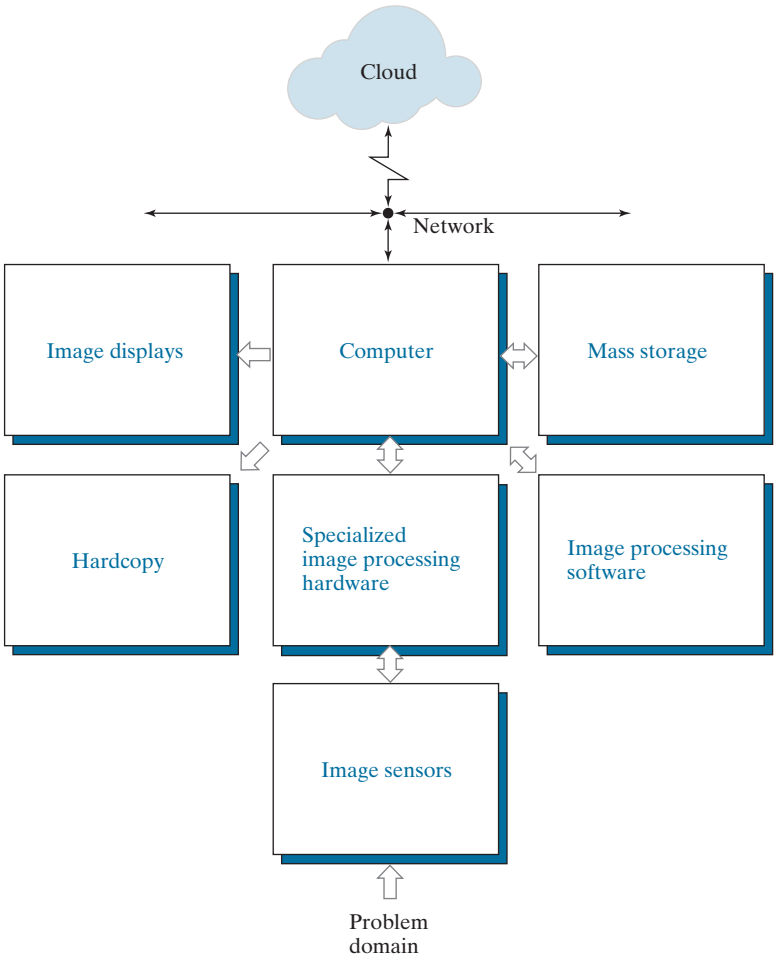


FIGURE 1.24 Components of a general-purpose image processing system.

software packages allow the integration of those modules and general-purpose software commands from at least one computer language. Commercially available image processing software, such as the well-known MATLAB® Image Processing Toolbox, is also common in a well-equipped image processing system.

Mass storage is a must in image processing applications. An image of size 1024×1024 pixels, in which the intensity of each pixel is an 8-bit quantity, requires one megabyte of storage space if the image is not compressed. When dealing with image databases that contain thousands, or even millions, of images, providing adequate storage in an image processing system can be a challenge. Digital storage for image processing applications falls into three principal categories: (1) short-term storage for use during processing; (2) on-line storage for relatively fast recall; and (3) archival storage, characterized by infrequent access. Storage is measured in bytes (eight bits), Kbytes (10^3 bytes), Mbytes (10^6 bytes), Gbytes (10^9 bytes), and Tbytes (10^{12} bytes).

One method of providing short-term storage is computer memory. Another is by specialized boards, called *frame buffers*, that store one or more images and can be accessed rapidly, usually at video rates (e.g., at 30 complete images per second). The latter method allows virtually instantaneous image *zoom*, as well as *scroll* (vertical shifts) and *pan* (horizontal shifts). Frame buffers usually are housed in the specialized image processing hardware unit in Fig. 1.24. On-line storage generally takes the form of magnetic disks or optical-media storage. The key factor characterizing on-line storage is frequent access to the stored data. Finally, archival storage is characterized by massive storage requirements but infrequent need for access. Magnetic tapes and optical disks housed in "jukeboxes" are the usual media for archival applications.

Image displays in use today are mainly color, flat screen monitors. Monitors are driven by the outputs of image and graphics display cards that are an integral part of the computer system. Seldom are there requirements for image display applications that cannot be met by display cards and GPUs available commercially as part of the computer system. In some cases, it is necessary to have stereo displays, and these are implemented in the form of headgear containing two small displays embedded in goggles worn by the user.

Hardcopy devices for recording images include laser printers, film cameras, heat-sensitive devices, ink-jet units, and digital units, such as optical and CD-ROM disks. Film provides the highest possible resolution, but paper is the obvious medium of choice for written material. For presentations, images are displayed on film transparencies or in a digital medium if image projection equipment is used. The latter approach is gaining acceptance as the standard for image presentations.

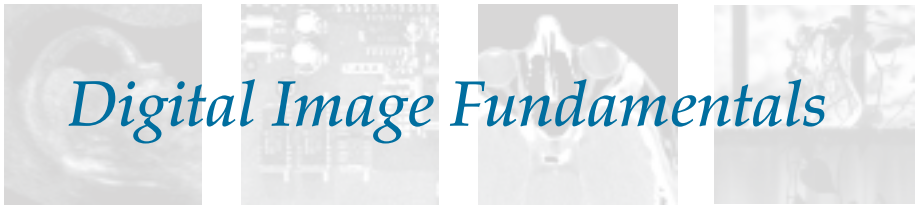
Networking and *cloud* communication are almost default functions in any computer system in use today. Because of the large amount of data inherent in image processing applications, the key consideration in image transmission is *bandwidth*. In dedicated networks, this typically is not a problem, but communications with remote sites via the internet are not always as efficient. Fortunately, transmission bandwidth is improving quickly as a result of optical fiber and other broadband technologies. Image data compression continues to play a major role in the transmission of large amounts of image data.

Summary, References, and Further Reading

The main purpose of the material presented in this chapter is to provide a sense of perspective about the origins of digital image processing and, more important, about current and future areas of application of this technology. Although the coverage of these topics in this chapter was necessarily incomplete due to space limitations, it should have left you with a clear impression of the breadth and practical scope of digital image processing. As we proceed in the following chapters with the development of image processing theory and applications, numerous examples are provided to keep a clear focus on the utility and promise of these techniques. Upon concluding the study of the final chapter, a reader of this book will have arrived at a level of understanding that is the foundation for most of the work currently underway in this field.

In past editions, we have provided a long list of journals and books to give readers an idea of the breadth of the image processing literature, and where this literature is reported. The list has been updated, and it has become so extensive that it is more practical to include it in the book website: *www.ImageProcessingPlace.com*, in the section entitled *Publications*.

2 Digital Image Fundamentals



> Those who wish to succeed must ask the right preliminary questions.
>
> *Aristotle*

Preview

This chapter is an introduction to a number of basic concepts in digital image processing that are used throughout the book. Section 2.1 summarizes some important aspects of the human visual system, including image formation in the eye and its capabilities for brightness adaptation and discrimination. Section 2.2 discusses light, other components of the electromagnetic spectrum, and their imaging characteristics. Section 2.3 discusses imaging sensors and how they are used to generate digital images. Section 2.4 introduces the concepts of uniform image sampling and intensity quantization. Additional topics discussed in that section include digital image representation, the effects of varying the number of samples and intensity levels in an image, the concepts of spatial and intensity resolution, and the principles of image interpolation. Section 2.5 deals with a variety of basic relationships between pixels. Finally, Section 2.6 is an introduction to the principal mathematical tools we use throughout the book. A second objective of that section is to help you begin developing a "feel" for how these tools are used in a variety of basic image processing tasks.

Upon completion of this chapter, readers should:

- Have an understanding of some important functions and limitations of human vision.
- Be familiar with the electromagnetic energy spectrum, including basic properties of light.
- Know how digital images are generated and represented.
- Understand the basics of image sampling and quantization.
- Be familiar with spatial and intensity resolution and their effects on image appearance.
- Have an understanding of basic geometric relationships between image pixels.
- Be familiar with the principal mathematical tools used in digital image processing.
- Be able to apply a variety of introductory digital image processing techniques.

2.1 ELEMENTS OF VISUAL PERCEPTION

Although the field of digital image processing is built on a foundation of mathematics, human intuition and analysis often play a role in the choice of one technique versus another, and this choice often is made based on subjective, visual judgments. Thus, developing an understanding of basic characteristics of human visual perception as a first step in our journey through this book is appropriate. In particular, our interest is in the elementary mechanics of how images are formed and perceived by humans. We are interested in learning the physical limitations of human vision in terms of factors that also are used in our work with digital images. Factors such as how human and electronic imaging devices compare in terms of resolution and ability to adapt to changes in illumination are not only interesting, they are also important from a practical point of view.

STRUCTURE OF THE HUMAN EYE

Figure 2.1 shows a simplified cross section of the human eye. The eye is nearly a sphere (with a diameter of about 20 mm) enclosed by three membranes: the *cornea* and *sclera* outer cover; the *choroid*; and the *retina*. The cornea is a tough, transparent tissue that covers the anterior surface of the eye. Continuous with the cornea, the sclera is an opaque membrane that encloses the remainder of the optic globe.

The choroid lies directly below the sclera. This membrane contains a network of blood vessels that serve as the major source of nutrition to the eye. Even superficial

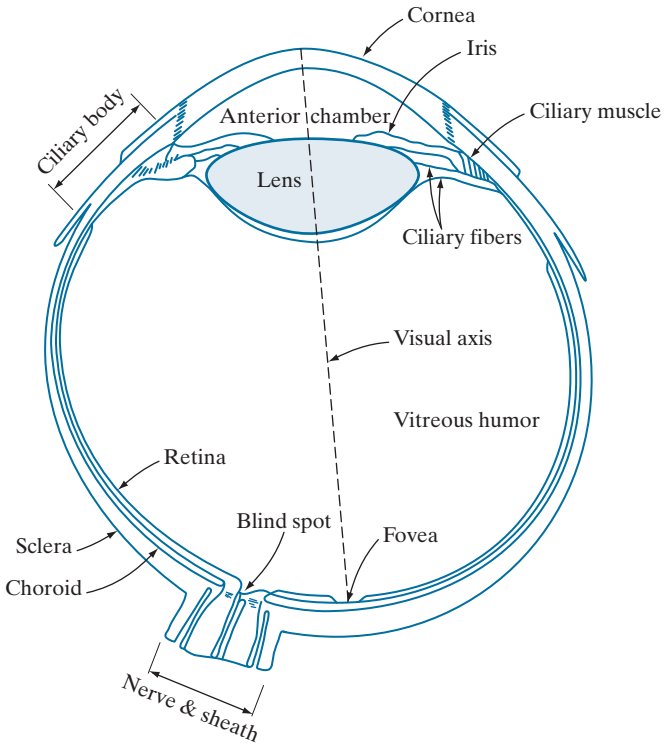


FIGURE 2.1 Simplified diagram of a cross section of the human eye.

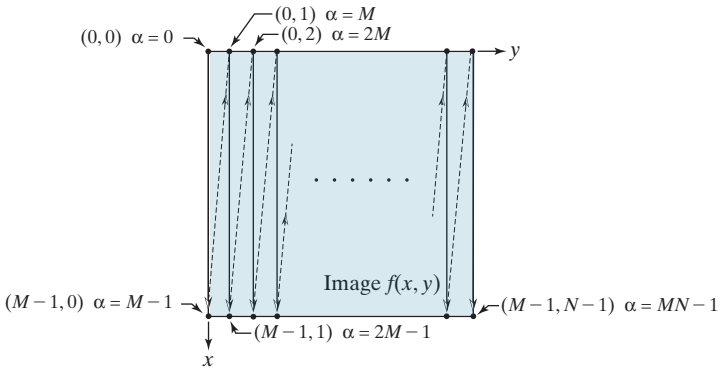


FIGURE 2.22 Illustration of column scanning for generating linear indices. Shown are several 2-D coordinates (in parentheses) and their corresponding linear indices.

Conversely, the coordinate indices for a given linear index value α are given by the equations†

$$x = \alpha \bmod M \tag{2-15}$$

and

$$y = (\alpha - x)/M \tag{2-16}$$

Recall that $\alpha \bmod M$ means "the remainder of the division of α by M." This is a formal way of stating that row numbers repeat themselves at the start of every column. Thus, when $\alpha = 0$, the remainder of the division of 0 by M is 0, so $x = 0$. When $\alpha = 1$, the remainder is 1, and so $x = 1$. You can see that x will continue to be equal to α until $\alpha = M - 1$. When $\alpha = M$ (which is at the beginning of the second column), the remainder is 0, and thus $x = 0$ again, and it increases by 1 until the next column is reached, when the pattern repeats itself. Similar comments apply to Eq. (2-16). See Problem 2.13 for a derivation of the preceding two equations.

SPATIAL AND INTENSITY RESOLUTION

Intuitively, *spatial resolution* is a measure of the smallest discernible detail in an image. Quantitatively, spatial resolution can be stated in several ways, with *line pairs per unit distance*, and *dots (pixels) per unit distance* being common measures. Suppose that we construct a chart with alternating black and white vertical lines, each of width W units (W can be less than 1). The width of a *line pair* is thus $2W$, and there are $W/2$ line pairs per unit distance. For example, if the width of a line is 0.1 mm, there are 5 line pairs per unit distance (i.e., per mm). A widely used definition of image resolution is the largest number of *discernible* line pairs per unit distance (e.g., 100 line pairs per mm). Dots per unit distance is a measure of image resolution used in the printing and publishing industry. In the U.S., this measure usually is expressed as *dots per inch* (dpi). To give you an idea of quality, newspapers are printed with a

†When working with modular number systems, it is more accurate to write $x \equiv \alpha \bmod M$, where the symbol $\equiv$ means *congruence*. However, our interest here is just on converting from linear to coordinate indexing, so we use the more familiar equal sign.

resolution of 75 dpi, magazines at 133 dpi, glossy brochures at 175 dpi, and the book page at which you are presently looking was printed at 2400 dpi.

To be meaningful, measures of spatial resolution must be stated with respect to spatial units. Image size by itself does not tell the complete story. For example, to say that an image has a resolution of 1024×1024 pixels is not a meaningful statement without stating the spatial dimensions encompassed by the image. Size by itself is helpful only in making comparisons between imaging capabilities. For instance, a digital camera with a 20-megapixel CCD imaging chip can be expected to have a higher capability to resolve detail than an 8-megapixel camera, assuming that both cameras are equipped with comparable lenses and the comparison images are taken at the same distance.

Intensity resolution similarly refers to the smallest *discernible* change in intensity level. We have considerable discretion regarding the number of spatial samples (pixels) used to generate a digital image, but this is not true regarding the number of intensity levels. Based on hardware considerations, the number of intensity levels usually is an integer power of two, as we mentioned when discussing Eq. (2-11). The most common number is 8 bits, with 16 bits being used in some applications in which enhancement of specific intensity ranges is necessary. Intensity quantization using 32 bits is rare. Sometimes one finds systems that can digitize the intensity levels of an image using 10 or 12 bits, but these are not as common.

Unlike spatial resolution, which must be based on a per-unit-of-distance basis to be meaningful, it is common practice to refer to the number of bits used to quantize intensity as the "*intensity resolution*." For example, it is common to say that an image whose intensity is quantized into 256 levels has 8 bits of intensity resolution. However, keep in mind that *discernible* changes in intensity are influenced also by noise and saturation values, and by the capabilities of human perception to analyze and interpret details in the context of an entire scene (see Section 2.1). The following two examples illustrate the effects of spatial and intensity resolution on discernible detail. Later in this section, we will discuss how these two parameters interact in determining perceived image quality.

EXAMPLE 2.2: Effects of reducing the spatial resolution of a digital image.

Figure 2.23 shows the effects of reducing the spatial resolution of an image. The images in Figs. 2.23(a) through (d) have resolutions of 930, 300, 150, and 72 dpi, respectively. Naturally, the lower resolution images are smaller than the original image in (a). For example, the original image is of size 2136×2140 pixels, but the 72 dpi image is an array of only 165×166 pixels. In order to facilitate comparisons, all the smaller images were zoomed back to the original size (the method used for zooming will be discussed later in this section). This is somewhat equivalent to "getting closer" to the smaller images so that we can make comparable statements about visible details.

There are some small visual differences between Figs. 2.23(a) and (b), the most notable being a slight distortion in the seconds marker pointing to 60 on the right side of the chronometer. For the most part, however, Fig. 2.23(b) is quite acceptable. In fact, 300 dpi is the typical minimum image spatial resolution used for book publishing, so one would not expect to see much difference between these two images. Figure 2.23(c) begins to show visible degradation (see, for example, the outer edges of the chronometer

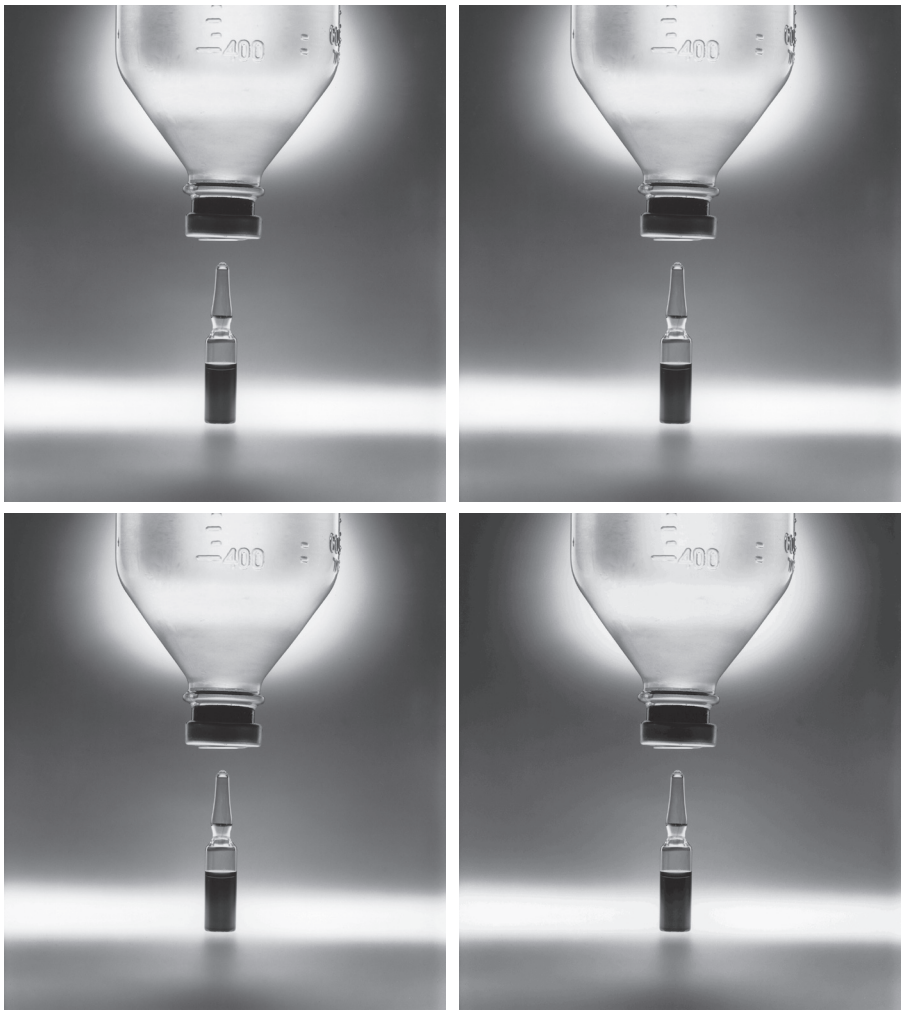
a b
c d

FIGURE 2.23
Effects of reducing spatial resolution. The images shown are at:
(a) 930 dpi,
(b) 300 dpi,
(c) 150 dpi, and
(d) 72 dpi.

case and compare the seconds marker with the previous two images). The numbers also show visible degradation. Figure 2.23(d) shows degradation that is visible in most features of the image. As we will discuss in Section 4.5, when printing at such low resolutions, the printing and publishing industry uses a number of techniques (such as locally varying the pixel size) to produce much better results than those in Fig. 2.23(d). Also, as we will show later in this section, it is possible to improve on the results of Fig. 2.23 by the choice of interpolation method used.

EXAMPLE 2.3: Effects of varying the number of intensity levels in a digital image.

Figure 2.24(a) is a 256-level grayscale image of a chemotherapy vial (bottom) and a drip bottle. The objective of this example is to reduce the number of intensities of this image from 256 to 2 in integer powers of 2, while leaving the image resolution at a fixed 783 dpi (the images are of size 2022×1800 pixels). Figures 2.24(b) through (d) were obtained by reducing the number of intensity levels to 128, 64, and 32, respectively (we will discuss how to reduce the number of levels in Chapter 3). The 128- and

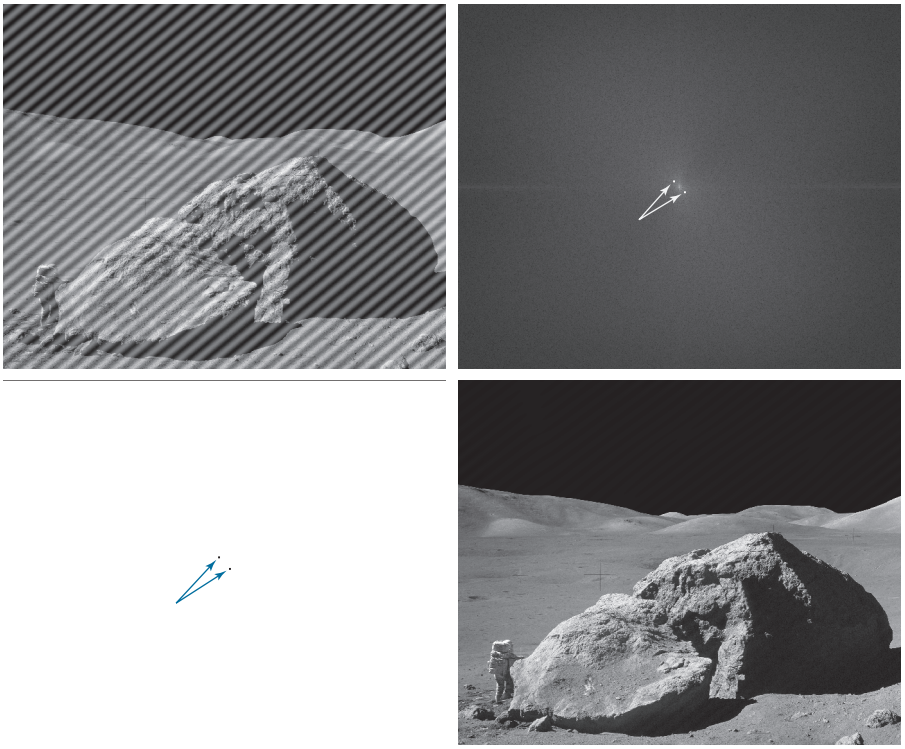


a b
c d

FIGURE 2.24
(a) 2022×1800, 256-level image. (b)-(d) Image displayed in 128, 64, and 32 intensity levels, while keeping the image size constant. (Original image courtesy of the National Cancer Institute.)

64-level images are visually identical for all practical purposes. However, the 32-level image in Fig. 2.24(d) has a set of almost imperceptible, very fine ridge-like structures in areas of constant intensity. These structures are clearly visible in the 16-level image in Fig. 2.24(e). This effect, caused by using an insufficient number of intensity levels in smooth areas of a digital image, is called *false contouring*, so named because the ridges resemble topographic contours in a map. False contouring generally is quite objectionable in images displayed using 16 or fewer uniformly spaced intensity levels, as the images in Figs. 2.24(e)-(h) show.

As a very rough guideline, and assuming integer powers of 2 for convenience, images of size 256×256 pixels with 64 intensity levels, and printed on a size format on the order of 5×5 cm, are about the lowest spatial and intensity resolution images that can be expected to be reasonably free of objectionable sampling distortions and false contouring.



a b
c d

FIGURE 2.45
(a) Image corrupted by sinusoidal interference. (b) Magnitude of the Fourier transform showing the bursts of energy caused by the interference (the bursts were enlarged for display purposes). (c) Mask used to eliminate the energy bursts. (d) Result of computing the inverse of the modified Fourier transform. (Original image courtesy of NASA.)

manifests itself as bright bursts of intensity, whose location is determined by the frequency of the sinusoidal interference (we will discuss these concepts in much more detail in Chapters 4 and 5). Typically, the bursts are easily observable in an image of the magnitude of the Fourier transform, $|T(u,v)|$. With reference to the diagram in Fig. 2.44, the corrupted image is $f(x,y)$, the transform in the leftmost box is the Fourier transform, and Fig. 2.45(b) is $|T(u,v)|$ displayed as an image. The bright dots shown are the bursts of intensity mentioned above. Figure 2.45(c) shows a mask image (called a *filter*) with white and black representing 1 and 0, respectively. For this example, the operation in the second box of Fig. 2.44 is to multiply the filter by the transform to remove the bursts associated with the interference. Figure 2.45(d) shows the final result, obtained by computing the inverse of the modified transform. The interference is no longer visible, and previously unseen image detail is now made quite clear. Observe, for example, the fiducial marks (faint crosses) that are used for image registration, as discussed earlier.

When the forward and inverse kernels of a transform are separable and symmetric, and $f(x,y)$ is a square image of size $M \times M$, Eqs. (2-55) and (2-56) can be expressed in matrix form:

$$\mathbf{T} = \mathbf{AFA} \tag{2-63}$$

where $\mathbf{F}$ is an $M \times M$ matrix containing the elements of $f(x,y)$ [see Eq. (2-9)], $\mathbf{A}$ is an $M \times M$ matrix with elements $a_{ij} = r_1(i,j)$, and $\mathbf{T}$ is an $M \times M$ transform matrix with elements $T(u,v)$, for $u,v = 0, 1, 2, \ldots, M-1$.

To obtain the inverse transform, we pre- and post-multiply Eq. (2-63) by an inverse transformation matrix **B**:

$$\mathbf{BTB} = \mathbf{BAFAB} \tag{2-64}$$

If $\mathbf{B} = \mathbf{A}^{-1}$,

$$\mathbf{F} = \mathbf{BTB} \tag{2-65}$$

indicating that **F** or, equivalently, $f(x, y)$, can be recovered completely from its forward transform. If **B** is not equal to $\mathbf{A}^{-1}$, Eq. (2-65) yields an approximation:

$$\hat{\mathbf{F}} = \mathbf{BAFAB} \tag{2-66}$$

In addition to the Fourier transform, a number of important transforms, including the *Walsh*, *Hadamard*, *discrete cosine*, *Haar*, and *slant* transforms, can be expressed in the form of Eqs. (2-55) and (2-56), or, equivalently, in the form of Eqs. (2-63) and (2-65). We will discuss these and other types of image transforms in later chapters.

PROBABILITY AND RANDOM VARIABLES

Probability is a branch of mathematics that deals with uncertainty. The following material is a brief introduction to probability and random variables. Many of these concepts are developed further as needed later in the book.

Sample Spaces, Events, and Probability

A *random experiment* is a process whose outcome cannot be predicted with certainty, but whose set of *all possible* outcomes can be specified. As noted earlier when discussing sets, the set of all possible outcomes of an experiment is called the *sample space* of the experiment, and is denoted by Ω. A familiar random experiment consists of tossing a single die and observing the numerical value of the face that lands facing up. The sample space of this experiment is the set $\Omega = \{1, 2, 3, 4, 5, 6\}$.

An *event* is a *subset* of the sample space. In the single-die experiment, the event $A = \{1, 3, 5\}$ is the subset of Ω that correspond to the odd faces of the die. We say that an event *occurs* if the outcome of an experiment is *any* of the elements of the event set.

To make the notion of a random experiment useful, we need a measure (a probability) that quantifies how likely it is than an event will occur. A *probability*, P, is a *function* that satisfies the following properties:

1. $0 \le P(A) \le 1$ for every event A.

2. $P(\Omega) = 1$.

3. If $A_1, A_2, \ldots, A_n$ are disjoint events, then

$$P(A_1 \cup A_2 \cup \cdots \cup A_n) = P(A_1) + P(A_2) + \cdots + P(A_n).$$

(2-67)

Recall from our earlier discussion on sets that two or more sets are disjoint if they have no elements in common.

These three properties are called the *axioms of probability*. Axiom 1 says that the probability must be a number in the range [0, 1], with 0 indicating that A never

occurs, and 1 indicating that A always occurs. Because Ω is the set containing all possible outcomes, the second axiom indicates that some event from Ω always occurs when the experiment is performed. Axiom 3 states that the probability of the union of a sequence of *disjoint* events is equal to the sum of the probabilities of the individual events.

EXAMPLE 2.12: Events and probabilities.

Consider the experiment of tossing a pair of dice, one after the other, and observing the faces that turn up, in the order in which the dice were tossed. The elements of Ω are of the form (i, j), where i and j are the values of the up face in the first and second toss, respectively. Thus, set Ω has 36 elements:

$$\Omega = \{(1,1),(1,2),\ldots,(1,6),(2,1),(2,2),\ldots,(2,6),\ldots,(6,1),(6,2),\ldots,(6,6)\}$$

If the dice are unbiased, each combination is equally likely to turn up. The probability of each is 1/36 because there are 36 equally likely outcomes to this experiment.

Let A denote the event that the first die turns up 1, and B the event that the second die turns up 2:

$$A = \{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6)\}$$

and

$$B = \{(1,2),(2,2),(3,2),(4,2),(5,2),(6,2)\}$$

Then, $P(A) = P(B) = 6/36 = 1/6$ because 6 of the 36 possible outcomes are favorable to A and the probability of each is 1/36; and similarly for B.

Events are sets, so we can use the results of our earlier discussion on sets to form more complex events involving A and B, and answer questions about their probability of occurrence. For example,

$$\begin{aligned} A \cap B &= \{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6)\} \cap \{(1,2),(2,2),(3,2),(4,2),(5,2),(6,2)\} \\ &= \{(1,2)\} \end{aligned}$$

is the event that the first die comes up 1 *and* the second comes up 2. This event has one element so its probability, $P(A \cap B)$, is 1/36. Similarly, the event that a 1 comes up in the first die *or* a 2 comes up in the second is given by $A \cup B$. This event set has 12 elements, so its probability, $P(A \cup B)$, is $12/36 = 1/3$.

The Sum (Addition) Rule of Probability

Axiom 3 is a special case of the *sum* (or *addition*) *rule* of probability, which states that the probability of the union of n events is equal to the sum of the probabilities of these events taken one at time, minus the sum of the probabilities of the events taken two at a time, plus the sum of the probabilities of the events taken three at a time, and so on, up to the sum of the probabilities of all the n events (Ross [2014]).

For two events, the sum rule is

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{2-68}$$

For three events, this expression becomes

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C) \tag{2-69}$$

Recall that the intersection, $\cap$, of two or more disjoint sets is null.

When the events are disjoint, all terms except the individual probabilities become zero, thus reducing the expression to the one given in Axiom 3. The rightmost term in Eq. (2-69) is a result of applying Eq. (2-68) to combined events (see Problem 2.44).

A probability of the form $P(A \cap B)$ is called a *joint probability* and is read "the probability of A *and* B." When the events are disjoint, the probability of both happening simultaneously is zero. A probability of the form $P(A \cup B)$ is read "the probability of A *or* B." Because this probability involves the union of A and B, which pools the elements of both, $P(A \cup B)$ is actually the probability of A or B or *both*.

EXAMPLE 2.13: Working with the addition rule of probability.

Two dice are rolled. What is the probability that their sum will be 6 *or* 8? As before, the sample space has 36 elements. The events summing to 6 are $A = \{(1,5),(5,1),(2,4),(4,2),(3,3)\}$ and the events summing to 8 are $B = \{(2,6),(6,2),(3,5),(5,3),(4,4)\}$. Then, $P(A) = P(B) = 5/36$, and $P(A \cap B) = 0$ because both events cannot happen simultaneously in one roll of the dice. Thus, $P(A \cup B) = 5/36 + 5/36 = 5/18$.

What is the probability that the sum is even *or* a multiple of 3? Let A be the first event and B the second. Of the 36 possible outcomes, 18 sums are even, so $P(A) = 18/36$. Twelve sums are multiples of 3, so $P(B) = 12/36$. It is possible for a sum to be even *and* a multiple of 3, so the events are no longer disjoint. If you list all possible events, you will find that A and B have 6 elements in common; therefore, $P(A \cap B) = 6/36$. Putting it all together,

$$\begin{aligned} P(\text{even OR multiple of 3}) &= P(\text{even}) + P(\text{multiple of 3}) - P(\text{both}) \\ &= P(A) + P(B) - P(A \cap B) = 24/36 = 2/3. \end{aligned}$$

When the events are not disjoint, summing the two individual probabilities counts their shared events twice (this is the overlap between the two sets in a Venn diagram, as in Fig. 2.35). Subtracting $P(A \cap B)$ eliminates the double from the total. Another way of arriving at the same conclusion is to recall that the union of two sets "pools" the elements of both. The intersection is the total number of elements common to both sets, thus resulting in duplicates. Because of this overlap when the events are not disjoint, $P(A \cup B)$ is the probability of A or B or *both* occurring.

Conditional Probability

The probability of event A, given that event B has occurred, is defined as

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2-70}$$

where $P(A|B)$ is called the *conditional probability*; it reads "the probability of A given B." As noted above, $P(A \cap B)$ is the joint probability of A and B.

3 Intensity Transformations and Spatial Filtering

It makes all the difference whether one sees darkness through the light or brightness through the shadows.

David Lindsay

Preview

The term *spatial domain* refers to the image plane itself, and image processing methods in this category are based on direct manipulation of pixels in an image. This is in contrast to image processing in a transform domain which, as we will discuss in Chapters 4 and 6, involves first transforming an image into the transform domain, doing the processing there, and obtaining the inverse transform to bring the results back into the spatial domain. Two principal categories of spatial processing are intensity transformations and spatial filtering. Intensity transformations operate on single pixels of an image for tasks such as contrast manipulation and image thresholding. Spatial filtering performs operations on the neighborhood of every pixel in an image. Examples of spatial filtering include image smoothing and sharpening. In the sections that follow, we discuss a number of "classical" techniques for intensity transformations and spatial filtering. We also discuss fuzzy techniques that allow us to incorporate imprecise, knowledge-based information in the formulation of image processing algorithms.

Upon completion of this chapter, readers should:

- Understand the meaning of spatial domain processing, and how it differs from transform domain processing.
- Be familiar with the principal techniques used for intensity transformations.
- Understand the physical meaning of image histograms and how they can be manipulated for image enhancement.
- Understand the mechanics of spatial filtering, and how spatial filters are formed.
- Understand the principles of spatial convolution and correlation.
- Be familiar with the principal types of spatial filters, and how they are applied.
- Be aware of the relationships between spatial filters, and the fundamental role of lowpass filters.
- Understand the principles of fuzzy logic and how these principles apply to digital image processing.

3.1 BACKGROUND

All the image processing techniques discussed in this chapter are implemented in the spatial domain, which we know from the discussion in Section 2.4 is the plane containing the pixels of an image. Spatial domain techniques operate directly on the pixels of an image, as opposed, for example, to the frequency domain (the topic of Chapter 4) in which operations are performed on the Fourier transform of an image, rather than on the image itself. As you will learn in progressing through the book, some image processing tasks are easier or more meaningful to implement in the spatial domain, while others are best suited for other approaches.

THE BASICS OF INTENSITY TRANSFORMATIONS AND SPATIAL FILTERING

The spatial domain processes we discuss in this chapter are based on the expression

$$g(x,y) = T\left[f(x,y)\right] \tag{3-1}$$

where $f(x,y)$ is an input image, $g(x,y)$ is the output image, and T is an operator on f defined over a neighborhood of point (x,y). The operator can be applied to the pixels of a single image (our principal focus in this chapter) or to the pixels of a set of images, such as performing the elementwise sum of a sequence of images for noise reduction, as discussed in Section 2.6. Figure 3.1 shows the basic implementation of Eq. (3-1) on a single image. The point (x_0, y_0) shown is an arbitrary location in the image, and the small region shown is a *neighborhood* of (x_0, y_0), as explained in Section 2.6. Typically, the neighborhood is rectangular, centered on (x_0, y_0), and much smaller in size than the image.

The process that Fig. 3.1 illustrates consists of moving the center of the neighborhood from pixel to pixel, and applying the operator T to the pixels in the neighborhood to yield an output value at that location. Thus, for any specific location (x_0, y_0),

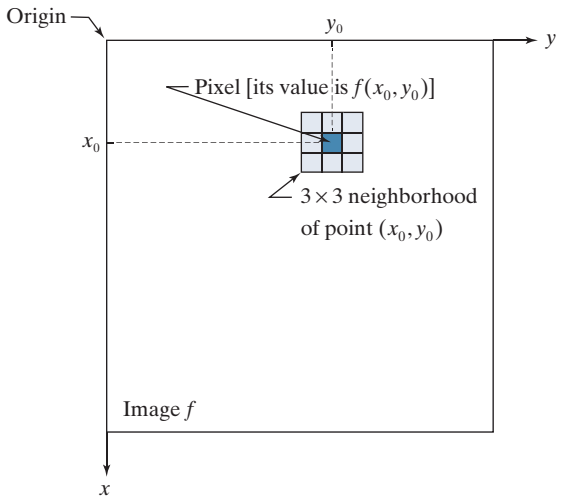


FIGURE 3.1 A 3×3 neighborhood about a point (x_0, y_0) in an image. The neighborhood is moved from pixel to pixel in the image to generate an output image. Recall from Chapter 2 that the value of a pixel at location (x_0, y_0) is $f(x_0, y_0)$, the value of the image at that location.

interval [0,255] and showing the spectrum in the same 8-bit display. The level of detail visible in this image as compared to an unmodified display of the spectrum is evident from these two images. Most of the Fourier spectra in image processing publications, including this book, have been scaled in this manner.

POWER-LAW (GAMMA) TRANSFORMATIONS

Power-law transformations have the form

$$s = cr^{\gamma} \tag{3-5}$$

where c and γ are positive constants. Sometimes Eq. (3-5) is written as $s = c(r+\varepsilon)^{\gamma}$ to account for offsets (that is, a measurable output when the input is zero). However, offsets typically are an issue of display calibration, and as a result they are normally ignored in Eq. (3-5). Figure 3.6 shows plots of s as a function of r for various values of γ. As with log transformations, power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels. Note also in Fig. 3.6 that a family of transformations can be obtained simply by varying γ. Curves generated with values of $\gamma > 1$ have exactly the opposite effect as those generated with values of $\gamma < 1$. When $c = \gamma = 1$ Eq. (3-5) reduces to the identity transformation.

The response of many devices used for image capture, printing, and display obey a power law. By convention, the exponent in a power-law equation is referred to as *gamma* [hence our use of this symbol in Eq. (3-5)]. The process used to correct these power-law response phenomena is called *gamma correction* or *gamma encoding*. For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5. As the curve for $\gamma = 2.5$ in Fig. 3.6 shows, such display systems would tend to produce

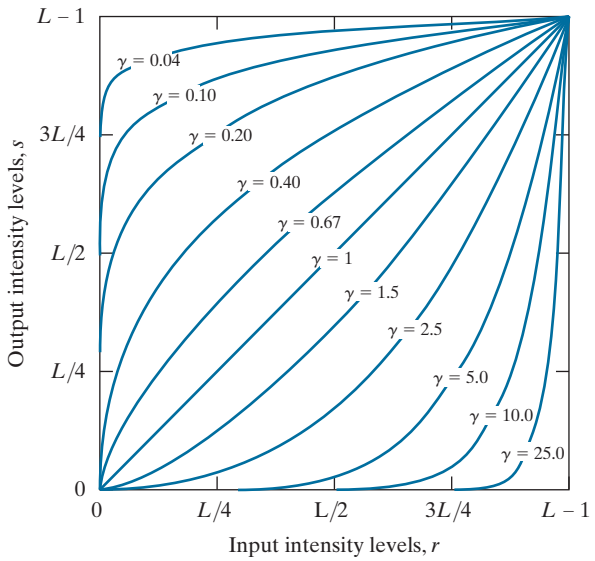


FIGURE 3.6 Plots of the gamma equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases). Each curve was scaled *independently* so that all curves would fit in the same graph. Our interest here is on the *shapes* of the curves, not on their relative values.

a b
c d

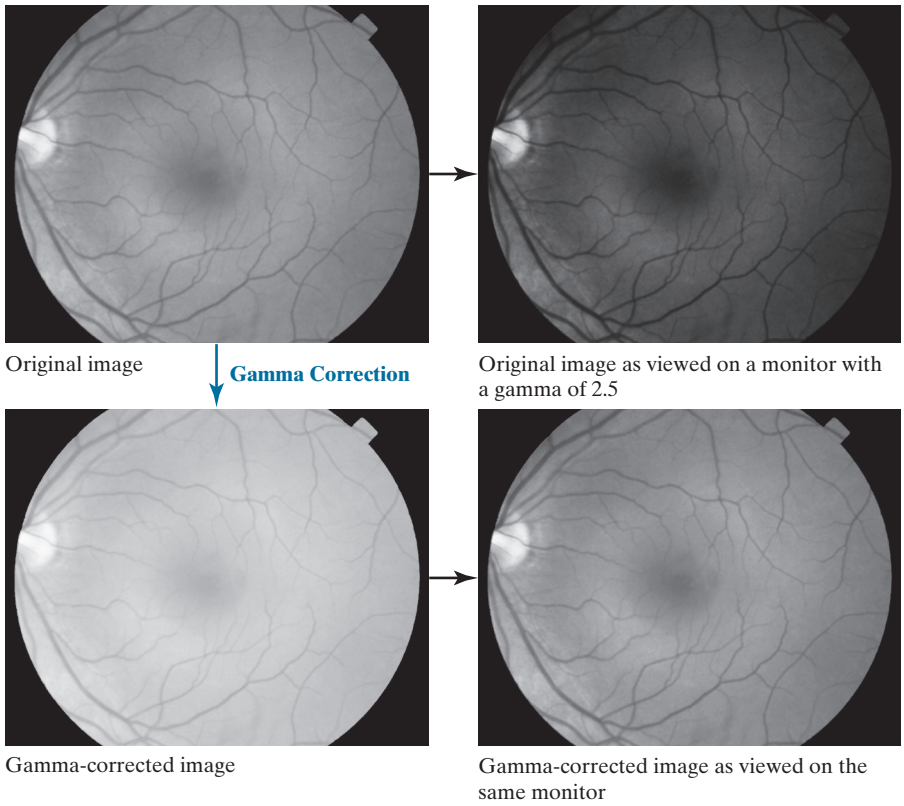


FIGURE 3.7
(a) Image of a human retina. (b) Image as as it appears on a monitor with a gamma setting of 2.5 (note the darkness). (c) Gamma-corrected image. (d) Corrected image, as it appears on the same monitor (compare with the original image). (Image (a) courtesy of the National Eye Institute, NIH)

Sometimes, a higher gamma makes the displayed image look better to viewers than the original because of an increase in contrast. However, the objective of gamma correction is to produce a *faithful* display of an input image.

images that are darker than intended. Figure 3.7 illustrates this effect. Figure 3.7(a) is an image of a human retina displayed in a monitor with a gamma of 2.5. As expected, the output of the monitor appears darker than the input, as Fig. 3.7(b) shows.

In this case, gamma correction consists of using the transformation $s = r^{1/2.5} = r^{0.4}$ to preprocess the image before inputting it into the monitor. Figure 3.7(c) is the result. When input into the same monitor, the gamma-corrected image produces an output that is close in appearance to the original image, as Fig. 3.7(d) shows. A similar analysis as above would apply to other imaging devices, such as scanners and printers, the difference being the device-dependent value of gamma (Poynton [1996]).

EXAMPLE 3.1: Contrast enhancement using power-law intensity transformations.

In addition to gamma correction, power-law transformations are useful for general-purpose contrast manipulation. Figure 3.8(a) shows a magnetic resonance image (MRI) of a human upper thoracic spine with a fracture dislocation. The fracture is visible in the region highlighted by the circle. Because the image is predominantly dark, an expansion of intensity levels is desirable. This can be accomplished using a power-law transformation with a fractional exponent. The other images shown in the figure were obtained by processing Fig. 3.8(a) with the power-law transformation function of Eq. (3-5). The values of gamma corresponding to images (b) through (d) are 0.6, 0.4, and 0.3, respectively ($c = 1$ in all cases).

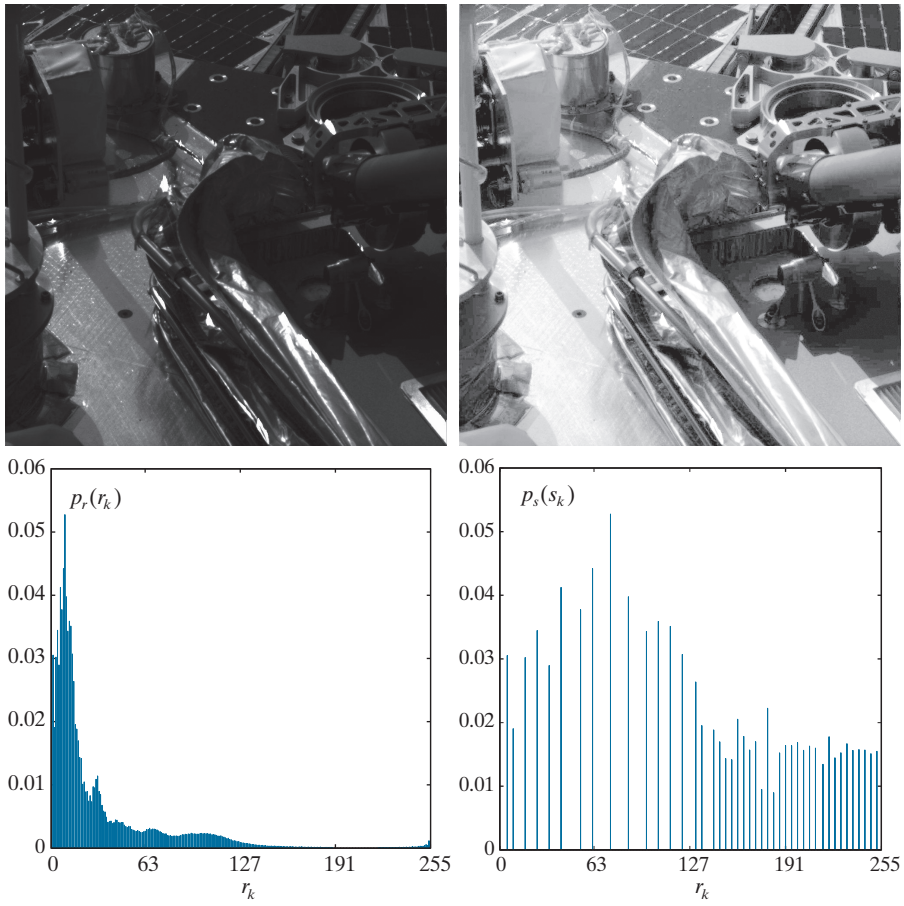


FIGURE 3.22
(a) Image from Phoenix Lander. (b) Result of histogram equalization. (c) Histogram of image (a). (d) Histogram of image (b). (Original image courtesy of NASA.)

where v is a dummy variable of integration. It follows from the preceding two equations that $G(z) = s = T(r)$ and, therefore, that z must satisfy the condition

$$z = G^{-1}(s) = G^{-1}\left[T(r)\right] \tag{3-19}$$

The transformation function $T(r)$ can be obtained using Eq. (3-17) after $p_r(r)$ has been estimated using the input image. Similarly, function $G(z)$ can be obtained from Eq. (3-18) because $p_z(z)$ is given.

Equations (3-17) through (3-19) imply that an image whose intensity levels have a specified PDF can be obtained using the following procedure:

1. Obtain $p_r(r)$ from the input image to use in Eq. (3-17).
2. Use the specified PDF, $p_z(z)$, in Eq. (3-18) to obtain the function $G(z)$.
3. Compute the inverse transformation $z = G^{-1}(s)$; this is a mapping from s to z, the latter being the values that have the specified PDF.
4. Obtain the output image by first equalizing the input image using Eq. (3-17); the

a b
c

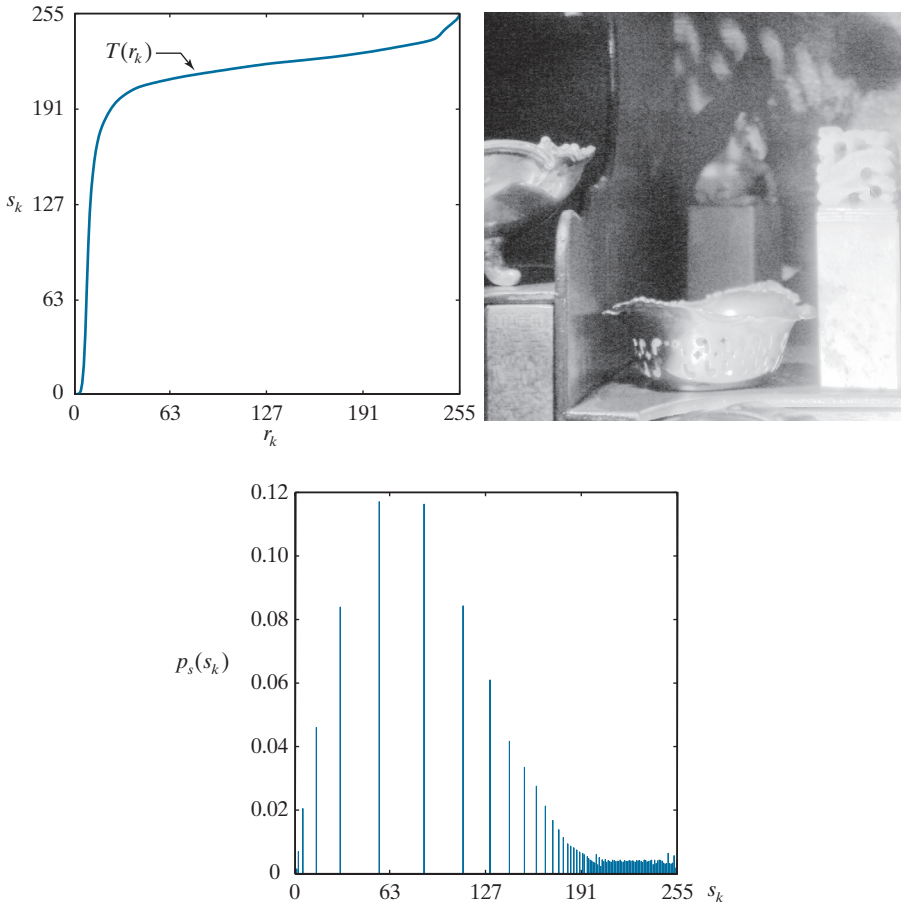


FIGURE 3.25
(a) Histogram equalization transformation obtained using the histogram in Fig. 3.24(b). (b) Histogram equalized image. (c) Histogram of equalized image.

is important to note that a rather modest change in the original histogram was all that was required to obtain a significant improvement in appearance.

Figure 3.26(d) shows the histogram of Fig. 3.26(c). The most distinguishing feature of this histogram is how its low end has shifted right toward the lighter region of the gray scale (but not excessively so), as desired. As you will see in the following section, we can do an even better job of enhancing Fig. 3.24(a) by using exact histogram specification.

EXACT HISTOGRAM MATCHING (SPECIFICATION)

The discrete histogram equalization and specification methods discussed in the preceding two sections generate images with histograms whose shapes generally do not resemble the shape of the specified histograms. You have seen already that these methods can produce effective results. However, there are applications that can benefit from a histogram processing technique capable of generating images whose histograms truly match specified shapes. Examples include normalizing large image data sets used for testing and validation in the pharmaceutical industry, establishing

a b
c d

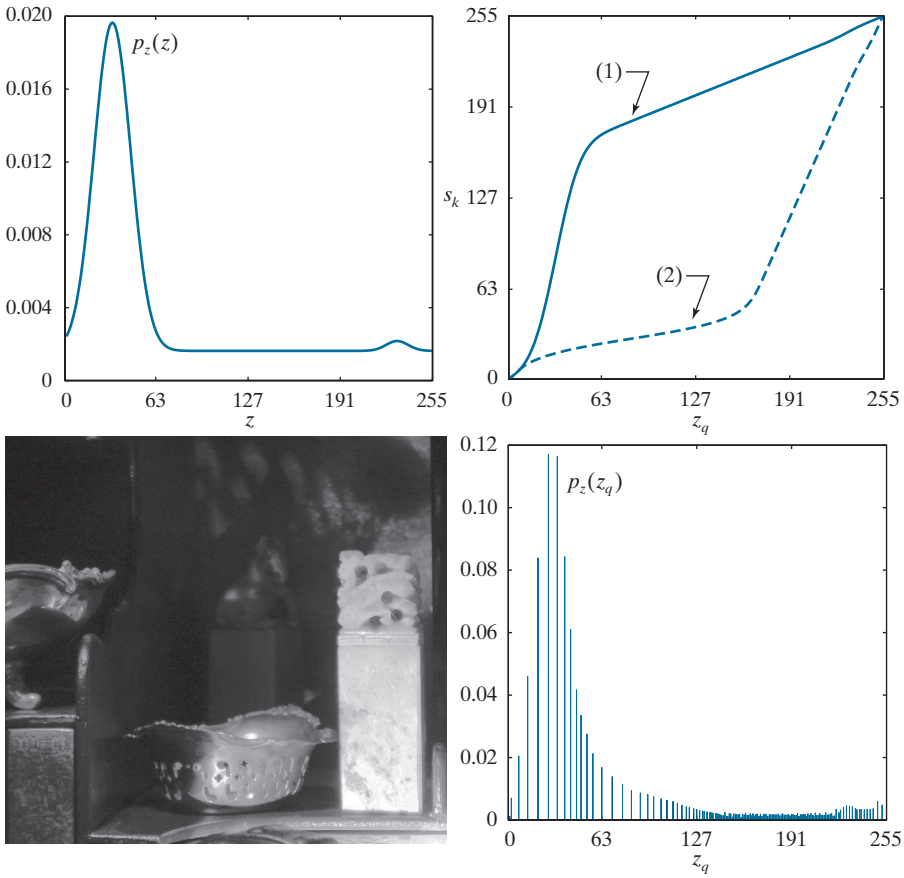


FIGURE 3.26 Histogram specification. (a) Specified histogram. (b) Transformation $G(z_q)$, labeled (1), and $G^{-1}(s_k)$, labeled (2). (c) Result of histogram specification. (d) Histogram of image (c).

a set of "golden images" for calibrating imaging systems, and establishing a norm for consistent medical image analysis and interpretation by humans. Also, as you will see later, being able to generate images with specified histograms simplifies experimentation when seeking histogram shapes that will produce a desired result.

The reason why discrete histogram equalization and specification do not produce exact specified histogram shapes is simple: they have no provisions for redistributing the intensities of an image to match a specified shape. In histogram equalization, changes in the number of pixels having a specific intensity occur as a result of rounding (see Example 3.5). Histogram specification also introduces changes as a result of matching values in the look-up table (see Example 3.7). However, the real impact on intensity values by these two methods results from shifting the histogram bins along the intensity scale. For example, the key difference between the histograms in Figs. 3.24 through 3.26 is the location of the histogram bins. For the histogram of the output image to have an *exact* specified shape, we have to find a way to change *and* redistribute the intensities of the pixels of the input image to create that shape. The following discussion shows how to do this.

Foundation

The following discussion is based on an approach developed by Coltuc, Bolon, and Chassery [2006] for implementing exact histogram specification. Consider a specified histogram that we wish an image to have:

$$H = \{h(0), h(1), h(2), \ldots, h(L-1)\} \tag{3-24}$$

Note that we are not using subscripts on the histogram elements, because we are working with a single type of histogram. This simplifies the notation considerably.

where L is the number of discrete intensity levels, and $h(j)$ is the number of pixels with intensity level j. This histogram is assumed to be both *unnormalized* and *valid*, in the sense that the sum of its components equals the total number of pixels in the image (which is always an integer):

$$\sum_{j=0}^{L-1} h(j) = MN \tag{3-25}$$

As usual, M and N are the number of rows and columns in the image, respectively.

Given a digital image and a histogram satisfying the preceding conditions, the procedure used for exact histogram specification consists of three basic steps:

We will give a more detailed set of steps later in this section.

(a) Order the image pixels according to a predefined criterion.
(b) Split the ordered pixels into L groups, such that group j has $h(j)$ pixels.
(c) Assign intensity value j to all pixels in group j.

Observe that we are both redistributing *and* changing the intensity of pixels of the output image to populate the bins in the specified histogram. Therefore, the output image is guaranteed to have that histogram, provided that Eq. (3-25) is satisfied.[†] The usefulness of the result depends on the ordering scheme used in Step (a).

See Section 2.6 regarding ordering.

Ordering: Consider a strict ordering relation on all MN pixels of an image so that

$$f(x_0, y_0) \prec f(x_1, y_1) \prec \cdots \prec f(x_{M-1,N-1}, y_{M-1,N-1}) \tag{3-26}$$

This equation represents a string of MN pixels ordered by a strict relation $\prec$, with the pair (x_i, y_i) denoting the coordinates of the ith pixel in the sequence. Keep in mind that $\prec$ may yield an ordering of pixels whose coordinates are not in any particular *spatial* sequence.

Recall from Section 2.6 that an ordering is based on the concept of *preceding*, and recall also that "preceding" is more general than just a numerical order. In a strict ordering, an element of a set (in this case the set of pixels in a digital image) cannot precede itself. In addition, if element a precedes element b, and element b precedes c in the ordered sequence, then this implies that a precedes c. In the present context,

[†]Because MN and L are fixed, and histogram bins must contain an integer number of pixels, a specified histogram may have to be adjusted sometimes in order to satisfy Eq. (3-25) (see Problem 3.16). In other words, there are instances in which the histogram matched by the method will have to be an altered version of an original specification (see Problem 3.18). Generally, the differences have negligible influence on the final result.

a b c

FIGURE 3.45 Result of filtering the test pattern in Fig. 3.42(a) using (a) zero padding, (b) mirror padding, and (c) replicate padding. A Gaussian kernel of size 187×187, with $K = 1$ and $\sigma = 31$ was used in all three cases.

EXAMPLE 3.16: Smoothing performance as a function of kernel and image size.

The amount of relative blurring produced by a smoothing kernel of a given size depends directly on image size. To illustrate, Fig. 3.46(a) shows the same test pattern used earlier, but of size 4096×4096 pixels, four times larger in each dimension than before. Figure 3.46(b) shows the result of filtering this image with the same Gaussian kernel and padding used in Fig. 3.45(b). By comparison, the former image shows considerably less blurring for the same size filter. In fact, Fig. 3.46(b) looks more like the image in Fig. 3.42(d), which was filtered using a 43×43 Gaussian kernel. In order to obtain results that are comparable to Fig. 3.45(b) we have to increase the size and standard deviation of the Gaussian kernel by four, the same factor as the increase in image dimensions. This gives a kernel of (odd) size

a b c

FIGURE 3.46 (a) Test pattern of size 4096×4096 pixels. (b) Result of filtering the test pattern with the same Gaussian kernel used in Fig. 3.45. (c) Result of filtering the pattern using a Gaussian kernel of size 745×745 elements, with $K = 1$ and $\sigma = 124$. Mirror padding was used throughout.

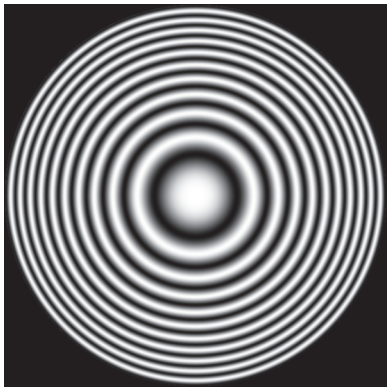


FIGURE 3.59 A zone plate image of size 597×597 pixels.

increases as a function of distance from the center, as you can see by noting that the rings get narrower the further they are from the center. This property makes a zone plate an ideal image for illustrating the behavior of the four filter types just discussed.

Figure 3.60(a) shows a 1-D, 128-element spatial lowpass filter function designed using MATLAB [compare with Fig. 3.38(b)]. As discussed earlier, we can use this 1-D function to construct a 2-D, separable lowpass filter kernel based on Eq. (3-51), or we can rotate it about its center to generate a 2-D, isotropic kernel. The kernel in Fig. 3.60(b) was obtained using the latter approach. Figures 3.61(a) and (b) are the results of filtering the image in Fig. 3.59 with the separable and isotropic kernels, respectively. Both filters passed the low frequencies of the zone plate while attenuating the high frequencies significantly. Observe, however, that the separable filter kernel produced a "squarish" (non-radially symmetric) result in the passed frequencies. This is a consequence of filtering the image in perpendicular directions with a separable kernel that is not isotropic. Using the isotropic kernel yielded a result that is uniform in all radial directions. This is as expected, because both the filter and the image are isotropic.

Figure 3.62 shows the results of filtering the zone plate with the four filters described in Table 3.7. We used the 2-D lowpass kernel in Fig. 3.60(b) as the basis for the highpass filter, and similar lowpass kernels for the bandreject filter. Figure 3.62(a) is the same as Fig. 3.61(b), which we repeat for convenience. Figure 3.62(b) is the highpass-filtered result. Note how effectively the low frequencies were filtered out. As is true of highpass-filtered images, the black areas were caused by negative values being clipped at 0 by the display. Figure 3.62(c) shows the same image scaled using Eqs. (2-31) and (2-32). Here we see clearly that only high frequencies were passed by the filter. Because the highpass kernel was constructed using the same lowpass kernel that we used to generate Fig. 3.62(a), it is evident by comparing the two results that the highpass filter passed the frequencies that were attenuated by the lowpass filter.

a b

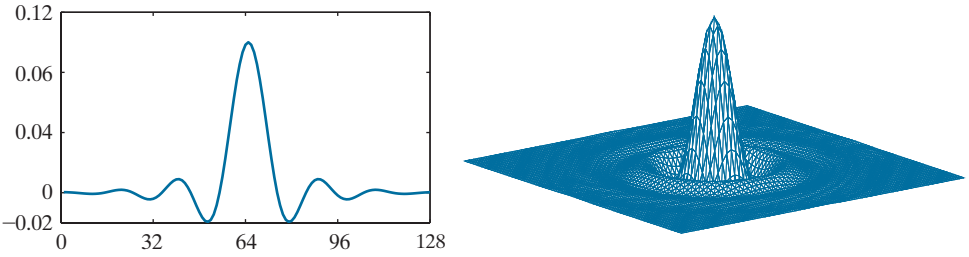


FIGURE 3.60 (a) A 1-D spatial lowpass filter function. (b) 2-D kernel obtained by rotating the 1-D profile about its center.

a b

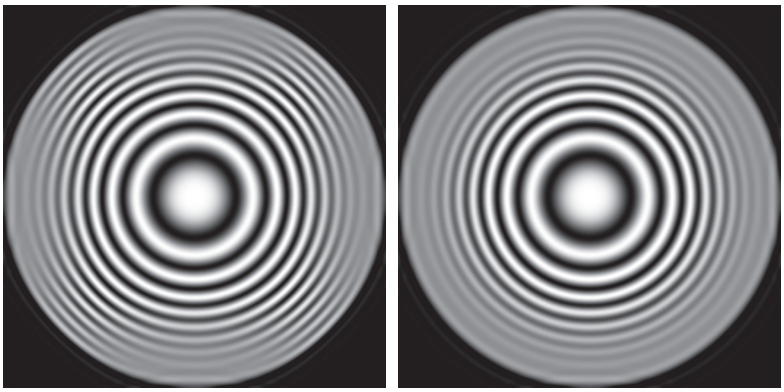
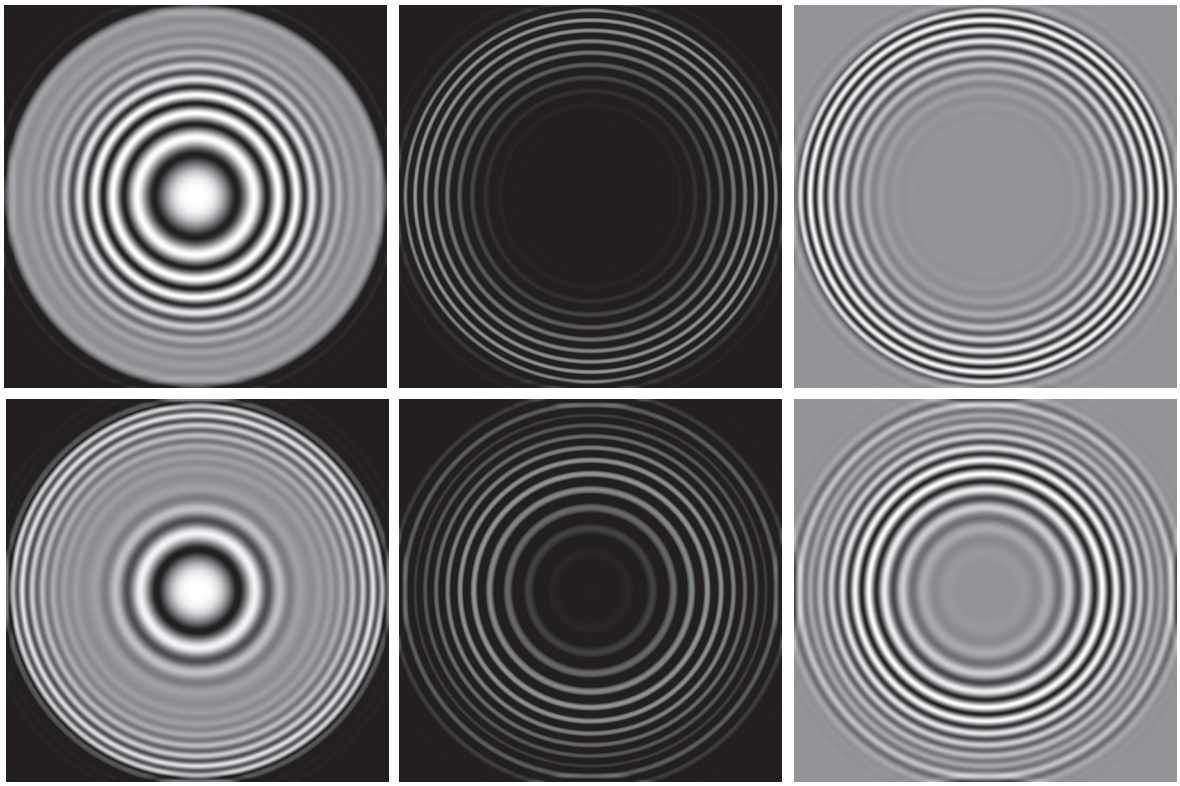


FIGURE 3.61
(a) Zone plate image filtered with a separable lowpass kernel. (b) Image filtered with the isotropic lowpass kernel in Fig. 3.60(b).



a b c
d e f

FIGURE 3.62
Spatial filtering of the zone plate image. (a) Lowpass result; this is the same as Fig. 3.61(b). (b) Highpass result. (c) Image (b) with intensities scaled. (d) Bandreject result. (e) Bandpass result. (f) Image (e) with intensities scaled.

for $j = 1, 2, \ldots, R$, where $\mu_{A_{jk}}(z_k)$ is the membership function of fuzzy set A_{jk} evaluated at the value of the kth input, and λ_j is called the *strength* (or *firing*) *level* of the jth rule. With reference to our discussion earlier, λ_j is simply the value used to clip the output function of the jth rule (see Step 3 in the summary related to Fig. 3.72).

We will illustrate how ELSE rules are used when we discuss fuzzy filtering later in this section.

The strength level of the ELSE rule is defined so that it increases as the strength of the THEN rules weaken, and vice versa. An ELSE rule may be viewed as performing a NOT operation on the results of the other rules. We know from Eq. (3-77) that $\mu_{\text{NOT}(A)}(z) = \mu_{\bar{A}}(z) = 1 - \mu_A(z)$. Using this idea in combining (ANDing) the strengths of the THEN rules gives the following strength level for the ELSE rule:

$$\lambda_E = \min\left\{1 - \lambda_j;\ j = 1, 2, \ldots, R\right\} \tag{3-96}$$

We see that if any of the THEN rules fires at "full strength" (its responses is 1), then the strength level of the ELSE rule will be 0. As the responses of the THEN rules weaken, the strength of the ELSE rule increases. The value of λ_E is used to clip fuzzy set B_E, in exactly the same manner that the other output fuzzy sets are clipped. The only difference is that the clipping value for the ELSE rule is determined by the strength levels of the other rules, rather than by the results of implication (see Fig. 3.72). The resulting, clipped fuzzy set B_E is used in the aggregation step, together with the other clipped output fuzzy sets.

When dealing with ORs in the antecedents, we replace the ANDs in Eq. (3-94) by ORs and the min in Eq. (3-95) by a max; Eq. (3-96) does not change. Although more complex antecedents and consequents than the ones discussed here could be developed, formulations using only ANDs or ORs are quite general and are used in a broad spectrum of image processing applications. The references at the end of this chapter contain additional (but used less frequently) definitions of fuzzy logical operators, and discuss other methods for implication (including multiple outputs) and defuzzification. In the next two sections, we will show how to apply fuzzy concepts to image processing.

USING FUZZY SETS FOR INTENSITY TRANSFORMATIONS

One of the principal applications of intensity transformations is contrast enhancement. We can state the process of enhancing the contrast of a grayscale image using the following rules:

IF a pixel is *dark*, THEN make it *darker*.

IF a pixel is *gray*, THEN make it *gray*.

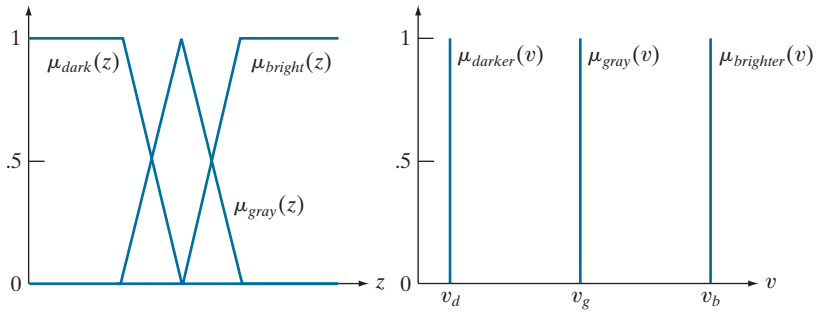
IF a pixel is *bright*, THEN make it *brighter*.

Keeping in mind that these are fuzzy terms, we can express the concepts of *dark*, *gray*, and *bright* by the membership functions in Fig. 3.73(a).

In terms of the output, we consider *darker* as being degrees of a dark intensity value (100% black being the limiting shade of dark), *brighter*, as being degrees of

a b

FIGURE 3.73 (a) Input and (b) output membership functions for fuzzy, rule-based contrast enhancement.



a bright shade (100% white being the limiting value), and *gray* as being degrees of an intensity in the middle of the gray scale. By "degrees" we mean the amount of a specific intensity. For example, 80% black is a dark gray. When interpreted as *constant* intensities whose strength is modified, the output membership functions are *singletons* (membership functions that are *constant*), as Fig. 3.73(b) shows. The degrees of an intensity in the range [0, 1] occur when the singletons are clipped by the strength of the response from their corresponding rules, as in the fourth column of Fig. 3.72 (we are working here with only one input, not two, as in the figure). Because we are dealing with constants in the output membership functions, it follows from Eq. (3-93) that the output, v_0, due to any input, z_0, is given by

$$v_0 = \frac{\mu_{dark}(z_0) \times v_d + \mu_{gray}(z_0) \times v_g + \mu_{bright}(z_0) \times v_b}{\mu_{dark}(z_0) + \mu_{gray}(z_0) + \mu_{bright}(z_0)} \tag{3-97}$$

where "×" indicates scalar multiplication. The summations in the numerator and denominator in this expression are simpler than in Eq. (3-93) because the output membership functions are constants modified (clipped) by the fuzzified values.

Fuzzy image processing is computationally intensive because the entire process of fuzzification, processing the antecedents of all rules, implication, aggregation, and defuzzification must be applied to every pixel in the input image. Thus, using singletons as in Eq. (3-97) reduces computational requirements significantly by simplifying implication, aggregation, and defuzzification. These savings can be important in applications that require high processing speeds.

EXAMPLE 3.25: Fuzzy, rule-based contrast modification.

Figure 3.74(a) shows an image whose predominant intensities span a narrow range of the gray scale [see the image histogram in Fig. 3.75(a)], and thus has the appearance of low contrast. As a basis for comparison, Fig. 3.74(b) is the result of histogram equalization. As the histogram of this image shows [see Fig. 3.75(b)], expanding the entire gray scale does increase contrast, but also spreads intensities on the high and low end that give the image an "overexposed" appearance. For example, the details in Professor Einstein's forehead and hair are mostly lost. Figure 3.74(c) shows the result of using the rule-based contrast modification approach discussed above. Figure 3.75(c) shows the input membership functions

a b c

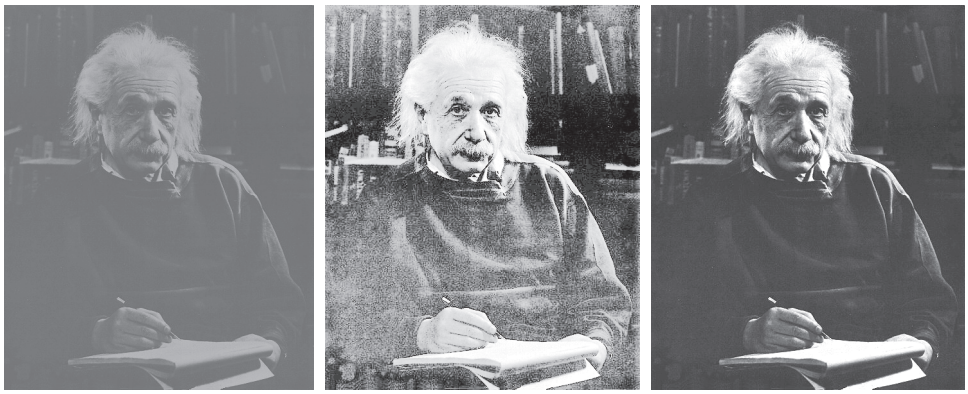


FIGURE 3.74
(a) Low-contrast image. (b) Result of histogram equalization. (c) Result of using fuzzy, rule-based contrast enhancement.

used, superimposed on the histogram of the original image. The output singletons were selected at $v_d = 0$ (black), $v_g = 127$ (mid gray), and $v_b = 255$ (white).

Comparing Figs. 3.74(b) and (c), we see in the latter a considerable improvement in tonality. Note, for example, the level of detail in the forehead and hair, as compared to the same regions in Fig. 3.74(b). The reason for the improvement can be explained easily by studying the histogram of Fig. 3.74(c), shown in Fig. 3.75(d). Unlike the histogram of the equalized image, this histogram has kept the same basic characteristics of the histogram of the original image. However, the dark levels (tall peaks in the low end of the histogram) were moved left, thus darkening the levels. The opposite was true for bright levels. The mid grays were spread slightly, but much less than in histogram equalization.

a b
c d

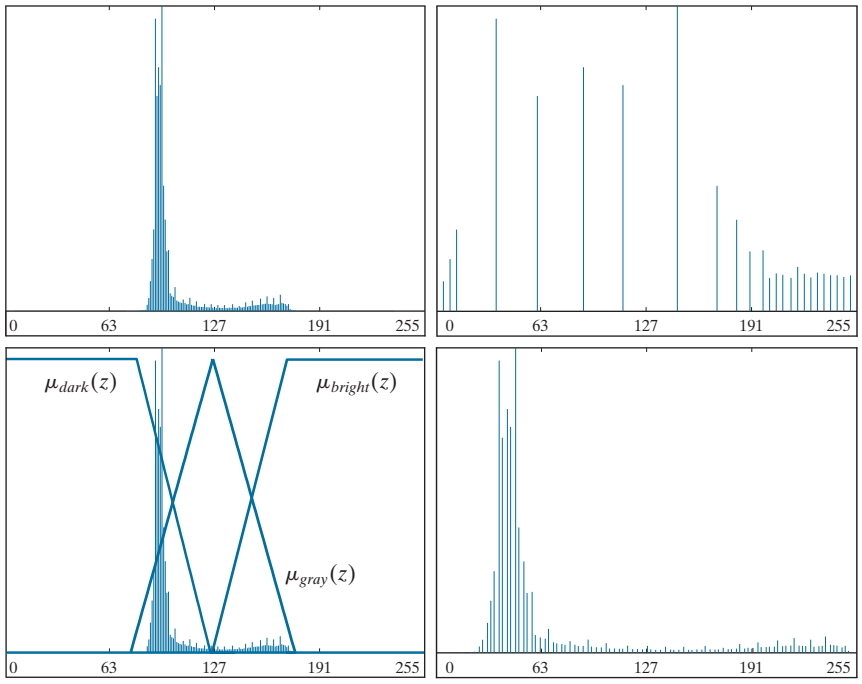
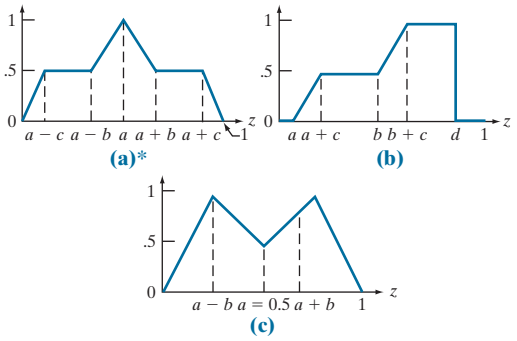


FIGURE 3.75
(a) Histogram of Fig. 3.74(a). (b) Histogram of Fig. 3.74(b). (c) Input membership functions superimposed on (a). (d) Histogram of Fig. 3.74(c).



3.61 Because the term z_0 is a constant in both μ_{red} and μ_3, show that Eq. (3-88) can be written as $Q_3(v) = \min\{\mu_{red}(z_0), \mu_{mat}(v)\}$.

3.62* What would be the effect of increasing the neighborhood size in the fuzzy filtering approach discussed at the end of Section 3.9? Explain. (You may use an example to support your answer).

3.63 You are employed to design a fuzzy, rule-based filtering system for reducing the effects of impulse noise on a noisy image with intensity values in the interval $[0, L-1]$. As in the filtering approach discussed at the end of Section 3.9, use only the differences d_2, d_4, d_6, and d_8 in a 3×3 neighborhood in order to simplify the problem. Let z_5 denote the intensity at the center of the neighborhood. The corresponding output intensity values should be $z_5' = z_5 + v$, where v is the output of your fuzzy system. That is, the output of your fuzzy system is a correction factor used to reduce the effect of a noise spike that may be present at the center of the 3×3 neighborhood. Assume that the noise spikes occur sufficiently apart so that you need not be concerned with multiple noise spikes being present in the same neighborhood. The spikes can be dark or light. Use triangular membership functions throughout.

(a)* Give a fuzzy system for this problem.

(b)* Specify the IF-THEN and ELSE rules.

(c) Specify the membership functions graphically, as in Fig. 3.77.

(d) Show a graphical representation of the rule set, as in Fig. 3.78.

(e) Give a summary diagram of your fuzzy system similar to the one in Fig. 3.72.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

3.1* Write a function **g = imPad4e(f,r,c,padtype,loc)** for padding image **f** with **r** rows above and below the image, and **c** columns to the left and right. If **padtype = 'zeros'**, or is omitted from the argument, the function should implement zero padding. If **padtype = 'replicate'**, replicate padding, as defined in Section 3.5, should be used. If **loc** is specified as **loc = 'post'**, the function should behave as above, except that **r** rows are placed only below the image and **c** columns are placed only to the right of it.

3.2 Intensity transformation of grayscale images.

(a) Write a function **[g,map] = intXform4e(f,mode,-param)** for transforming the intensities of an input 8-bit grayscale image **f**. The intensities of **f** (and output image **g**) are assumed to be in the range $[0, 1]$ (use function **intScaling4e** from Chapter 2 in the body of **intXform4e** to make the conversion to $[0, 1]$ automatic). The type of transformation performed is specified in parameter **mode**, a character string with values: **'negative'**, **'log'**, **'gamma'**, or **'external'**. The first two specifications implement Eqs. (3-3) and (3-4), (with $c = 1.0$). The third specification implements Eq. (3-5), in which case **param** is a scalar equal to the value of γ (use $c = 1.0$). Specifying **mode** as **'external'** means that the user is specifying the transformation function (e.g., for histogram equalization), whose values must be in the range $[0, 1]$ and be provided as a 1-D array in **param**. On the output, **map** is the transformation function computed by **intXform4e** (or provided by the user if **'external'** was specified for **mode**).

(b) Read and display the image **spillway-dark.tif**. Apply a log transformation function to it. Display the result.

4 Filtering in the Frequency Domain

Filter: A device or material for suppressing or minimizing waves or oscillations of certain frequencies.

Frequency: The number of times that a periodic function repeats the same sequence of values during a unit variation of the independent variable.

Webster's New Collegiate Dictionary

Preview

After a brief historical introduction to the Fourier transform and its importance in image processing, we start from basic principles of function sampling, and proceed step-by-step to derive the one- and two-dimensional discrete Fourier transforms. Together with convolution, the Fourier transform is a staple of frequency-domain processing. During this development, we also touch upon several important aspects of sampling, such as aliasing, whose treatment requires an understanding of the frequency domain and thus are best covered in this chapter. This material is followed by a formulation of filtering in the frequency domain, paralleling the spatial filtering techniques discussed in Chapter 3. We conclude the chapter with a derivation of the equations underlying the fast Fourier transform (FFT), and discuss its computational advantages. These advantages make frequency-domain filtering practical and, in many instances, superior to filtering in the spatial domain.

Upon completion of this chapter, readers should:

- Understand the meaning of frequency domain filtering, and how it differs from filtering in the spatial domain.
- Be familiar with the concepts of sampling, function reconstruction, and aliasing.
- Understand convolution in the frequency domain, and how it is related to filtering.
- Know how to obtain frequency domain filter functions from spatial kernels, and vice versa.
- Be able to construct filter transfer functions directly in the frequency domain.
- Understand why image padding is important.
- Know the steps required to perform filtering in the frequency domain.
- Understand when frequency domain filtering is superior to filtering in the spatial domain.
- Be familiar with other filtering techniques in the frequency domain, such as unsharp masking and homomorphic filtering.
- Understand the origin and mechanics of the fast Fourier transform, and how to use it effectively in image processing.

4.1 BACKGROUND

We begin the discussion with a brief outline of the origins of the Fourier transform and its impact on countless branches of mathematics, science, and engineering.

A BRIEF HISTORY OF THE FOURIER SERIES AND TRANSFORM

The French mathematician Jean Baptiste Joseph Fourier was born in 1768 in the town of Auxerre, about midway between Paris and Dijon. The contribution for which he is most remembered was outlined in a memoir in 1807, and later published in 1822 in his book, *La Théorie Analitique de la Chaleur* (*The Analytic Theory of Heat*). This book was translated into English 55 years later by Freeman (see Freeman [1878]). Basically, Fourier's contribution in this field states that any periodic function can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient (we now call this sum a *Fourier series*). It does not matter how complicated the function is; if it is periodic and satisfies some mild mathematical conditions, it can be represented by such a sum. This is taken for granted now but, at the time it first appeared, the concept that complicated functions could be represented as a sum of simple sines and cosines was not at all intuitive (see Fig. 4.1). Thus, it is not surprising that Fourier's ideas were met initially with skepticism.

Functions that are not periodic (but whose area under the curve is finite) can be expressed as the integral of sines and/or cosines multiplied by a weighting function. The formulation in this case is the *Fourier transform*, and its utility is even greater than the Fourier series in many theoretical and applied disciplines. Both representations share the important characteristic that a function, expressed in either a Fourier series or transform, can be reconstructed (recovered) completely via an inverse process, with no loss of information. This is one of the most important characteristics of these representations because it allows us to work in the *Fourier domain* (generally called the *frequency domain*) and then return to the original domain of the function without losing any information. Ultimately, it is the utility of the Fourier series and transform in solving practical problems that makes them widely studied and used as fundamental tools.

The initial application of Fourier's ideas was in the field of heat diffusion, where they allowed formulation of differential equations representing heat flow in such a way that solutions could be obtained for the first time. During the past century, and especially in the past 60 years, entire industries and academic disciplines have flourished as a result of Fourier's initial ideas. The advent of digital computers and the "discovery" of a fast Fourier transform (FFT) algorithm in the early 1960s revolutionized the field of signal processing. These two core technologies allowed for the first time practical processing of a host of signals of exceptional importance, ranging from medical monitors and scanners to modern electronic communications.

As you learned in Section 3.4, it takes on the order of $MNmn$ operations (multiplications and additions) to filter an $M \times N$ image with a kernel of size $m \times n$ elements. If the kernel is separable, the number of operations is reduced to $MN(m + n)$. In Section 4.11, you will learn that it takes on the order of $2MN \log_2 MN$ operations to perform the equivalent filtering process in the frequency domain, where the 2 in front arises from the fact that we have to compute a forward and an inverse FFT.

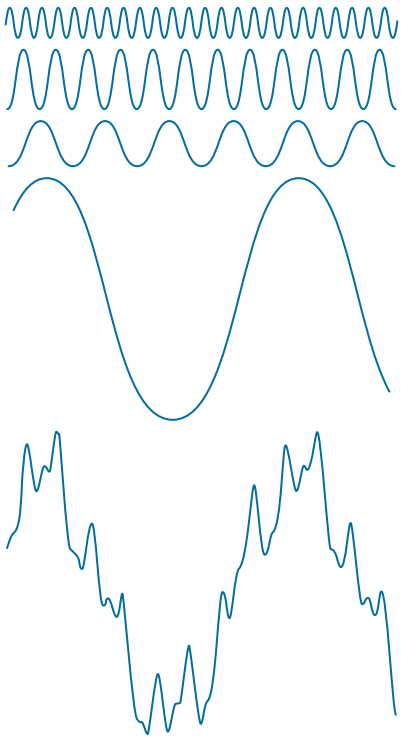


FIGURE 4.1
The function at the bottom is the sum of the four functions above it. Fourier's idea in 1807 that periodic functions could be represented as a weighted sum of sines and cosines was met with skepticism.

To get an idea of the relative computational advantages of filtering in the frequency versus the spatial domain, consider square images and kernels, of sizes $M \times M$ and $m \times m$, respectively. The *computational advantage* (as a function of kernel size) of filtering one such image with the FFT as opposed to using a nonseparable kernel is defined as

$$C_n(m) = \frac{M^2 m^2}{2M^2 \log_2 M^2} = \frac{m^2}{4 \log_2 M} \tag{4-1}$$

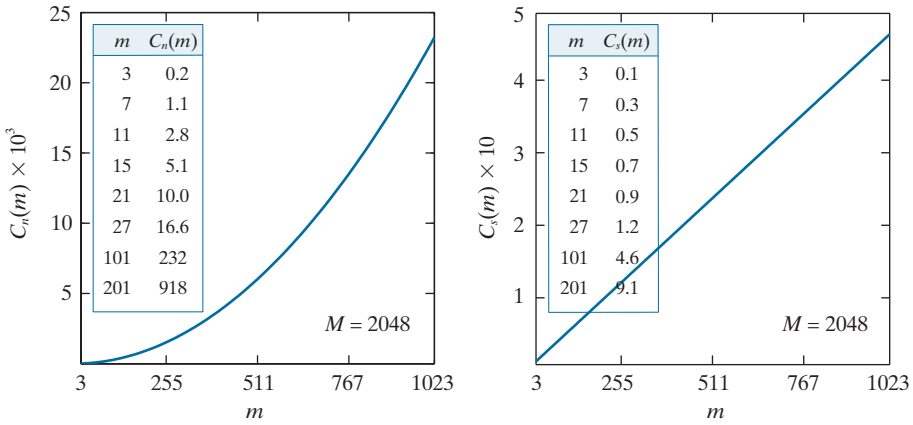
If the kernel is separable, the advantage becomes

$$C_s(m) = \frac{2M^2 m}{2M^2 \log_2 M^2} = \frac{m}{2 \log_2 M} \tag{4-2}$$

In either case, when $C(m) > 1$ the advantage (in terms of fewer computations) belongs to the FFT approach; otherwise the advantage favors spatial filtering.

a b

FIGURE 4.2
(a) Computational advantage of the FFT over non-separable spatial kernels.
(b) Advantage over separable kernels. The numbers for $C(m)$ in the inset tables are not to be multiplied by the factors of 10 shown for the curves.



The computational advantages given by Eqs. (4-1) and (4-2) do not take into account the fact that the FFT performs operations between complex numbers, and other secondary (but small in comparison) computations discussed later in the chapter. Thus, comparisons should be interpreted only as guidelines,

Figure 4.2(a) shows a plot of $C_n(m)$ as a function of m for an image of intermediate size ($M = 2048$). The inset table shows a more detailed look for smaller kernel sizes. As you can see, the FFT has the advantage for kernels of sizes 7×7 and larger. The advantage grows rapidly as a function of m, being over 200 for $m = 101$, and close to 1000 for $m = 201$. To give you a feel for the meaning of this advantage, if filtering a bank of images of size 2048×2048 takes 1 minute with the FFT, it would take on the order of 17 hours to filter the same set of images with a nonseparable kernel of size 201×201 elements. This is a significant difference, and is a clear indicator of the importance of frequency-domain processing using the FFT.

In the case of separable kernels, the computational advantage is not as dramatic, but it is still meaningful. The "cross over" point now is around $m = 27$, and when $m = 101$ the difference between frequency- and spatial-domain filtering is still manageable. However, you can see that with $m = 201$ the advantage of using the FFT approaches a factor of 10, which begins to be significant. Note in both graphs that the FFT is an overwhelming favorite for large spatial kernels.

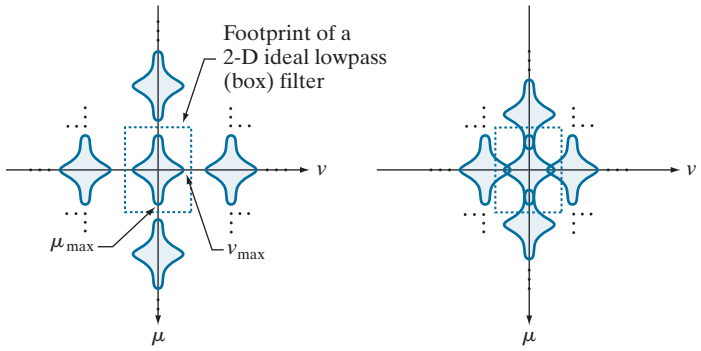
Our focus in the sections that follow is on the Fourier transform and its properties. As we progress through this chapter, it will become evident that Fourier techniques are useful in a broad range of image processing applications. We conclude the chapter with a discussion of the FFT.

ABOUT THE EXAMPLES IN THIS CHAPTER

As in Chapter 3, most of the image filtering examples in this chapter deal with image enhancement. For example, smoothing and sharpening are traditionally associated with image enhancement, as are techniques for contrast manipulation. By its very nature, beginners in digital image processing find enhancement to be interesting and relatively simple to understand. Therefore, using examples from image enhancement in this chapter not only saves having an extra chapter in the book but, more importantly, is an effective tool for introducing newcomers to filtering techniques in the frequency domain. We will use frequency domain processing methods for other applications in Chapters 5, 7, 8, 10, and 12.

a b

FIGURE 4.16 Two-dimensional Fourier transforms of (a) an over-sampled, and (b) an under-sampled, band-limited function.



$$\frac{1}{\Delta T} > 2\mu_{\max} \tag{4-65}$$

and

$$\frac{1}{\Delta Z} > 2\nu_{\max} \tag{4-66}$$

Stated another way, we say that no information is lost if a 2-D, band-limited, continuous function is represented by samples acquired at rates greater than twice the highest frequency content of the function in both the μ- and ν-directions.

Figure 4.16 shows the 2-D equivalents of Figs. 4.6(b) and (d). A 2-D ideal filter transfer function has the form illustrated in Fig. 4.14(a) (but in the frequency domain). The dashed portion of Fig. 4.16(a) shows the location of the filter function to achieve the necessary isolation of a single period of the transform for reconstruction of a band-limited function from its samples, as in Fig. 4.8. From Fig 4.10, we know that if the function is under-sampled, the periods overlap, and it becomes impossible to isolate a single period, as Fig. 4.16(b) shows. Aliasing would result under such conditions.

ALIASING IN IMAGES

In this section, we extend the concept of aliasing to images, and discuss in detail several aspects of aliasing related to image sampling and resampling.

Extensions from 1-D Aliasing

As in the 1-D case, a continuous function $f(t,z)$ of two continuous variables, t and z, can be band-limited in general only if it extends infinitely in both coordinate directions. The very act of limiting the spatial duration of the function (e.g., by multiplying it by a box function) introduces corrupting frequency components extending to infinity in the frequency domain, as explained in Section 4.3 (see also Problem 4.15). Because we cannot sample a function infinitely, aliasing is always present in digital images, just as it is present in sampled 1-D functions. There are two principal manifestations of aliasing in images: spatial aliasing and temporal aliasing. *Spatial aliasing* is caused by under-sampling, as discussed in Section 4.3, and tends to be more visible

(and objectionable) in images with repetitive patterns. *Temporal aliasing* is related to time intervals between images of a sequence of dynamic images. One of the most common examples of temporal aliasing is the "wagon wheel" effect, in which wheels with spokes in a sequence of images (for example, in a movie) appear to be rotating backwards. This is caused by the frame rate being too low with respect to the speed of wheel rotation in the sequence, and is similar to the phenomenon described in Fig. 4.11, in which under sampling produced a signal that appeared to be of much lower frequency than the original.

Our focus in this chapter is on spatial aliasing. The key concerns with spatial aliasing in images are the introduction of artifacts such as jaggedness in line features, spurious highlights, and the appearance of frequency patterns not present in the original image. Just as we used Fig. 4.9 to explain aliasing in 1-D functions, we can develop an intuitive grasp of the nature of aliasing in images using some simple graphics. The sampling grid in the center section of Fig. 4.17 is a 2-D representation of the impulse train in Fig. 4.15. In the grid, the little white squares correspond to the location of the impulses (where the image is sampled) and black represents the separation between samples. Superimposing the sampling grid on an image is analogous to multiplying the image by an impulse train, so the same sampling concepts we discussed in connection with the impulse train in Fig. 4.15 are applicable here. The focus now is to analyze graphically the interaction between sampling rate (the separation of the sampling points in the grid) and the frequency of the 2-D signals being sampled.

Figure 4.17 shows a sampling grid partially overlapping three 2-D signals (regions of an image) of low, mid, and high spatial frequencies (relative to the separation between sampling cells in the grid). Note that the level of spatial "detail" in the regions is proportional to frequency (i.e., higher-frequency signals contain more bars). The sections of the regions inside the sampling grip are rough manifestations of how they would appear after sampling. As expected, all three digitized regions

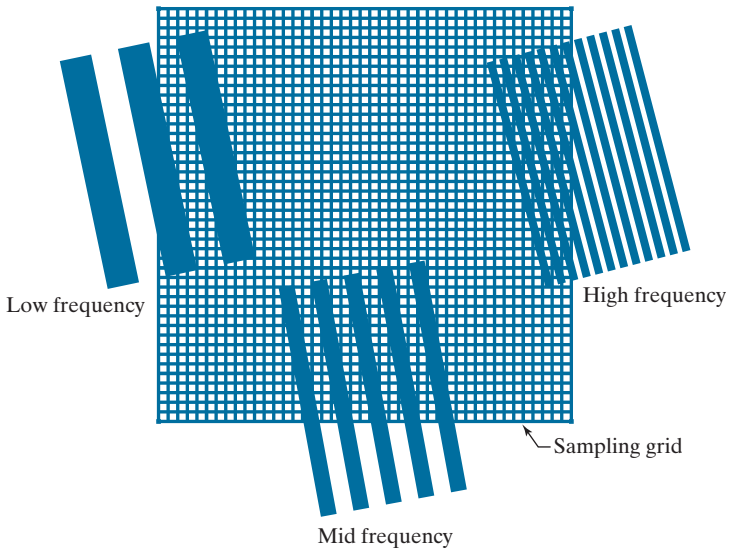
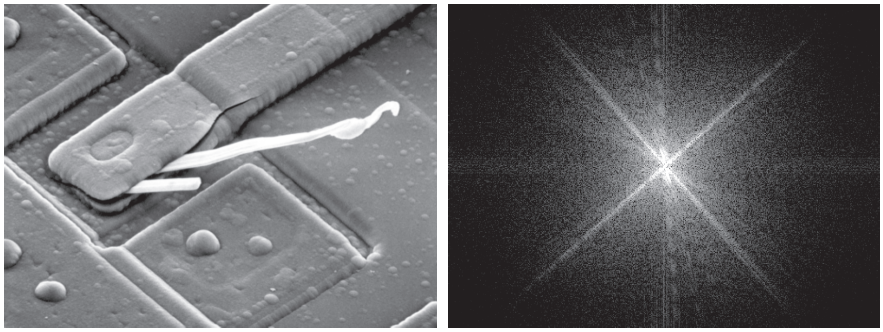


FIGURE 4.17 Various aliasing effects resulting from the interaction between the frequency of 2-D signals and the sampling rate used to digitize them. The regions outside the sampling grid are continuous and free of aliasing.



a b

FIGURE 4.28 (a) SEM image of a damaged integrated circuit. (b) Fourier spectrum of (a). (Original image courtesy of Dr. J. M. Hudak, Brockhouse Institute for Materials Research, McMaster University, Hamilton, Ontario, Canada.)

between frequency content and rate of change of intensity levels in an image, can lead to some very useful results. We will show in Section 4.8 the effects of modifying various frequency ranges in the transform of Fig. 4.28(a).

FREQUENCY DOMAIN FILTERING FUNDAMENTALS

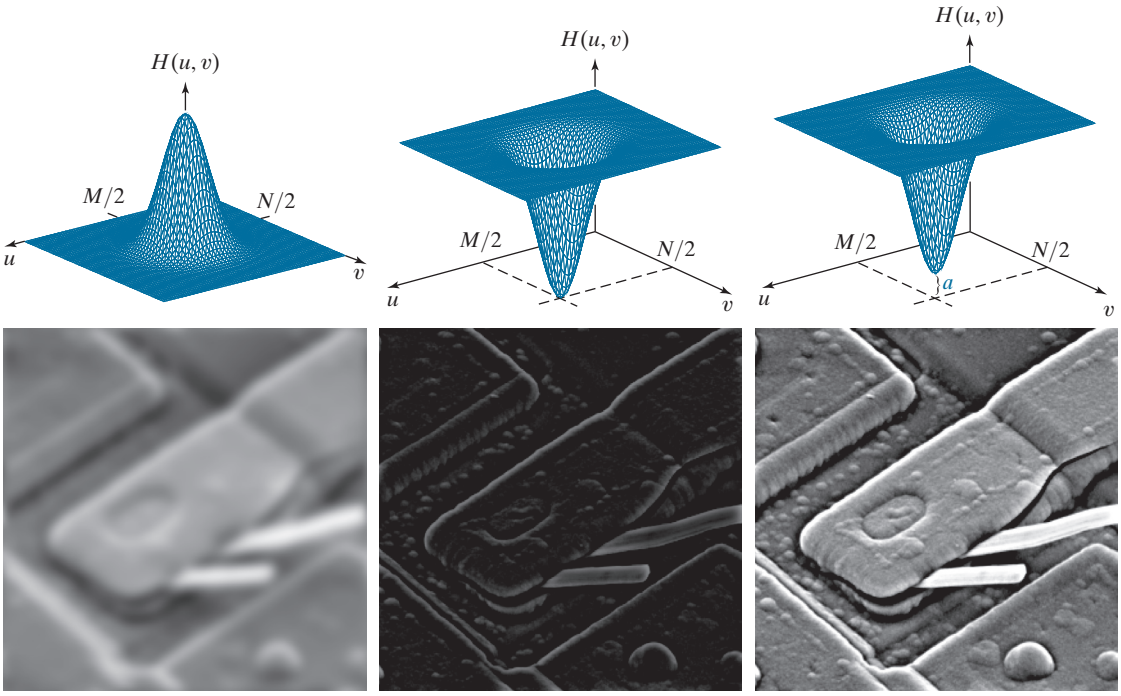
Filtering in the frequency domain consists of modifying the Fourier transform of an image, then computing the inverse transform to obtain the spatial domain representation of the processed result. Thus, given (a padded) digital image, $f(x,y)$, of size $P \times Q$ pixels, the basic filtering equation in which we are interested has the form:

If H is real and symmetric and f is real (as is typically the case), then the IDFT in Eq. (4-104) should yield real quantities in theory. In practice, the inverse often contains parasitic complex terms from roundoff error and other computational inaccuracies. Thus, it is customary to take the real part of the IDFT to form g.

$$g(x,y) = \text{Real}\left\{\Im^{-1}\left[H(u,v)F(u,v)\right]\right\} \tag{4-104}$$

where $\Im^{-1}$ is the IDFT, $F(u,v)$ is the DFT of the input image, $f(x,y)$, $H(u,v)$ is a *filter transfer function* (which we often call just a *filter* or *filter function*), and $g(x,y)$ is the *filtered* (*output*) *image*. Functions F, H, and g are arrays of size $P \times Q$, the same as the padded input image. The product $H(u,v)F(u,v)$ is formed using elementwise multiplication, as defined in Section 2.6. The filter transfer function modifies the transform of the input image to yield the processed output, $g(x,y)$. The task of specifying $H(u,v)$ is simplified considerably by using functions that are symmetric about their center, which requires that $F(u,v)$ be centered also. As explained in Section 4.6, this is accomplished by multiplying the input image by $(-1)^{x+y}$ prior to computing its transform.[†]

[†] Some software implementations of the 2-D DFT (e.g., MATLAB) do not center the transform. This implies that filter functions must be arranged to correspond to the same data format as the uncentered transform (i.e., with the origin at the top left). The net result is that filter transfer functions are more difficult to generate and display. We use centering in our discussions to aid in visualization, which is crucial in developing a clear understanding of filtering concepts. Either method can be used in practice, provided that consistency is maintained.



a b c
d e f

FIGURE 4.30 Top row: Frequency domain filter transfer functions of (a) a lowpass filter, (b) a highpass filter, and (c) an offset highpass filter. Bottom row: Corresponding filtered images obtained using Eq. (4-104). The offset in (c) is $a = 0.85$, and the height of $H(u,v)$ is 1. Compare (f) with Fig. 4.28(a).

a b c

FIGURE 4.31 (a) A simple image. (b) Result of blurring with a Gaussian lowpass filter without padding. (c) Result of lowpass filtering with zero padding. Compare the vertical edges in (b) and (c).

a b

FIGURE 4.56
(a) Original, blurry image. (b) Image enhanced using the Laplacian in the frequency domain. Compare with Fig. 3.52(d). (Original image courtesy of NASA.)

encompasses a very small neighborhood, while the formulation in Eqs. (4-125) and (4-126) encompasses the entire image.

UNSHARP MASKING, HIGH-BOOST FILTERING, AND HIGH-FREQUENCY-EMPHASIS FILTERING

In this section, we discuss frequency domain formulations of the unsharp masking and high-boost filtering image sharpening techniques introduced in Section 3.6.3. Using frequency domain methods, the mask defined in Eq. (3.6-8) is given by

$$g_{\text{mask}}(x,y) = f(x,y) - f_{\text{LP}}(x,y) \tag{4-128}$$

with

$$f_{\text{LP}}(x,y) = \Im^{-1}\left[H_{\text{LP}}(u,v)F(u,v)\right] \tag{4-129}$$

where $H_{\text{LP}}(u,v)$ is a lowpass filter transfer function, and $F(u,v)$ is the DFT of $f(x,y)$. Here, $f_{\text{LP}}(x,y)$ is a smoothed image analogous to $\bar{f}(x,y)$ in Eq. (3-78). Then, as in Eq. (3-79),

$$g(x,y) = f(x,y) + kg_{\text{mask}}(x,y) \tag{4-130}$$

This expression defines *unsharp masking* when $k = 1$ and high-boost filtering when $k > 1$. Using the preceding results, we can express Eq. (4-130) entirely in terms of frequency domain computations involving a lowpass filter:

$$g(x,y) = \Im^{-1}\left\{\Big(1 + k\big[1 - H_{\text{LP}}(u,v)\big]\Big)F(u,v)\right\} \tag{4-131}$$

We can express this result in terms of a highpass filter using Eq. (4-118):

$$g(x,y) = \Im^{-1}\left\{\left[1 + kH_{\text{HP}}(u,v)\right]F(u,v)\right\} \tag{4-132}$$

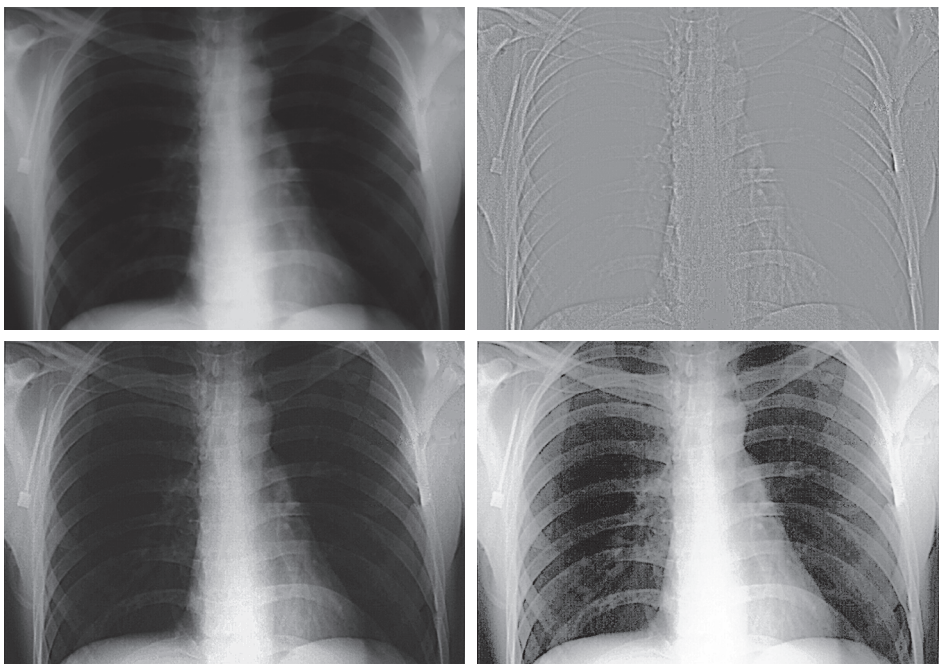
The expression contained within the square brackets is called a *high-frequency-emphasis filter transfer function*. As noted earlier, highpass filters set the dc term to zero, thus reducing the average intensity in the filtered image to 0. The high-frequency-emphasis filter does not have this problem because of the 1 that is added to the highpass filter transfer function. Constant k gives control over the proportion of high frequencies that influences the final result. A slightly more general formulation of high-frequency-emphasis filtering is the expression

$$g(x,y) = \Im^{-1}\left\{\left[k_1 + k_2 H_{\text{HP}}(u,v)\right]F(u,v)\right\} \tag{4-133}$$

where $k_1 \geq 0$ offsets the value the transfer function so as not to zero-out the dc term [see Fig. 4.30(c)], and $k_2 > 0$ controls the contribution of high frequencies.

EXAMPLE 4.22: Image enhancement using high-frequency-emphasis filtering.

Figure 4.57(a) shows a 503×720-pixel chest X-ray image with a narrow range of intensity levels. The objective of this example is to enhance the image using high-frequency-emphasis filtering. X-rays cannot be focused in the same manner that optical lenses can, and the resulting images generally tend to be slightly blurred. Because the intensities in this particular image are biased toward the dark end of the



a b
c d

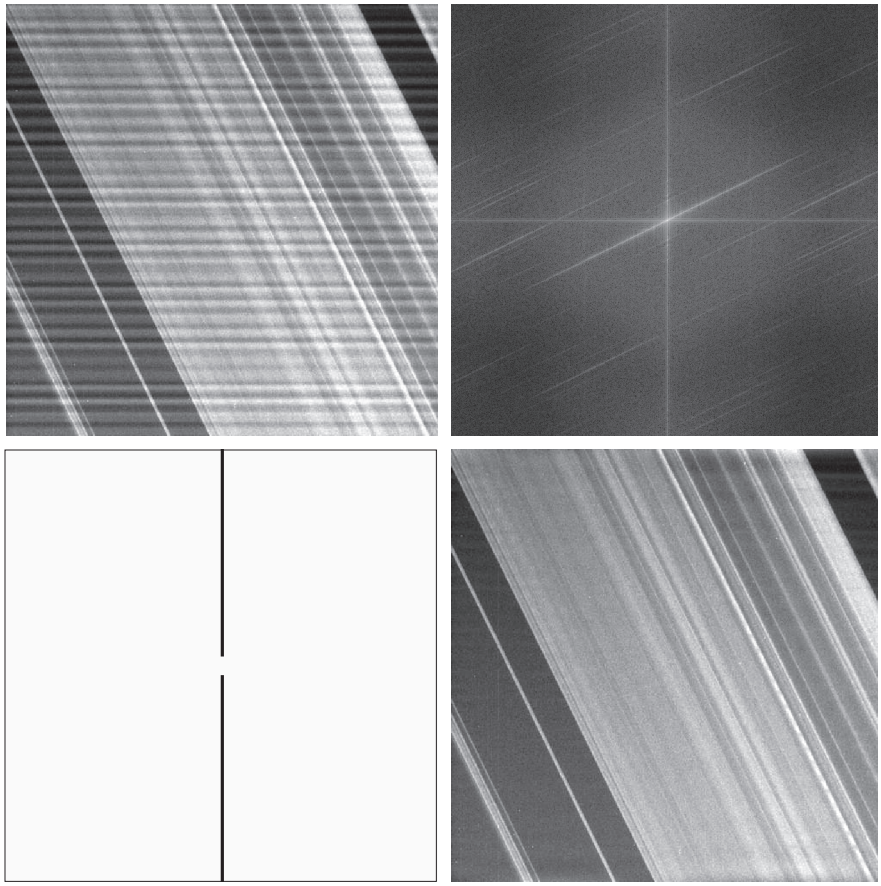
FIGURE 4.57
(a) A chest X-ray. (b) Result of filtering with a GHPF function. (c) Result of high-frequency-emphasis filtering using the same GHPF. (d) Result of performing histogram equalization on (c). (Original image courtesy of Dr. Thomas R. Gest, Division of Anatomical Sciences, University of Michigan Medical School.)

EXAMPLE 4.25: Using notch filtering to remove periodic interference.

Figure 4.65(a) shows an image of part of the rings surrounding the planet Saturn. This image was captured by *Cassini*, the first spacecraft to enter the planet's orbit. The nearly sinusoidal pattern visible in the image was caused by an AC signal superimposed on the camera video signal just prior to digitizing the image. This was an unexpected problem that corrupted some images from the mission. Fortunately, this type of interference is fairly easy to correct by postprocessing. One approach is to use notch filtering.

Figure 4.65(b) shows the DFT spectrum. Careful analysis of the vertical axis reveals a series of small bursts of energy near the origin which correspond to the nearly sinusoidal interference. A simple approach is to use a narrow notch rectangle filter starting with the lowest frequency burst, and extending for the remainder of the vertical axis. Figure 4.65(c) shows the transfer function of such a filter (white represents 1 and black 0). Figure 4.65(d) shows the result of processing the corrupted image with this filter. This result is a significant improvement over the original image.

To obtain and image of just the interference pattern, we isolated the frequencies in the vertical axis using a notch pass transfer function, obtained by subtracting the notch reject function from 1 [see Fig. 4.66(a)]. Then, as Fig. 4.66(b) shows, the IDFT of the filtered image is the spatial interference pattern.

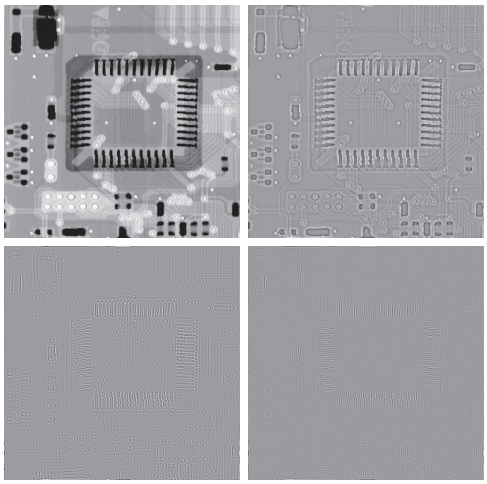


a b
c d

FIGURE 4.65
(a) Image of Saturn rings showing nearly periodic interference. (b) Spectrum. (The bursts of energy in the vertical axis near the origin correspond to the interference pattern). (c) A vertical notch reject filter transfer function. (d) Result of filtering. (The thin black border in (c) is not part of the data.) (Original image courtesy of Dr. Robert A. West, NASA/JPL.)

(b) Do you think the result would have been different if the order of the filtering process had been reversed?

4.62 Consider the sequence of images shown below. The image on the top left is a segment of an X-ray image of a commercial printed circuit board. The images following it are, respectively, the results of subjecting the image to 1, 10, and 100 passes of a Gaussian highpass filter with $D_0 = 30$. The images are of size 330×334 pixels, with each pixel being represented by 8 bits of gray. The images were scaled for display, but this has no effect on the problem statement.



(Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

(a) It appears from the images that changes will cease to take place after a finite number of passes. Show whether or not this is the case. You may ignore computational round-off errors. Let $c_{\min}$ denote the smallest positive number representable in the machine in which the computations are conducted.

(b) If you determined in (a) that changes would cease after a finite number of iterations, determine the minimum value of that number.

(*Hint:* Study the solution to Problem 4.53.)

4.63 As illustrated in Fig. 4.57, combining high-frequency emphasis and histogram equalization is an effective method for achieving edge sharpening and contrast enhancement.

(a)* Show whether or not it matters which process is applied first.

(b) If the order does matter, give a rationale for using one or the other method first.

4.64 Use a Butterworth highpass filter to construct a homomorphic filter transfer function that has the same general shape as the function in Fig. 4.59.

4.65 Suppose that you are given a set of images generated by an experiment dealing with the analysis of stellar events. Each image contains a set of bright, widely scattered dots corresponding to stars in a sparsely occupied region of the universe. The problem is that the stars are barely visible as a result of superimposed illumination from atmospheric dispersion. If these images are modeled as the product of a constant illumination component with a set of impulses, give an enhancement procedure based on homomorphic filtering designed to bring out the image components due to the stars themselves.

4.66 How would you generate an image of only the interference pattern visible in Fig. 4.64(a)?

4.67* Show the validity of Eqs. (4-171) and (4-172). (*Hint:* Use proof by induction.)

4.68 A skilled medical technician is assigned the job of inspecting a set of images generated by an electron microscope experiment. In order to simplify the inspection task, the technician decides to use digital image enhancement and, to this end, examines a set of representative images and finds the following problems: (1) bright, isolated dots that are of no interest; (2) lack of sharpness; (3) not enough contrast in some images; and (4) shifts in the average intensity to values other than A_0, which is the average value required to perform correctly certain intensity measurements. The technician wants to correct these problems and then display in white all intensities in a band between intensities I_1 and I_2, while keeping normal tonality in the remaining intensities. Propose a sequence of processing steps that the technician can follow to achieve the desired goal. You may use techniques from both Chapters 3 and 4.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

4.1 Write a function **g = minusOne4e(f)** that multiplies **f** by $(-1)^{x+y}$ to produce **g**. Array **f** can be 1-D (row or column) or 2-D. The input image must be floating point, so your function should perform a validation check for this.

4.2 Implementation and testing of the 2-D FFT and its inverse using a 1-D FFT algorithm.

(a)* Obtain a routine that computes the 1-D FFT in the language you are using for projects. For example, excellent FFT implementations in C are available from www.fftw.org. If you are working in MATLAB, use function **fft**. Use the 1-D FFT routine to implement a function **F = dft2D4e(f)** that computes the 2-D forward FFT of image **f**, as explained in Section 4.11.

(b) Write a function **f = idft2D4e(F)** that computes the inverse FFT of an input transform **F**. (*Hint:* Work with the conjugate of **F** so that you can use the forward FFT function from (a) to compute the inverse, as explained in Section 4.11.)

(c) Read the image **rose512.tif** and scale it to the range [0,1] using the default settings of function **intScaling4e**. Denote the result by **f**. Test your functions by (1) computing **F**, the forward FFT of **f**, and (2) obtaining **g**, the real part of the inverse FFT of **F**. Display **f**, **g**, and the difference, **d** = **f** − **g**, of the two. Display the maximum and minimum values of **d**. The displays of **f** and **g** should look identical, and **d** should appear as a black image.

(d)* Compute the centered transform and display the spectrum of **F** as **S** = **log(1 + abs(F))**. Scale **S** using the **'full'** option in function **intScaling4e** before displaying it.

4.3 Lowpass filter transfer functions.

(a)* Write a function **H = lpFilterTF4e(type,P,Q,param)** to generate a **P** × **Q** lowpass filter transfer function, **H**, with the following properties. If **type** = **'ideal'**, **param** should be a scalar equal to the cut-off frequency D_0 in Eq. (4-111). If **type** = **'gaussian'**, **param** should be a scalar equal to the standard deviation D_0 in Eq. (4-116). If **type** = **'butterworth'**, **param** should be a 1×2 array (vector) containing the cutoff frequency and filter order, $[D_0, n]$, in Eq. (4-117).

(b)* Generate a lowpass ideal filter transfer function of size 512×512 with $D_0 = 96$. Display your result as an image.

(c) Generate a lowpass Gaussian filter transfer function of size 512×512 with $D_0 = 96$. Display your result as an image.

(d) Generate a lowpass Butterworth filter transfer function of size 512×512. Choose $D_0 = 96$ and $n = 2$. Display your result as an image.

4.4 Highpass filter transfer functions.

(a)* Write a function **H = hpFilterTF4e(type,P,Q,param)** to generate a **P** × **Q** highpass filter transfer function, **H**, with the following properties. If **type** = **'ideal'**, **param** should be a scalar equal to the cut-off frequency D_0 in Eq. (4-119). If **type** = **'gaussian'**, **param** should be a scalar equal to the standard deviation D_0 in Eq. (4-120). If **type** = **'butterworth'**, **param** should be a 1×2 array (vector) the cutoff frequency and filter order, $[D_0, n]$, in Eq. (4-121).

(b)* Generate an ideal highpass filter transfer function of size 512×512 with $D_0 = 96$. Display your result as an image.

(c) Generate a highpass Gaussian filter transfer function of size 512×512 with $D_0 = 96$. Display your result as an image.

(d) Generate a highpass Butterworth filter transfer function of size 512×512. Choose $D_0 = 96$ and $n = 2$.

4.5 Frequency domain filtering package.

(a)* Write a function, **g = dftFiltering4e(f,H,padmode scaling)** to filter image **f** with a given filter transfer function **H**. Your function should implement the seven steps in the filtering algorithm discussed in Section 4.7. **If padmode** = **'replicate'** or is not included in the argument, then replicate padding should be used. If **padmode** = **'zeros'**, zero padding should be used.

5 Image Restoration and Reconstruction

> Things which we see are not themselves what we see . . .
> It remains completely unknown to us what the objects may be
> by themselves and apart from the receptivity of our senses.
> We know only but our manner of perceiving them.
>
> *Immanuel Kant*

Preview

As in image enhancement, the principal goal of restoration techniques is to improve an image in some predefined sense. Although there are areas of overlap, image enhancement is largely a subjective process, while image restoration is for the most part an objective process. Restoration attempts to recover an image that has been degraded by using a priori knowledge of the degradation phenomenon. Thus, restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image. In this chapter, we consider linear, space invariant restoration models that are applicable in a variety of restoration situations. We also discuss fundamental techniques of image reconstruction from projections, and their application to computed tomography (CT), one of the most important commercial applications of image processing, especially in health care.

Upon completion of this chapter, readers should:

- Be familiar with the characteristics of various noise models used in image processing, and how to estimate from image data the parameters that define those models.
- Be familiar with linear, nonlinear, and adaptive spatial filters used to restore (denoise) images that have been degraded only by noise.
- Know how to apply notch filtering in the frequency domain for removing periodic noise in an image.
- Understand the foundation of linear, space invariant system concepts, and how they can be applied in formulating image restoration solutions in the frequency domain.
- Be familiar with direct inverse filtering and its limitations.
- Understand minimum mean-square-error (Wiener) filtering and its advantages over direct inverse filtering.
- Understand constrained, least-squares filtering.
- Be familiar with the fundamentals of image reconstruction from projections, and their application to computed tomography.

5.1 A MODEL OF THE IMAGE DEGRADATION/RESTORATION PROCESS

In this chapter, we model image degradation as an operator $\mathcal{H}$ that, together with an additive noise term, operates on an input image $f(x, y)$ to produce a degraded image $g(x, y)$ (see Fig. 5.1). Given $g(x, y)$, some knowledge about $\mathcal{H}$, and some knowledge about the additive noise term $\eta(x, y)$, the objective of restoration is to obtain an estimate $\hat{f}(x, y)$ of the original image. We want the estimate to be as close as possible to the original image and, in general, the more we know about $\mathcal{H}$ and η, the closer $\hat{f}(x, y)$ will be to $f(x, y)$.

We will show in Section 5.5 that, if $\mathcal{H}$ is a linear, position-invariant operator, then the degraded image is given in the spatial domain by

$$g(x, y) = (h \star f)(x, y) + \eta(x, y) \tag{5-1}$$

where $h(x, y)$ is the spatial representation of the degradation function. As in Chapters 3 and 4, the symbol "$\star$" indicates convolution. It follows from the convolution theorem that the equivalent of Eq. (5-1) in the frequency domain is

$$G(u, v) = H(u, v)F(u, v) + N(u, v) \tag{5-2}$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in Eq. (5-1). These two equations are the foundation for most of the restoration material in this chapter.

In the following three sections, we work only with degradations caused by noise. Beginning in Section 5.5 we look at several methods for image restoration in the presence of both $\mathcal{H}$ and η.

5.2 NOISE MODELS

The principal sources of noise in digital images arise during image acquisition and/or transmission. The performance of imaging sensors is affected by a variety of environmental factors during image acquisition, and by the quality of the sensing elements themselves. For instance, in acquiring images with a CCD camera, light levels and sensor temperature are major factors affecting the amount of noise in the resulting image. Images are corrupted during transmission principally by interference in the transmission channel. For example, an image transmitted using a wireless network might be corrupted by lightning or other atmospheric disturbance.

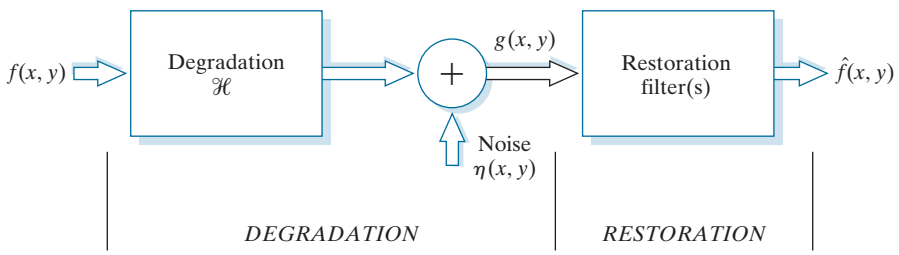
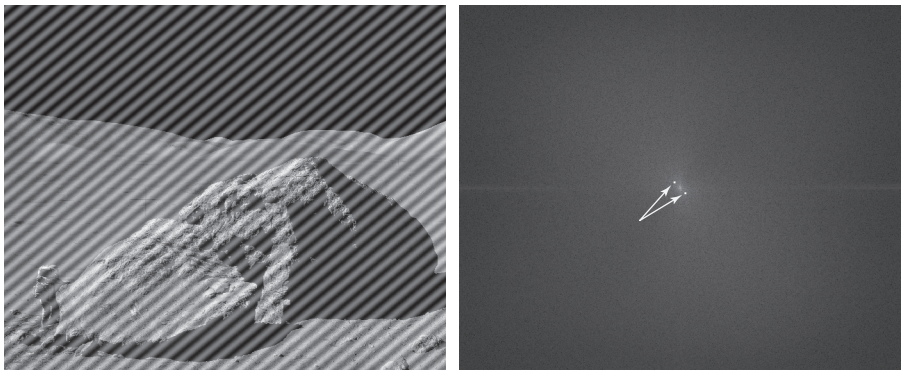


FIGURE 5.1 A model of the image degradation/restoration process.



a b

FIGURE 5.5
(a) Image corrupted by additive sinusoidal noise. (b) Spectrum showing two conjugate impulses caused by the sine wave. (Original image courtesy of NASA.)

of noise components directly from the image, but this is possible only in simplistic cases. Automated analysis is possible in situations in which the noise spikes are either exceptionally pronounced, or when knowledge is available about the general location of the frequency components of the interference (see Section 5.4).

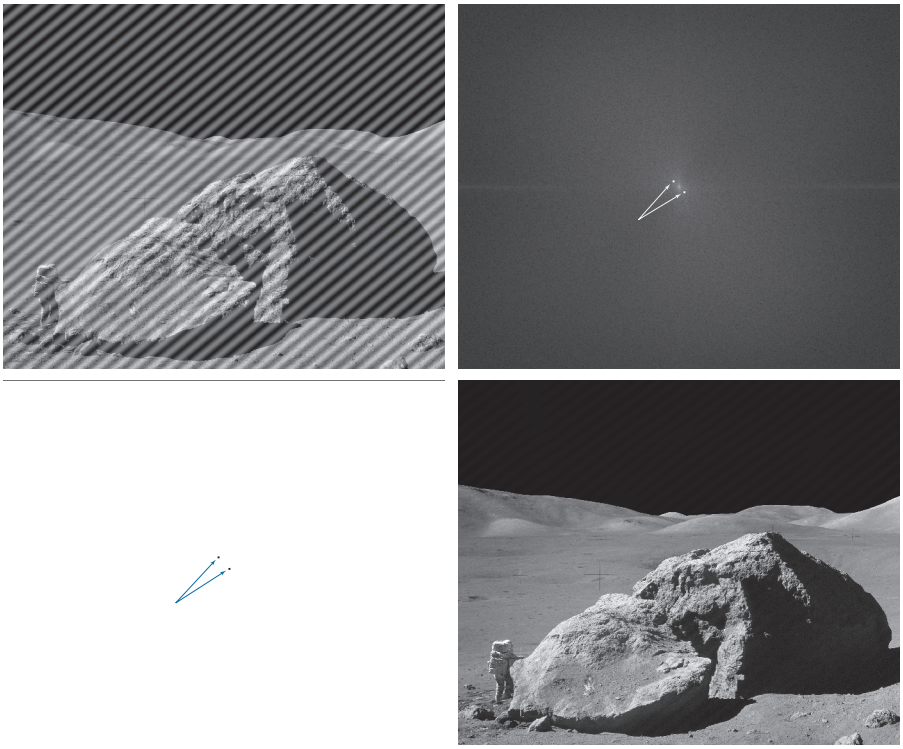
The parameters of noise PDFs may be known partially from sensor specifications, but it is often necessary to estimate them for a particular imaging arrangement. If the imaging system is available, one simple way to study the characteristics of system noise is to capture a set of "flat" images. For example, in the case of an optical sensor, this is as simple as imaging a solid gray board that is illuminated uniformly. The resulting images typically are good indicators of system noise.

When only images already generated by a sensor are available, it is often possible to estimate the parameters of the PDF from small patches of reasonably constant background intensity. For example, the vertical strips shown in Fig. 5.6 were cropped from the Gaussian, Rayleigh, and uniform images in Fig. 5.4. The histograms shown were calculated using image data from these small strips. The histograms in Fig. 5.4 that correspond to the histograms in Fig. 5.6 are the ones in the middle of the group of three in Figs. 5.4(d), (e), and (k). We see that the shapes of these histograms correspond quite closely to the shapes of the corresponding histograms in Fig. 5.6. Their heights are different due to scaling, but the shapes are unmistakably similar.

The simplest use of the data from the image strips is for calculating the mean and variance of intensity levels. Consider a strip (subimage) denoted by S, and let $p_S(z_i)$, $i = 0, 1, 2, \ldots, L-1$, denote the probability estimates (normalized histogram values) of the intensities of the pixels in S, where L is the number of possible intensities in the entire image (e.g., 256 for an 8-bit image). As in Sections 2.6 and 3.3, we estimate the mean and variance of the pixel values in S as follows:

$$\bar{z} = \sum_{i=0}^{L-1} z_i p_S(z_i) \tag{5-19}$$

and



a b
c d

FIGURE 5.16
(a) Image corrupted by sinusoidal interference. (b) Spectrum showing the bursts of energy caused by the interference. (The bursts were enlarged for display purposes.) (c) Notch filter (the radius of the circles is 2 pixels) used to eliminate the energy bursts. (The thin borders are not part of the data.) (d) Result of notch reject filtering. (Original image courtesy of NASA.)

which, as you know from Chapter 4, are responsible for the intensity differences between smooth areas. Figure 5.18(c) shows the filter transfer function we used, and Fig. 5.18(d) shows the filtered result. Most of the fine scan lines were eliminated or significantly attenuated. In order to get an image of the noise pattern, we proceed as before by converting the reject filter into a pass filter, and then filtering the input image with it. Figure 5.19 shows the result.

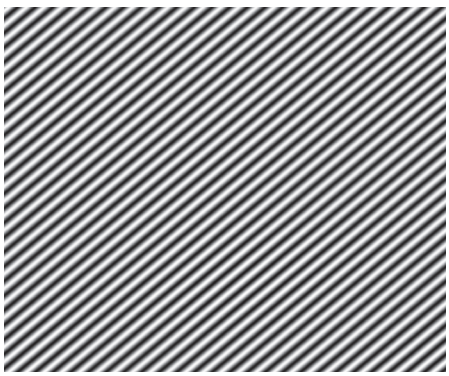
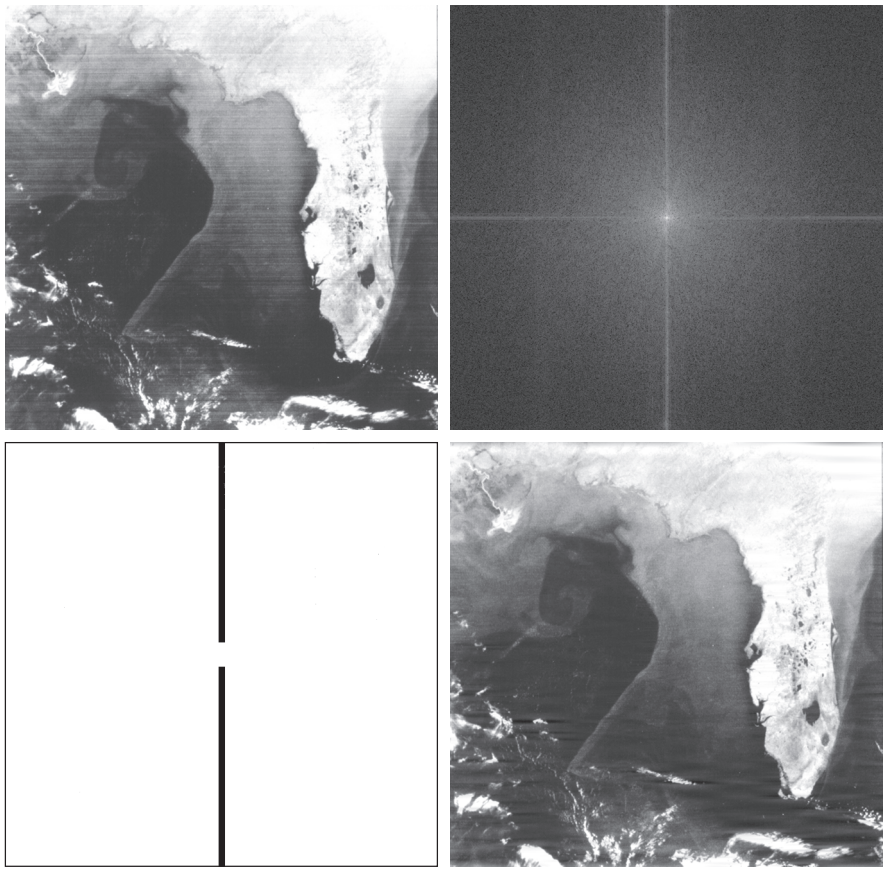


FIGURE 5.17
Sinusoidal pattern extracted from the DFT of Fig. 5.16(a) using a notch pass filter.



a b
c d

FIGURE 5.18
(a) Satellite image of Florida and the Gulf of Mexico. (Note horizontal sensor scan lines.) (b) Spectrum of (a). (c) Notch reject filter transfer function. (The thin black border is not part of the data.) (d) Filtered image. (Original image courtesy of NOAA.)

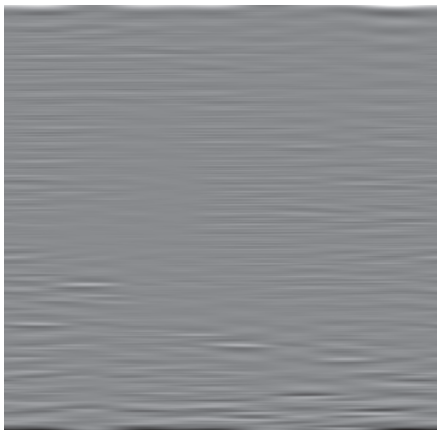
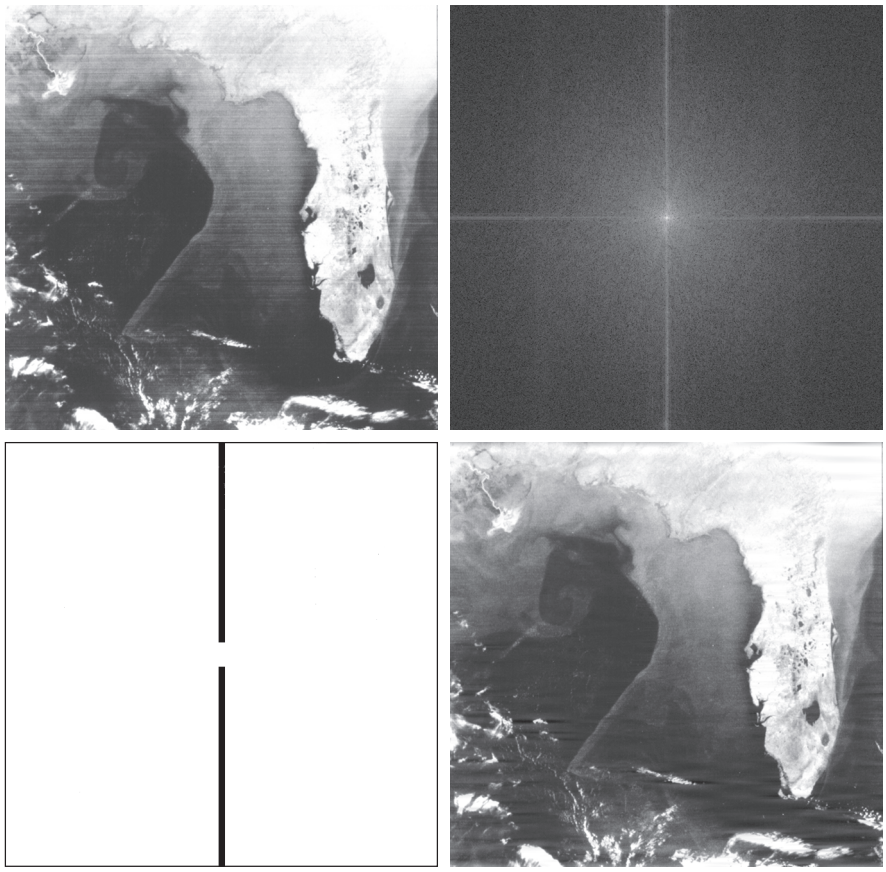


FIGURE 5.19
Noise pattern extracted from Fig. 5.18(a) by notch pass filtering.



a b
c d

FIGURE 5.18
(a) Satellite image of Florida and the Gulf of Mexico. (Note horizontal sensor scan lines.) (b) Spectrum of (a). (c) Notch reject filter transfer function. (The thin black border is not part of the data.) (d) Filtered image. (Original image courtesy of NOAA.)

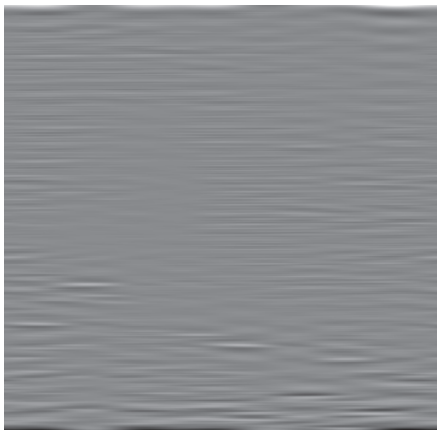
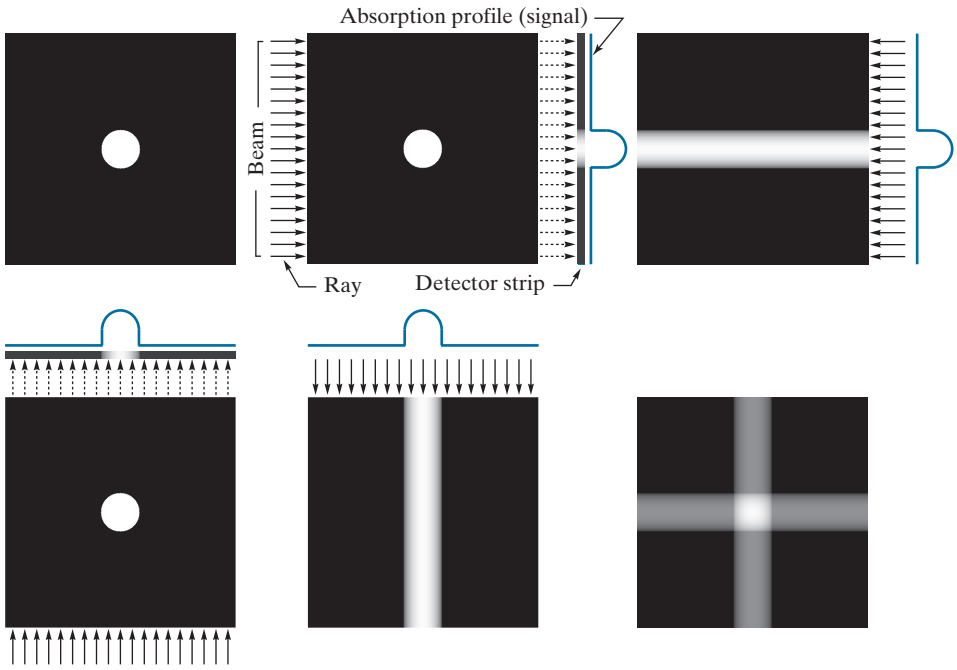


FIGURE 5.19
Noise pattern extracted from Fig. 5.18(a) by notch pass filtering.

a b c
d e f
g h i

FIGURE 5.29 (a) 8-bit image corrupted by motion blur and additive noise. (b) Result of inverse filtering. (c) Result of Wiener filtering. (d)–(f) Same sequence, but with noise variance one order of magnitude less. (g)–(i) Same sequence, but noise variance reduced by five orders of magnitude from (a). Note in (h) how the deblurred image is quite visible through a "curtain" of noise.



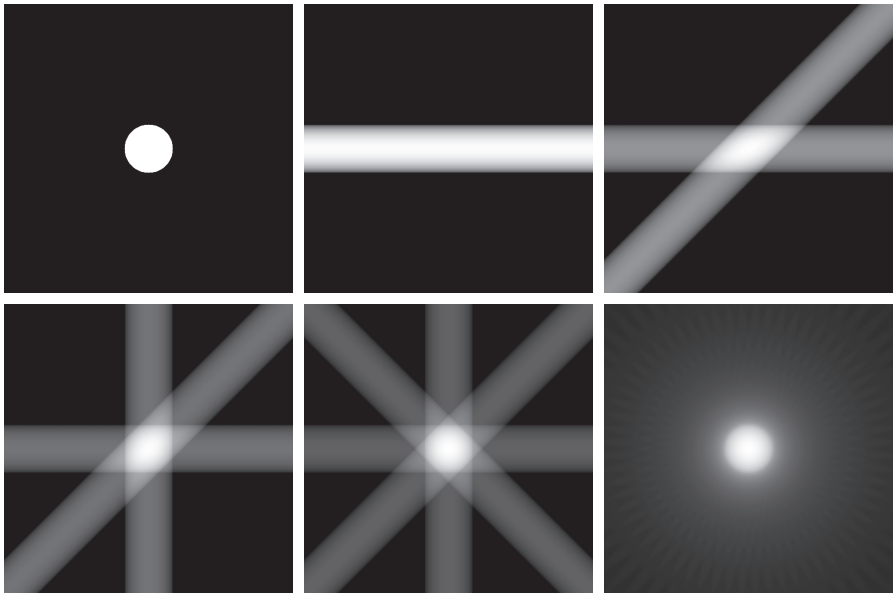
a b c
d e f

FIGURE 5.32
(a) Flat region with a single object. (b) Parallel beam, detector strip, and profile of sensed 1-D absorption signal. (c) Result of back-projecting the absorption profile. (d) Beam and detectors rotated by 90°. (e) Backprojection. (f) The sum of (c) and (e), intensity-scaled. The intensity where the backprojections intersect is twice the intensity of the individual back-projections.

terms of digital images, this means duplicating the same 1-D signal across the image, perpendicularly to the direction of the beam. For example, Fig. 5.32(c) was created by duplicating the 1-D signal in all columns of the reconstructed image. For obvious reasons, the approach just described is called *backprojection*.

Next, suppose that we rotate the position of the source-detector pair by 90°, as in Fig. 5.32(d). Repeating the procedure explained in the previous paragraph yields a backprojection image in the vertical direction, as Fig. 5.32(e) shows. We continue the reconstruction by *adding* this result to the previous backprojection, resulting in Fig. 5.32(f). Now, we begin to suspect that the object of interest is contained in the square shown, whose amplitude is twice the amplitude of the individual backprojections because the signals were added. We should be able to learn more about the shape of the object in question by taking more views in the manner just described, as Fig. 5.33 shows. As the number of projections increases, the amplitude strength of non-intersecting backprojections decreases relative to the strength of regions in which multiple backprojections intersect. The net effect is that brighter regions will dominate the result, and backprojections with few or no intersections will fade into the background as the image is scaled for display.

Figure 5.33(f), which was formed from 32 backprojections, illustrates this concept. Note, however, that while this reconstructed image is a reasonably good approximation to the shape of the original object, the image is blurred by a "halo" effect, the formation of which can be seen in progressive stages in Fig. 5.33. For example, the halo in Fig. 5.33(e) appears as a "star" whose intensity is lower than that of the



a b c
d e f

FIGURE 5.33
(a) Same as Fig. 5.32(a). (b)-(e) Reconstruction using 1, 2, 3, and 4 backprojections 45° apart. (f) Reconstruction with 32 backprojections 5.625° apart (note the blurring).

object, but higher than the background. As the number of views increases, the shape of the halo becomes circular, as in Fig. 5.33(f). Blurring in CT reconstruction is an important issue, whose solution is addressed later in this section. Finally, we conclude from the discussion of Figs. 5.32 and 5.33 that backprojections 180° apart are mirror images of each other, so we have to consider only angle increments halfway around a circle in order to generate all the backprojections required for reconstruction.

EXAMPLE 5.14: Backprojections of a planar region containing two objects.

Figure 5.34 illustrates reconstruction using backprojections on a region that contains two objects with different absorption properties (the larger object has higher absorption). Figure 5.34(b) shows the result of using one backprojection. We note three principal features in this figure, from bottom to top: a thin horizontal gray band corresponding to the unoccluded portion of the small object, a brighter (more absorption) band above it corresponding to the area shared by both objects, and an upper band corresponding to the rest of the elliptical object. Figures 5.34(c) and (d) show reconstruction using two projections 90° apart and four projections 45° apart, respectively. The explanation of these figures is similar to the discussion of Figs. 5.33(c) through (e). Figures 5.34(e) and (f) show more accurate reconstructions using 32 and 64 backprojections, respectively. The last two results are quite close visually, and they both show the blurring problem mentioned earlier.

PRINCIPLES OF X-RAY COMPUTED TOMOGRAPHY (CT)

As with the Fourier transform discussed in the last chapter, the basic mathematical concepts required for CT were in place many years before the availability of digital

5.46 Show that the Radon transform [Eq. (5-102)] of the Gaussian shape $f(x,y) = A\exp(-x^2 - y^2)$ is given by $g(\rho,\theta) = A\sqrt{\pi}\exp(-\rho^2)$. (*Hint:* Refer to Example 5.15, where we used symmetry to simplify integration.)

5.47 Do the following:

(a)* Show that the Radon transform [Eq. (5-102)] of the unit impulse $\delta(x,y)$ is a straight vertical line passing through the origin of the $\rho\theta$-plane .

(b) Show that the radon transform of the impulse $\delta(x - x_0, y - y_0)$ is a sinusoidal curve in the $\rho\theta$-plane.

5.48 Prove the validity of the following properties of the Radon transform [Eq. (5-102)]:

(a)* *Linearity:* The Radon transform is a linear operator. (See Section 2.6 regarding linearity.)

(b) *Translation property:* The radon transform of $f(x - x_0, y - y_0)$ is $g(\rho - x_0\cos\theta - y_0\sin\theta, \theta)$.

(c)* *Convolution property:* The Radon transform of the convolution of two functions is equal to the convolution of the Radon transforms of the two functions.

5.49 Provide the steps that lead from Eq. (5-113) to Eq. (5-114). [*Hint:* $G(\omega,\theta + 180°) = G(-\omega,\theta)$.]

5.50* Prove the validity of Eq. (5-125).

5.51 Prove the validity of Eq. (5-127).

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

5.1 Read the image **book-cover-gaussian.tif**. You are told that this image has been corrupted by additive Gaussian noise. Find estimates of the mean and standard deviation of the noise. (*Hint:* Take a look at Fig. 5.6 and review project function **centralMoments4e**.)

5.2 Obtain functions for generating 2-D arrays of uniform and Gaussian random numbers in the language you are using for your projects. If you are using MATLAB, functions **rand** and **randn** are the noise generators of choice for this purpose. Read the image **testpattern512.tif**, scale it to the range [0, 1] using the default mode of project function **intScaling4e**, and do the following.

(a)* Fix the mean at 0.25 and add three levels of Gaussian noise to the image by varying the standard deviation. The three levels should be such that the noise appears (1) *mild* (you can tell the noise is there, but it is barely perceivable; (2) *intermediate* (the noise is definitely present, but all the image features are still clearly visible); and (3) *heavy* (the noise is objectionable, causing some of the image features to be obscured by the noise); (4) *extra heavy* (the noise dominates the image; most of the smaller and light features in the image are obscured by noise). For comparisons of your results to be meaningful, you should scale the image to the full range [0, 1], using the **'full'** and **'floating'** options in function **intScaling4e**. Show all four results and list the values of standard deviation you used. Explain why image details begin to disappear as the noise level increases significantly.

(b) Repeat the four levels of noise outlined in (a) using uniform noise. Note that the parameters to specify are a and b in Eq. (5-13). Try to make your images appear as close as possible to their Gaussian counterparts. Explain why image details begin to disappear as the noise level increases significantly. Note any significant differences between corresponding images in (a) and (b).

(c) You will find that the images in (b) have higher contrast than their Gaussian counterparts in (a). Explain why.

5.3 Working with salt-and-pepper noise.

(a)* Explain how you can modify a generator of uniform random numbers to produce salt-and-pepper noise with specified probabilities P_s for salt pixels and P_p for pepper pixels.

6 Wavelet and Other Image Transforms

Do not conform any longer to the pattern of this world, but be transformed by the renewing of your mind.

Romans 12:2

Preview

The discrete Fourier transform of Chapter 4 is a member of an important class of linear transforms that include the Hartley, sine, cosine, Walsh-Hadamard, Slant, Haar, and wavelet transforms. These transforms, which are the subject of this chapter, decompose functions into weighted sums of orthogonal or biorthogonal basis functions, and can be studied using the tools of linear algebra and functional analysis. When approached from this point of view, images are vectors in the vector space of all images. Basis functions determine the nature and usefulness of image transforms. Transforms are the coefficients of linear expansions. And for a given image and transform (or set of basis functions), both the orthogonality of the basis functions and the coefficients of the resulting transform are computed using inner products. All of an image's transforms are equivalent in the sense that they contain the same information and total energy. They are reversible and differ only in the way that the information and energy is distributed among the transform's coefficients.

Upon competion of this chapter, readers should:

- Understand image transforms in the context of series expansions.
- Be familiar with a variety of important image transforms and transform basis functions.
- Know the difference between orthogonal and biorthogonal basis functions.
- Be able to construct the transformation matrices of the discrete Fourier, Hartley, sine, cosine, Walsh-Hadamard, Slant, and Haar transforms.
- Be able to compute traditional image transforms, like the Fourier and Haar transforms, using elementary matrix operations.
- Understand the time-frequency plane and its relationship to wavelet transforms.
- Be able to compute 1-D and 2-D fast wavelet transforms (FWTs) using filter banks.
- Understand wavelet packet representations.
- Be familiar with the use of discrete orthogonal transforms in image processing.

6.1 PRELIMINARIES

Consult the Tutorials section of the book website for a brief tutorial on vectors and matrices.

In linear algebra and functional analysis, a *vector space* (or more formally an *abstract vector space*) is a set of mathematical objects or entities, called *vectors*, that can be added together and multiplied by *scalars*. An *inner product space* is an abstract vector space over a field of numbers, together with an *inner product function* that maps two vectors of the vector space to a scalar of the number field such that

In Chapter 2, the inner product of two column vectors, $\mathbf{u}$ and $\mathbf{v}$, is denoted $\mathbf{u} \cdot \mathbf{v}$ [see Eq. (2-50)]. In this chapter, $\langle \mathbf{u}, \mathbf{v} \rangle$ is used to denote inner products within any inner product space satisfying conditions (a)–(d), including the Euclidean inner product space and real-valued column vectors of Chapter 2.

(a) $\langle u, v \rangle = \langle v, u \rangle^*$
(b) $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$
(c) $\langle \alpha u, v \rangle = \alpha \langle u, v \rangle$
(d) $\langle v, v \rangle \geq 0$ and $\langle v, v \rangle = 0$ if and only if $v = 0$

where u, v, and w are vectors, α is a scalar, and $\langle \ldots \rangle$ denotes the inner product operation. A simple example of a vector space is the set of directed line segments in two dimensions, where the line segments are represented mathematically as 2×1 column vectors, and the addition of vectors is the arithmetic equivalent of combining the line segments in a head to tail manner. An example of an inner product space is the set of real numbers $\mathbf{R}$ combined with inner product function $\langle u, v \rangle = uv$, where the "vectors" are real numbers, the inner product function is multiplication, and axioms (a) through (d) above correspond to the commutative, distributive, associative, and "positivity of even powers" properties of multiplication, respectively.

Three inner product spaces are of particular interest in this chapter:

Euclidean space $\mathbf{R}^N$ is an infinite set containing all real N-tuples.

1. *Euclidean* space $\mathbf{R}^N$ over real number field $\mathbf{R}$ with *dot* or *scalar* inner product

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^T \mathbf{v} = u_0 v_0 + u_1 v_1 + \ldots + u_{N-1} v_{N-1} = \sum_{i=0}^{N-1} u_i v_i \tag{6-1}$$

where $\mathbf{u}$ and $\mathbf{v}$ are $N \times 1$ column vectors.

A complex vector space with an inner product is called a *complex inner product space* or *unitary* space.

2. *Unitary* space $\mathbf{C}^N$ over complex number field $\mathbf{C}$ with inner product function

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^{*T} \mathbf{v} = \sum_{i=0}^{N-1} u_i^* v_i = \langle \mathbf{v}, \mathbf{u} \rangle^* \tag{6-2}$$

where * denotes the complex conjugate operation, and $\mathbf{u}$ and $\mathbf{v}$ are complex-valued $N \times 1$ column vectors.

The notation $C[a, b]$ is also used in the literature.

3. Inner product space $C([a, b])$, where the vectors are continuous functions on the interval $a \leq x \leq b$ and the inner product function is the *integral inner product*

$$\langle f(x), g(x) \rangle = \int_a^b f^*(x) g(x) dx \tag{6-3}$$

Equations (6-4) through (6-15) are valid for all inner product spaces, including those defined by Eqs. (6-1) to (6-3).

In all three inner product spaces, the *norm* or *length* of vector z, denoted as $\|z\|$, is

$$\|z\| = \sqrt{\langle z, z \rangle} \tag{6-4}$$

and the *angle* between two nonzero vectors z and w is

$$\theta = \cos^{-1}\frac{\langle z, w\rangle}{\|z\|\|w\|} \tag{6-5}$$

If the norm of z is 1, z is said to be *normalized*. If $\langle z, w\rangle = 0$ in Eq. (6-5), $\theta = 90°$ and z and w are said to be *orthogonal*. A natural consequence of these definitions is that a set of nonzero vectors $w_0, w_1, w_2, \ldots$ is mutually or pairwise orthogonal if and only if

$$\langle w_k, w_l\rangle = 0 \quad \text{for } k \neq l \tag{6-6}$$

While you must always take the context into account, we generally use the word "vector" for vectors in an abstract sense. A vector can be an $N \times 1$ matrix (i.e., column vector) or a continuous function.

Recall from linear algebra that a *basis* of a vector space is a set of linearly independent vectors for which any vector in the space can be written uniquely as a linear combination of basis vectors. The linear combinations are the *span* of the basis vectors. A set of vectors is *linearly independent* if no vector in the set can be written as a linear combination of the others.

They are an *orthogonal basis* of the inner product space that they are said to *span*. If the *basis vectors* are normalized, they are an *orthonormal basis* and

$$\langle w_k, w_l\rangle = \delta_{kl} = \begin{cases} 0 & \text{for } k \neq l \\ 1 & \text{for } k = l \end{cases} \tag{6-7}$$

Similarly, a set of vectors w_0, w_1, w_2, ... and a complementary set of *dual vectors* $\tilde{w}_0, \tilde{w}_1, \tilde{w}_2, \ldots$ are said to be *biorthogonal* and a *biorthogonal basis* of the vector space that they span if

$$\langle \tilde{w}_k, w_l\rangle = 0 \quad \text{for } k \neq l \tag{6-8}$$

They are a *biorthonormal basis* if and only if

$$\langle \tilde{w}_k, w_l\rangle = \delta_{kl} = \begin{cases} 0 & \text{for } k \neq l \\ 1 & \text{for } k = l \end{cases} \tag{6-9}$$

As a mechanism for concisely describing an infinite set of vectors, the basis of an inner product space is one of the most useful concepts in linear algebra. The following derivation, which relies on the orthogonality of basis vectors, is foundational to the matrix-based transforms of the next section. Let $W = \{w_0, w_1, w_2, \ldots\}$ be an orthogonal basis of inner product space V, and let $z \in V$. Vector z can then be expressed as the following linear combination of basis vectors

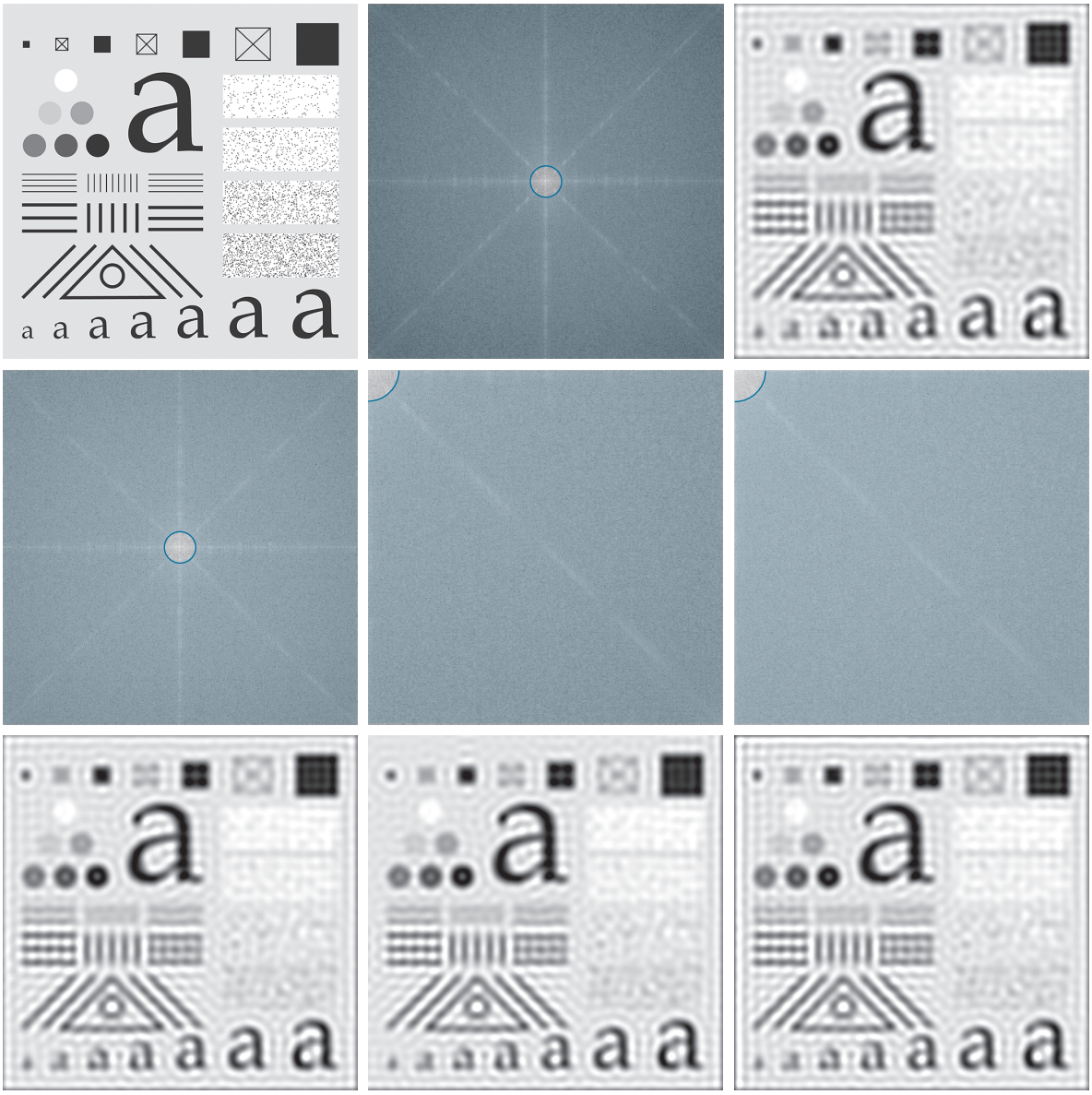
$$z = \alpha_0 w_0 + \alpha_1 w_1 + \alpha_2 w_2 + \ldots \tag{6-10}$$

While you must always take to the context into account, we often use the phrase "orthogonal basis" or "orthogonal transform" to refer to any basis or transform that is orthogonal, orthonormal, biorthogonal, or biorthonormal.

whose inner product with basis vector w_i is

$$\begin{aligned} \langle w_i, z\rangle &= \langle w_i, \alpha_0 w_0 + \alpha_1 w_1 + \alpha_2 w_2 + \ldots\rangle \\ &= \alpha_0 \langle w_i, w_0\rangle + \alpha_1 \langle w_i, w_1\rangle + \ldots + \alpha_i \langle w_i, w_i\rangle + \ldots \end{aligned} \tag{6-11}$$

Since the w_i are mutually orthogonal, the inner products on the right side of Eq. (6-11) are 0 unless the subscripts of the vectors whose inner products are being



a b c
d e f
g h i

FIGURE 6.15 (a) Original image of the 688×688 test pattern from Fig. 4.41(a). (b) Discrete Fourier transform (DFT) of the test pattern in (a) after padding to size 1376×1376. The blue overlay is an ideal lowpass filter (ILPF) with a radius of 60. (c) Result of Fourier filtering. (d)–(f) Discrete Hartley transform, discrete cosine transform (DCT), and discrete sine transform (DST) of the test pattern in (a) after padding. The blue overlay is the same ILPF in (b), but appears bigger in (e) and (f) because of the higher frequency resolution of the DCT and DST. (g)–(i) Results of filtering for the Hartley, cosine, and sine transforms, respectively.

from its nearest neighboring approximations, and complementary functions, called *wavelets*, are used to encode the differences between adjacent approximations. The *discrete wavelet transform* (DWT) uses those wavelets, together with a single scaling function, to represent a function or image as a linear combination of the wavelets and scaling function. Thus, the wavelets and scaling function serve as an othonormal or biorthonormal basis of the DWT expansion. The Daubechies and Biorthogonal B-splines of Figs. 6.3(f) and (g) and the Haar basis functions of the previous section are but three of the many bases that can be used in DWTs.

The discrete wavelet transform, like all transforms considered in this chapter, generates linear expansions of functions with respect to sets of orthonormal or biorthonormal expansion functions.

In this section, we present a mathematical framework for the interpretation and application of discrete wavelet transforms. We use the discrete wavelet transform with respect to Haar basis functions to illustrate the concepts introduced. As you proceed through the material, remember that the discrete wavelet transform of a function with respect to Haar basis functions is *not* the Haar transform of the function (although the two are intimately related).

The coefficients of a 1-D full-scale DWT with respect to Haar wavelets and a 1-D Haar transform are the same.

SCALING FUNCTIONS

Consider the set of basis functions composed of all integer translations and binary scalings of the real, square-integrable *father scaling function* $\varphi(x)$—that is, the set of scaled and translated functions $\{\varphi_{j,k}(x) \mid j,k \in \mathbf{Z}\}$ where

$\mathbf{Z}$ is the set of integers.

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k) \tag{6-121}$$

In this equation, integer *translation* k determines the position of $\varphi_{j,k}(x)$ along the x-axis and *scale* j determines its shape—i.e., its width and amplitude. If we restrict j to some value, say $j = j_0$, then $\{\varphi_{j_0,k} \mid k \in \mathbf{Z}\}$ is the basis of the function space spanned by the $\varphi_{j,k}(x)$ for $j = j_0$ and $k = \ldots, -1, 0, 1, 2, \ldots$, denoted V_{j_0}. Increasing j_0 increases the number of representable functions in V_{j_0}, allowing functions with smaller variations and finer detail to be included in the space. As is demonstrated in Fig. 6.19 with Haar scaling functions, this is a consequence of the fact that as j_0 increases, the scaling functions used to represent the functions in V_{j_0} become narrower and separated by smaller changes in x.

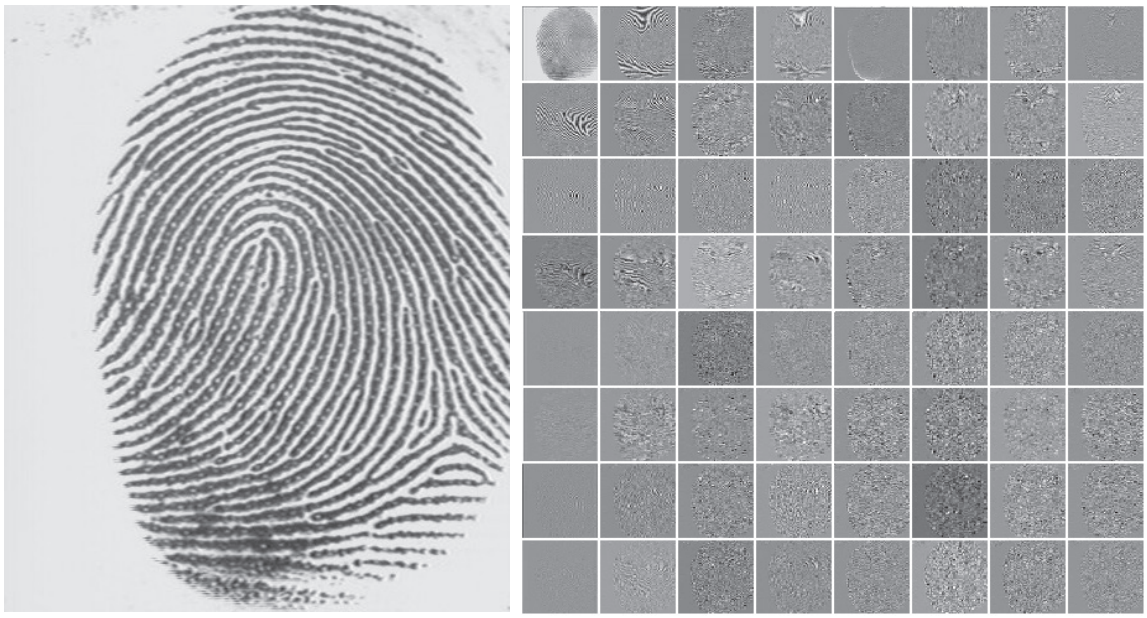
Recall from Section 6.1 that the span of a basis is the set of functions that can be represented as linear combinations of the basis functions.

EXAMPLE 6.15: The Haar scaling function.

Consider the unit-height, unit-width scaling function

$$\varphi(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{6-122}$$

and note it is the Haar basis function $h_0(x)$ from Eq. (6-115). Figure 6.19 shows a few of the pulse-shaped scaling functions that can be generated by substituting Eq. (6-122) into Eq. (6-121). Note when the scale is 1 [i.e., when $j = 1$ as in Figs. 6.19(d) and (e)], the scaling functions are half as wide as when the scale is 0 (i.e., when $j = 0$ as in Figs. 6.19(a) and (b)]. Moreover, for a given interval on x, there are



a b

FIGURE 6.39 (a) A scanned fingerprint and (b) its three-scale, full wavelet packet decomposition. Although the 64 subimages of the packet decomposition appear to be square (e.g., note the approximation subimage), this is merely an aberration of the program used to produce the result. (Original image courtesy of the National Institute of Standards and Technology.)

The cost function just described is both computationally simple and easily adapted to tree optimization routines. The optimization algorithm must use the function to minimize the "cost" of the leaf nodes in the decomposition tree. Minimal energy leaf nodes should be favored because they have more near-zero values, which leads to greater compression. Because the cost function of Eq. (6-163) is a local measure that uses only the information available at the node under consideration, an efficient algorithm for finding minimal energy solutions is easily constructed as follows:

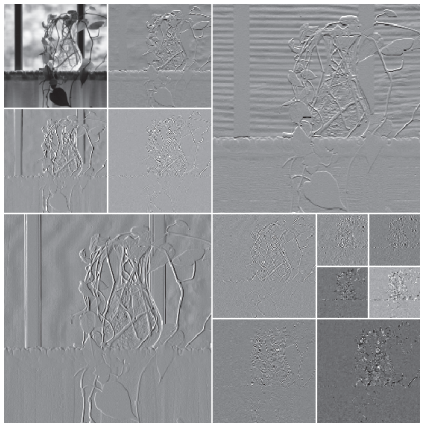
For each node of the analysis tree, beginning with the root and proceeding level by level to the leaves:

1. Compute both the energy of the node, denoted E_P (for parent energy), and the energy of its four offspring—denoted as E_A, E_H, E_V, and E_D. For two-dimensional wavelet packet decompositions, the parent is a two-dimensional array of approximation or detail coefficients; the offspring are the filtered approximation, horizontal, vertical, and diagonal details.
2. If the combined energy of the offspring is less than the energy of the parent (that is, $E_A + E_H + E_V + E_D < E_P$), include the offspring in the analysis tree. If the combined energy of the offspring is greater than or equal to that of the parent, prune the offspring, keeping only the parent. It is a leaf of the optimized analysis tree.

The preceding algorithm can be used to (1) prune wavelet packet trees or (2) design procedures for computing optimal trees from scratch. In the latter case, nonessential siblings—descendants of nodes that

ysis tree, labeling all nodes with the names of the proper scaling and wavelet spaces.

(b) Draw and label the decomposition's frequency spectrum.



6.48 Using the Haar wavelet, determine the minimum entropy packet decomposition for the function for $f(x) = 0.25$ for $n = 0, 1, \ldots, 15$. Employ the nonnormalized Shannon entropy

$$E\left[f(x)\right] = \sum_{x} f^2(x) \ln\left[f^2(x)\right]$$

as the minimization criterion. Draw the optimal tree, labeling the nodes with the computed entropy values.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

6.1* Write a function **a = tmat4e(xform,n)** to generate transformation matrices of size **n x n**. Input string **xform** should select between the Fourier, Hartley, cosine, and sine transforms. Test your function by generating the transformation matrices of size 8×8 for each of the supported transforms and comparing the results to those in Figs. 6.7, 6.8, 6.10, and 6.13.

6.2 Write a pair of functions **t = transform4e(f,xform)** and **f = invTransform4e(t,xform)** to compute the forward and inverse 1- and 2-D transforms of discrete function **f**. If input **f** is two dimensional, it can assumed to be a square matrix for simplicity. Input **xform** should select a transformation in accordance with **tmat4e** of Problem 6.1. If input **f** is a row or column vector, compute a 1-D transform; if **f** is a matrix, compute a 2-D transform.

(a) Use **transform4e** to check your answers to Problem 6.29.

(b) Use **invTransform4e** to verify that all four transforms are reversible.

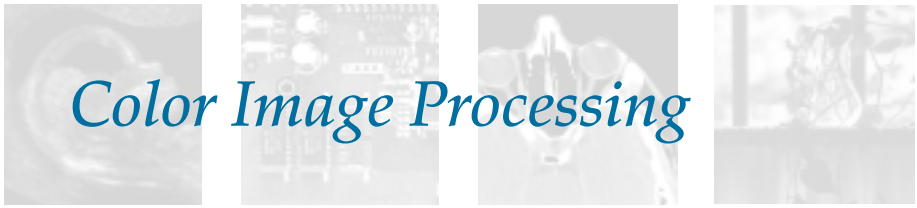
6.3 Write a function **thp = idealFilter4e(f,xform,type,r)** to filter 2-D input function **f** using an ideal highpass or lowpass filter with a cutoff frequency of radius **r**. As in Projects 6.1 and 6.2, input **xform** should select the transform employed. Use input **type** to specify either highpass or lowpass filtering.

(a) Download the image **characterTestPattern688.tif** and use your function to duplicate the results of Example 6.12.

(b) Use function **idealFilter4e** to highpass filter the downloaded image from (a) using the supported transforms. Use a cutoff frequency of radius 60 and compare your results to that of Fig. 4.53(a).

6.4 Write a function **i = basisImage4e(xform,n)** to generate and display the basis images of 2-D transform **xform**. Input **xform** should select between the transforms supported by **tmat4e** from Project 6.1. Organize and display the **n x n** basis images in an **n x n** array as demonstrated in Fig. 6.6(a). Use your function to generate and display the basis images of the Fourier, Hartley, sine, and cosine transforms of size 4×4 (i.e, with **n** set to 4). (*Note*: You should display the real and imaginary parts of the Fourier basis images separately.)

7 Color Image Processing



> It is only after years of preparation that the young artist should touch color—not color used descriptively, that is, but as a means of personal expression.
>
> *Henri Matisse*
>
> For a long time I limited myself to one color—as a form of discipline.
>
> *Pablo Picasso*

Preview

Using color in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of color shades, compared to only about two dozen shades of gray. The latter factor is particularly important in manual image analysis. Color image processing is divided into two major areas: *pseudo-* and *full-color* processing. In the first category, the issue is one of assigning color(s) to a particular grayscale intensity or range of intensities. In the second, images typically are acquired using a full-color sensor, such as a digital camera, or color scanner. Until just a few years ago, most digital color image processing was done at the pseudo- or reduced-color level. However, because color sensors and processing hardware have become available at reasonable prices, full-color image processing techniques are now used in a broad range of applications. In the discussions that follow, it will become evident that some of the grayscale methods covered in previous chapters are applicable also to color images.

Upon completion of this chapter, readers should:

- Understand the fundamentals of color and the color spectrum.
- Be familiar with several of the color models used in digital image processing.
- Know how to apply basic techniques in pseudo-color image processing, including intensity slicing and intensity-to-color transformations.
- Be familiar with how to determine if a grayscale method is extendible to color images.
- Understand the basics of working with full-color images, including color transformations, color complements, and tone/color corrections.
- Be familiar with the role of noise in color image processing.
- Know how to perform spatial filtering on color images.
- Understand the advantages of using color in image segmentation.

7.1 COLOR FUNDAMENTALS

Although the process employed by the human brain in perceiving and interpreting color is a physiopsychological phenomenon that is not fully understood, the physical nature of color can be expressed on a formal basis supported by experimental and theoretical results.

In 1666, Sir Isaac Newton discovered that when a beam of sunlight passes through a glass prism, the emerging light is not white, but consists instead of a continuous spectrum of colors ranging from violet at one end to red at the other. As Fig. 7.1 shows, the color spectrum may be divided into six broad regions: violet, blue, green, yellow, orange, and red. When viewed in full color (see Fig. 7.2), no color in the spectrum ends abruptly; rather, each color blends smoothly into the next.

Basically, the colors that humans and some other animals perceive in an object are determined by the nature of the light reflected from the object. As illustrated in Fig. 7.2, visible light is composed of a relatively narrow band of frequencies in the electromagnetic spectrum. A body that reflects light that is balanced in all visible wavelengths appears white to the observer. However, a body that favors reflectance in a limited range of the visible spectrum exhibits some shades of color. For example, green objects reflect light with wavelengths primarily in the 500 to 570 nm range, while absorbing most of the energy at other wavelengths.

Characterization of light is central to the science of color. If the light is *achromatic* (void of color), its only attribute is its *intensity*, or amount. Achromatic light is what you see on movie films made before the 1930s. As defined in Chapter 2, and used numerous times since, the term *gray* (or *intensity*) *level* refers to a scalar measure of intensity that ranges from black, to grays, and finally to white.

Chromatic light spans the electromagnetic spectrum from approximately 400 to 700 nm. Three basic quantities used to describe the quality of a chromatic light source are: radiance, luminance, and brightness. *Radiance* is the total amount of energy that flows from the light source, and it is usually measured in watts (W). *Luminance*, measured in lumens (lm), is a measure of the amount of energy that an observer *perceives* from a light source. For example, light emitted from a source operating in the far infrared region of the spectrum could have significant energy (radiance), but an observer would hardly perceive it; its luminance would be almost zero. Finally, *brightness* is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity, and is one of the key factors in describing color sensation.

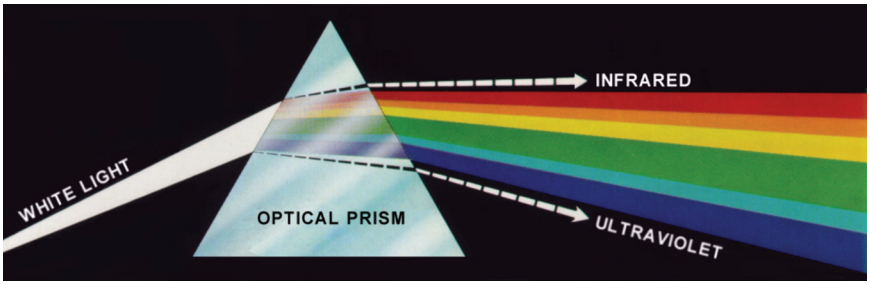


FIGURE 7.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lighting Division.)

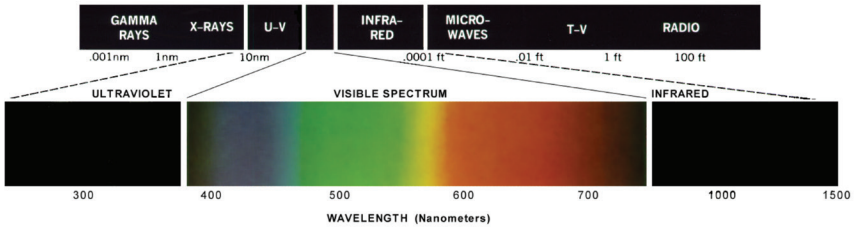


FIGURE 7.2 Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lighting Division.)

As noted in Section 2.1, cones are the sensors in the eye responsible for color vision. Detailed experimental evidence has established that the 6 to 7 million cones in the human eye can be divided into three principal sensing categories, corresponding roughly to red, green, and blue. Approximately 65% of all cones are sensitive to red light, 33% are sensitive to green light, and only about 2% are sensitive to blue. However, the blue cones are the most sensitive. Figure 7.3 shows average experimental curves detailing the absorption of light by the red, green, and blue cones in the eye. Because of these absorption characteristics, the human eye sees colors as variable combinations of the so-called *primary colors*: *red* (R), *green* (G), and *blue* (B).

For the purpose of standardization, the CIE (Commission Internationale de l'Eclairage—the International Commission on Illumination) designated in 1931 the following specific wavelength values to the three primary colors: blue = 435.8 nm, green = 546.1 nm, and red = 700 nm. This standard was set before results such as those in Fig. 7.3 became available in 1965. Thus, the CIE standards correspond only approximately with experimental data. It is important to keep in mind that defining three specific primary color wavelengths for the purpose of standardization does

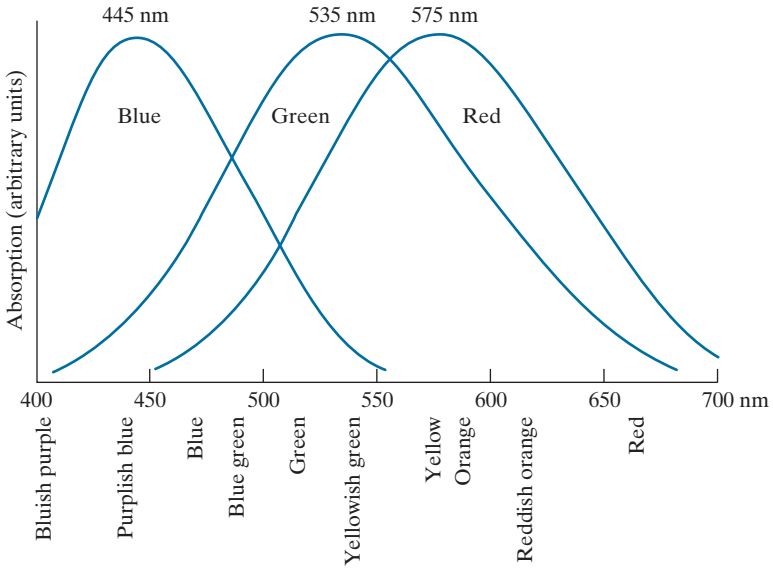


FIGURE 7.3 Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.

FIGURE 7.8
A 24-bit RGB color cube.

numbers representable by the number bits in the images. If, as above, the primary images are 8-bit images, the limits of the cube along each axis becomes [0, 255]. Then, for example, white would be at point [255, 255, 255] in the cube.

EXAMPLE 7.1: Generating a cross-section of the RGB color cube and its thee hidden planes.

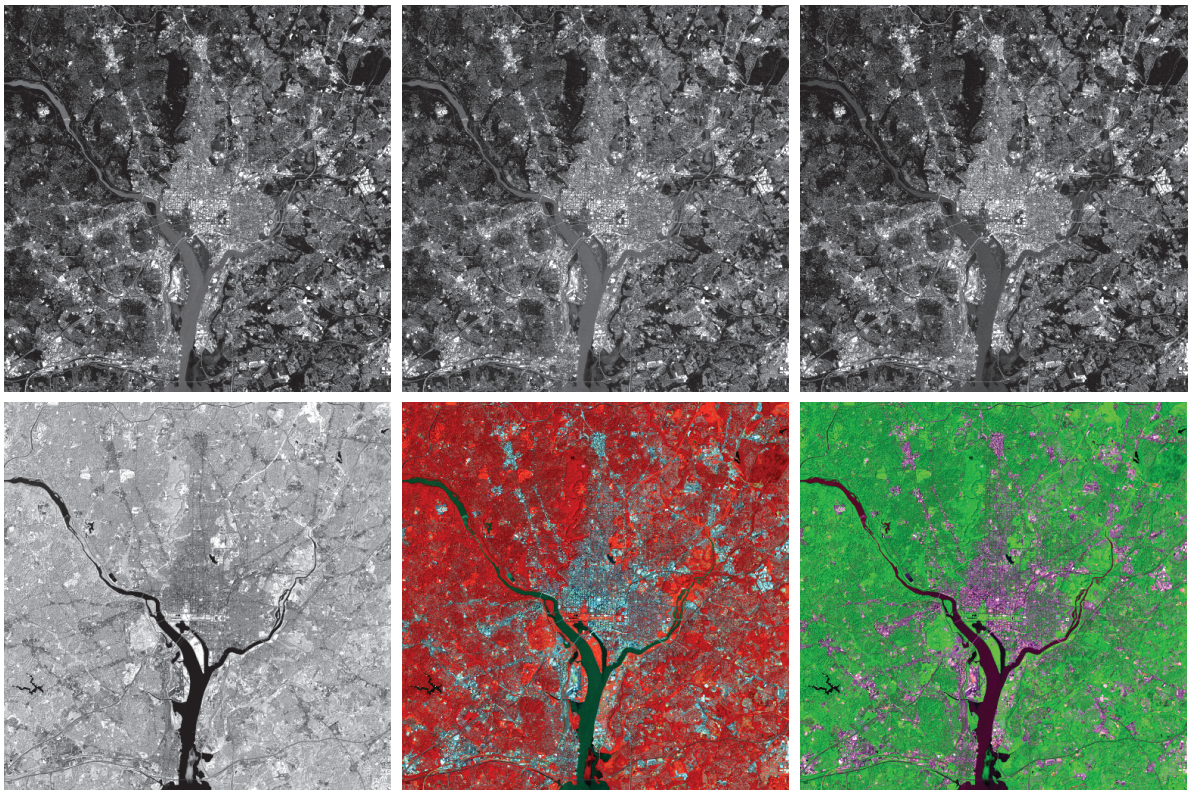
The cube in Fig. 7.8 is a solid, composed of the $(2^8)^3$ colors mentioned in the preceding paragraph. A useful way to view these colors is to generate color planes (faces or cross sections of the cube). This is done by fixing one of the three colors and allowing the other two to vary. For instance, a cross-sectional plane through the center of the cube and parallel to the GB-plane in Fig. 7.8 is the plane (127, G, B) for $G, B = 0, 1, 2, \ldots, 255$. Figure 7.9(a) shows that an image of this cross-sectional plane is generated by feeding the three individual component images into a color monitor. In the component images, 0 represents black and 255 represents white. Observe that each component image into the monitor is a grayscale image. The monitor does the job of combining the intensities of these images to generate an RGB image. Figure 7.9(b) shows the three hidden surface planes of the cube in Fig. 7.8, generated in a similar manner.

Acquiring a color image is the process shown in Fig. 7.9(a) in reverse. A color image can be acquired by using three filters, sensitive to red, green, and blue, respectively. When we view a color scene with a monochrome camera equipped with one of these filters, the result is a monochrome image whose intensity is proportional to the response of that filter. Repeating this process with each filter produces three monochrome images that are the RGB component images of the color scene. In practice, RGB color image sensors usually integrate this process into a single device. Clearly, displaying these three RGB component images as in Fig. 7.9(a) would yield an RGB color rendition of the original color scene.

THE CMY AND CMYK COLOR MODELS

As indicated in Section 7.1, cyan, magenta, and yellow are the secondary colors of light or, alternatively, they are the primary colors of pigments. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light from reflected white light, which itself is composed of equal amounts of red, green, and blue light.

Most devices that deposit colored pigments on paper, such as color printers and copiers, require CMY data input or perform an RGB to CMY conversion internally. This conversion is performed using the simple operation



a b c
d e f

FIGURE 7.25 (a)–(d) Red (R), green (G), blue (B), and near-infrared (IR) components of a LANDSAT multispectral image of the Washington, D.C. area. (e) RGB color composite image obtained using the IR, G, and B component images. (f) RGB color composite image obtained using the R, IR, and B component images. (Original multispectral images courtesy of NASA.)

the fourth is in the near infrared (IR) band (see Table 1.1 and Fig. 1.10). The latter band is responsive to the biomass content of a scene, and we want to use this fact to create a composite RGB color image in which vegetation is emphasized and the other components of the scene are displayed in more muted tones.

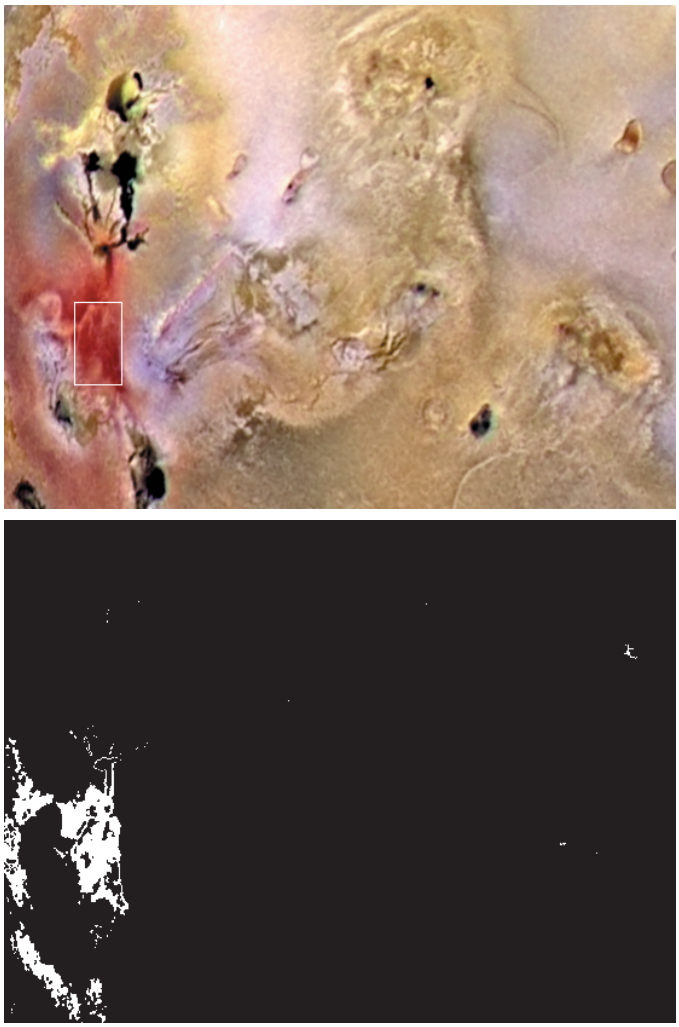
Figure 7.25(e) is an RGB composite obtained by replacing the red image by infrared. As you see, vegetation shows as a bright red, and the other components of the scene, which had a weaker response in the near-infrared band, show in pale shades of blue-green. Figure 7.25(f) is a similar image, but with the green replaced by infrared. Here, vegetation shows in a bright green color, and the other components of the scene show in purplish color shades, indicating that their major components are in the red and blue bands. Although the last two images do not introduce any new physical information, these images are much easier to interpret visually once it is known that the dominant component of the images are pixels of areas heavily populated by vegetation.

The type of processing just illustrated uses the physical characteristics of a single band in a multispectral image to emphasize areas of interest. The same approach can help visualize events of interest

when compared to a spherical or elliptical enclosure. Note that the preceding discussion is a generalization of the color-slicing method introduced in Section 7.5.

EXAMPLE 7.15: Color segmentation in RGB color space.

The rectangular region shown Fig. 7.42(a) contains samples of reddish colors we wish to segment out of the color image. This is the same problem we considered in Example 7.14 using hue, but now we approach the problem using RGB color vectors. The approach followed was to compute the mean vector **a** using the color points contained within the rectangle in Fig. 7.42(a), and then to compute the standard deviation of the red, green, and blue values of those samples. A box was centered at **a**, and its dimensions along each of the RGB axes were selected as 1.25 times the standard deviation of the data along the corresponding axis. For example, let σ_R denote the standard deviation of the red components

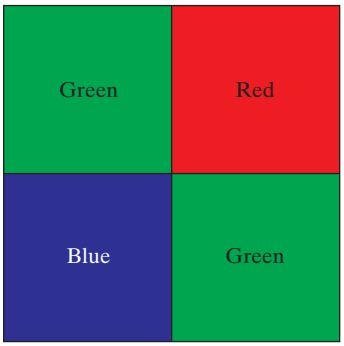


a
b

FIGURE 7.42 Segmentation in RGB space. (a) Original image with colors of interest shown enclosed by a rectangle. (b) Result of segmentation in RGB vector space. Compare with Fig. 7.40(h).

7.24* Given an image in the RGB, CMY, or CMYK color system, how would you implement the color equivalent of gray-scale histogram matching (specification) from Section 3.3?

7.25 Consider the following 500×500 RGB image, in which the squares are fully saturated red, green, and blue, and each of the colors is at maximum intensity. An HSI image is generated from this image. Answer the following questions.



(a) Describe the appearance of each HSI component image.

(b)* The saturation component of the HSI image is smoothed using an averaging kernel of size 125×125. Describe the appearance of the result. (You may ignore image border effects in the filtering operation.)

(c) Repeat (b) for the hue image.

7.26 Answer the following.

(a)* Refer to the discussion in Section 7.7 about segmentation in the RGB color space. Give a procedure (in flow chart form) for determining whether a color vector (point) $\mathbf{z}$ is inside a cube with sides W, centered at an average color vector $\mathbf{a}$. Distance computations are not allowed.

(b) If the box is aligned with the axes this process also can be implemented on an image-by-image basis. Show how you would do it.

7.27 Show that Eq. (7-49) reduces to Eq. (7-48) when $\mathbf{C} = \mathbf{I}$, the identity matrix.

7.28 Sketch the surface in RGB space for the points that satisfy the equation

$$D(\mathbf{z},\mathbf{a}) = \left[(\mathbf{z}-\mathbf{a})^T \mathbf{C}^{-1} (\mathbf{z}-\mathbf{a})\right]^{\frac{1}{2}} = D_0$$

where D_0 is a positive constant. Assume that $\mathbf{a} = \mathbf{0}$, and that

$$\mathbf{C} = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

7.29 Refer to the discussion on color edge detection in Section 7.7. One might think that a logical approach for defining the gradient of an RGB image at any point (x, y) would be to compute the gradient vector (see Section 3.6) of each component image and then form a gradient vector for the color image by summing the three individual gradient vectors. Unfortunately, this method can at times yield erroneous results. Specifically, it is possible for a color image with clearly defined edges to have a zero gradient if this method were used. Give an example of such an image. (*Hint:* To simplify your analysis, set one of the color planes to a constant value.)

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book web site: www.ImageProcessingPlace.com).

7.1 RGB color cube.

(a)* (a) Write a function **g = rgbcube4e(vz,vy,vz)** to generate and display the RGB color cube in Fig. 7.8 (see Fig. 7.7 for axis-color definitions). The inputs are the three coordinates of your viewing position with reference to the origin of the cube. You should be able to view the cube from any 3-D viewpoint, and be able to extract any of its face images. Output **g** is an image of the cube displayed by this function. (*Hint:* Consider using MATLAB function **patch** to generate the cube, and the pair of functions **getframe** and **frame2im** to capture **g**.)

8 Image Compression and Watermarking

> But life is short and information endless ... Abbreviation is a necessary evil and the abbreviator's business is to make the best of a job which, although bad, is still better than nothing.
>
> *Aldous Huxley*
>
> The Titanic will protect itself.
>
> *Robert Ballard*

Preview

Image compression, the art and science of reducing the amount of data required to represent an image, is one of the most useful and commercially successful technologies in the field of digital image processing. The number of images that are compressed and decompressed daily is staggering, and the compressions and decompressions themselves are virtually invisible to the user. Everyone who owns a digital camera, surfs the web, or streams the latest Hollywood movies over the Internet benefits from the algorithms and standards that will be discussed in this chapter. The material, which is largely introductory in nature, is applicable to both still-image and video applications. We will introduce both theory and practice, examining the most frequently used compression techniques, and describing the industry standards that make them useful. The chapter concludes with an introduction to *digital image watermarking*, the process of inserting visible and invisible data (such as copyright information) into images.

Upon competion of this chapter, students should:

- Be able to measure the amount of information in a digital image.
- Understand the main sources of data redundancy in digital images.
- Know the difference between lossy and error-free compression, and the amount of compression that is possible with each.
- Be familiar with the popular image compression standards, such as JPEG and JPEG-2000, that are in use today.
- Understand the principal image compression methods, and how and why they work.
- Be able to compress and decompress grayscale, color, and video imagery.
- Know the difference between visible, invisible, robust, fragile, public, private, restricted-key, and unrestricted-key watermarks.
- Understand the basics of watermark insertion and extraction in both the spatial and transform domain.

8.1 FUNDAMENTALS

The term *data compression* refers to the process of reducing the amount of data required to represent a given quantity of information. In this definition, *data* and *information* are not the same; data are the means by which information is conveyed. Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain *redundant data*. If we let b and b' denote the number of bits (or information-carrying units) in two representations of the same information, the *relative data redundancy*, R, of the representation with b bits is

$$R = 1 - \frac{1}{C} \tag{8-1}$$

where C, commonly called the *compression ratio*, is defined as

$$C = \frac{b}{b'} \tag{8-2}$$

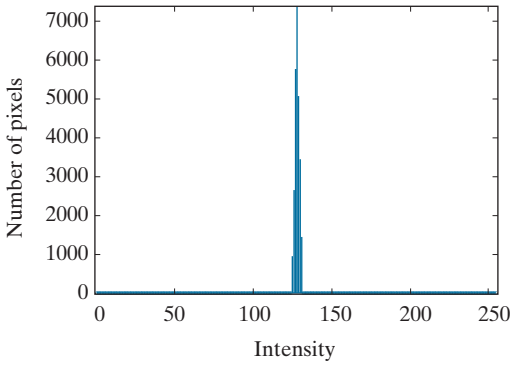
If $C = 10$ (sometimes written 10:1), for instance, the larger representation has 10 bits of data for every 1 bit of data in the smaller representation. The corresponding relative data redundancy of the larger representation is 0.9 ($R = 0.9$), indicating that 90% of its data is redundant.

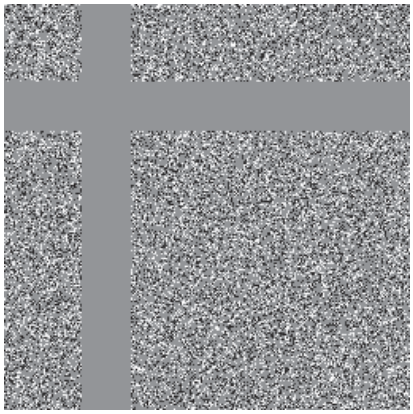
In the context of digital image compression, b in Eq. (8-2) usually is the number of bits needed to represent an image as a 2-D array of intensity values. The 2-D intensity arrays introduced in Section 2.4 are the preferred formats for human viewing and interpretation—and the standard by which all other representations are judged. When it comes to compact image representation, however, these formats are far from optimal. Two-dimensional intensity arrays suffer from three principal types of data redundancies that can be identified and exploited:

1. *Coding redundancy*. A code is a system of symbols (letters, numbers, bits, and the like) used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of *code symbols*, called a *code word*. The number of symbols in each code word is its *length*. The 8-bit codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.
2. *Spatial* and *temporal redundancy*. Because the pixels of most 2-D intensity arrays are correlated spatially (i.e., each pixel is similar to or dependent upon neighboring pixels), information is unnecessarily replicated in the representations of the correlated pixels. In a video sequence, temporally correlated pixels (i.e., those similar to or dependent upon pixels in nearby frames) also duplicate information.
3. *Irrelevant information*. Most 2-D intensity arrays contain information that is ignored by the human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used.

a b

FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.





quantization. This terminology is consistent with normal use of the word, which generally means the mapping of a broad range of input values to a limited number of output values (see Section 2.4). Because information is lost, quantization is an irreversible operation.

MEASURING IMAGE INFORMATION

Consult the book website for a brief review of information and probability theory.

In the previous sections, we introduced several ways to reduce the amount of data used to represent an image. The question that naturally arises is: How few bits are actually needed to represent the information in an image? That is, is there a minimum amount of data that is sufficient to describe an image without losing information? *Information theory* provides the mathematical framework to answer this and related questions. Its fundamental premise is that the generation of information can be modeled as a probabilistic process which can be measured in a manner that agrees with intuition. In accordance with this supposition, a random event E with probability $P(E)$ is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \tag{8-5}$$

units of information. If $P(E) = 1$ (that is, the event always occurs), $I(E) = 0$ and no information is attributed to it. Because no uncertainty is associated with the event, no information would be transferred by communicating that the event has occurred [it *always* occurs if $P(E) = 1$].

The base of the logarithm in Eq. (8-5) determines the unit used to measure information. If the base m logarithm is used, the measurement is said to be in m-ary units. If the base 2 is selected, the unit of information is the *bit*. Note that if $P(E) = ½$, $I(E) = -\log_2 ½$ or 1 bit. That is, 1 bit is the amount of information conveyed when one of two possible equally likely events occurs. A simple example is flipping a coin and communicating the result.

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_1, \ldots, a_J\}$ with associated probabilities $\{P(a_1), P(a_1), \ldots, P(a_J)\}$, the average information per source output, called the *entropy* of the source, is

$$H = -\sum_{j=1}^{J} P(a_j) \log P(a_j) \tag{8-6}$$

The a_j in this equation are called *source symbols*. Because they are statistically independent, the source itself is called a *zero-memory source*.

If an image is considered to be the output of an imaginary zero-memory "intensity source," we can use the histogram of the observed image to estimate the symbol probabilities of the source. Then, the intensity source's entropy becomes

Equation (8-6) is for zero-memory sources with J source symbols. Equation (8-7) uses probablitiy estimates for the $L-1$ intensity values in an image.

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k) \tag{8-7}$$

where variables L, r_k, and $p_r(r_k)$ are as defined earlier and in Section 3.3. Because the base 2 logarithm is used, Eq. (8-7) is the average information per intensity output of the imaginary intensity source in bits. It is not possible to code the *intensity values* of the imaginary source (and thus the sample image) with fewer than $\tilde{H}$ bits/pixel.

EXAMPLE 8.2: Image entropy estimates.

The entropy of the image in Fig. 8.1(a) can be estimated by substituting the intensity probabilities from Table 8.1 into Eq. (8-7):

$$\begin{aligned}\tilde{H} &= -\left[0.25 \log_2 0.25 + 0.47 \log_2 0.47 + 0.25 \log_2 0.25 + 0.03 \log_2 0.03\right] \\ &= -\left[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)\right] \\ &\approx 1.6614 \text{ bits/pixel}\end{aligned}$$

In a similar manner, the entropies of the images in Fig. 8.1(b) and (c) can be shown to be 8 bits/pixel and 1.566 bits/pixel, respectively. Note that the image in Fig. 8.1(a) appears to have the most visual information, but has almost the lowest computed entropy—1.66 bits/pixel. The image in Fig. 8.1(b) has almost five times the entropy of the image in (a), but appears to have about the same (or less) visual information. The image in Fig. 8.1(c), which seems to have little or no information, has almost the same entropy as the image in (a). The obvious conclusion is that the amount of entropy, and thus information in an image, is far from intuitive.

Shannon's First Theorem

Recall that the variable-length code in Example 8.1 was able to represent the intensities of the image in Fig. 8.1(a) using only 1.81 bits/pixel. Although this is higher than the 1.6614 bits/pixel entropy estimate from Example 8.2, Shannon's first theorem, also called the *noiseless coding theorem* (Shannon [1948]), assures us that the

image in Fig. 8.1(a) can be represented with as few as 1.6614 bits/pixel. To prove it in a general way, Shannon looked at representing groups of consecutive source symbols with a single code word (rather than one code word per source symbol), and showed that

$$\lim_{n\to\infty}\left[\frac{L_{\text{avg},n}}{n}\right] = H \tag{8-8}$$

where $L_{\text{avg},n}$ is the average number of code symbols required to represent all n-symbol groups. In the proof, he defined the *nth extension* of a zero-memory source to be the hypothetical source that produces n-symbol blocks† using the symbols of the original source, and computed $L_{\text{avg},n}$ by applying Eq. (8-4) to the code words used to represent the n-symbol blocks. Equation (8-8) tells us that $L_{\text{avg},n}/n$ can be made arbitrarily close to H by encoding infinitely long extensions of the single-symbol source. That is, it is possible to represent the output of a zero-memory source with an average of H information units per source symbol.

If we now return to the idea that an image is a "sample" of the intensity source that produced it, a block of n source symbols corresponds to a group of n adjacent pixels. To construct a variable-length code for n-pixel blocks, the relative frequencies of the blocks must be computed. But the nth extension of a hypothetical intensity source with 256 intensity values has 256^n possible n-pixel blocks. Even in the simple case of $n = 2$, a 65,536 element histogram and up to 65,536 variable-length code words must be generated. For $n = 3$, as many as 16,777,216 code words are needed. So even for small values of n, computational complexity limits the usefulness of the extension coding approach in practice.

Finally, we note that although Eq. (8-7) provides a lower bound on the compression that can be achieved when directly coding statistically independent pixels, it breaks down when the pixels of an image are correlated. Blocks of correlated pixels can be coded with fewer average bits per pixel than the equation predicts. Rather than using source extensions, less correlated descriptors (such as intensity run-lengths) are normally selected and coded without extension. This was the approach used to compress Fig. 8.1(b) in the section on spatial and temporal redundancy. When the output of a source of information depends on a finite number of preceding outputs, the source is called a *Markov source* or *finite memory source*.

FIDELITY CRITERIA

It was noted earlier that the removal of "irrelevant visual" information involves a loss of real or quantitative image information. Because information is lost, a means of quantifying the nature of the loss is needed. Two types of criteria can be used for such an assessment: (1) objective fidelity criteria, and (2) subjective fidelity criteria.

†The output of the nth extension is an n-tuple of symbols from the underlying *single-symbol source*. It was considered a *block random variable* in which the probability of each n-tuple is the product of the probabilities of its individual symbols. The entropy of the nth extension is then n times the entropy of the single-symbol source from which it is derived.

In the third and final stage of the encoding process, the *symbol coder* of Fig. 8.5 generates a fixed-length or variable-length code to represent the quantizer output, and maps the output in accordance with the code. In many cases, a variable-length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values, thus minimizing coding redundancy. This operation is reversible. Upon its completion, the input image has been processed for the removal of each of the three redundancies described in the previous sections.

The Decoding or Decompression Process

The decoder of Fig. 8.5 contains only two components: a *symbol decoder* and an *inverse mapper*. They perform, in reverse order, the inverse operations of the encoder's symbol encoder and mapper. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general decoder model. In video applications, decoded output frames are maintained in an internal frame store (not shown) and used to reinsert the temporal redundancy that was removed at the encoder.

IMAGE FORMATS, CONTAINERS, AND COMPRESSION STANDARDS

In the context of digital imaging, an *image file format* is a standard way to organize and store image data. It defines how the data is arranged and the type of compression (if any) that is used. An *image container* is similar to a file format, but handles multiple types of image data. Image *compression standards*, on the other hand, define procedures for compressing and decompressing images—that is, for reducing the amount of data needed to represent an image. These standards are the underpinning of the widespread acceptance of image compression technology.

Figure 8.6 lists the most important image compression standards, file formats, and containers in use today, grouped by the type of image handled. The entries in blue are international standards sanctioned by the *International Standards Organization* (ISO), the *International Electrotechnical Commission* (IEC), and/or the *International Telecommunications Union* (ITU-T)—a *United Nations* (UN) organization that was once called the *Consultative Committee of the International Telephone and Telegraph* (CCITT). Two video compression standards, VC-1 by the *Society of Motion Pictures and Television Engineers* (SMPTE) and AVS by the *Chinese Ministry of Information Industry* (MII), are also included. Note that they are shown in black, which is used in Fig. 8.6 to denote entries that are not sanctioned by an international standards organization.

Tables 8.3 through 8.5 summarize the standards, formats, and containers listed in Fig. 8.6. Responsible organizations, targeted applications, and key compression methods are identified. The compression methods themselves are the subject of Sections 8.2 through 8.11, where we will describe the principal lossy and error-free compression methods in use today. The focus of these sections is on methods that have proven useful in mainstream binary, continuous-tone still-image, and video compression standards. The standards themselves are used to demonstrate the methods presented. In Tables 8.3 through 8.5, forward references to the relevant sections in which the compression methods are described are enclosed in square brackets.

$$2^0 + 2^1 + 2^2 \le 8 < 2^0 + 2^1 + 2^2 + 2^3$$
$$7 \le 8 < 15$$

The unary code of 3 is 1110 and Eq. (8-17) of Step 2 yields

$$8 - \sum_{j=0}^{3-1} 2^{j+0} = 8 - \sum_{j=0}^{2} 2^j = 8 - \left(2^0 + 2^1 + 2^2\right) = 8 - 7 = 1 = 0001$$

which when truncated to its $3 + 0$ least significant bits becomes 001. The concatenation of the results from Steps 1 and 2 then yields 1110001. Note that this is the entry in column 4 of Table 8.6 for $n = 8$. Finally, we note that like the Huffman codes of the last section, the Golomb codes of Table 8.6 are variable-length, instantaneous, and uniquely decodable block codes.

8.4 ARITHMETIC CODING

With reference to Tables 8.3–8.5, arithmetic coding is used in

- JBIG1
- JBIG2
- JPEG-2000
- H.264
- MPEG-4 AVC

and other compression standards.

Unlike the variable-length codes of the previous two sections, *arithmetic coding* generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias (Abramson [1963]), a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller, and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound established by Shannon's first theorem of Section 8.1.

Figure 8.12 illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, $a_1a_2a_3a_3a_4$, from a four-symbol source is coded. At the start of the coding process, the message is assumed to occupy the entire half-open interval $[0, 1)$. As Table 8.7 shows, this interval is subdivided initially into four regions based on the probabilities of each source symbol. Symbol a_1, for example, is associated with subinterval $[0, 0.2)$. Because it is the first symbol of the message being coded, the message interval is initially narrowed to $[0, 0.2)$. Thus, in Fig. 8.12, $[0, 0.2)$ is expanded to the full height of the figure, and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original

TABLE 8.7
Arithmetic coding example.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

EXAMPLE 8.10: CCITT compression example.

Figure 8.16(a) is a 300 dpi scan of a 7×9.25 inch book page displayed at about 1/3 scale. Note that about half of the page contains text, around 9% is occupied by a halftone image, and the rest is white space. A section of the page is enlarged in Fig. 8.16(b). Keep in mind that we are dealing with a binary image; the illusion of gray tones is created, as was described in Section 4.5, by the halftoning process used in printing. If the binary pixels of the image in Fig. 8.16(a) are stored in groups of 8 pixels per byte, the 1952×2697 bit scanned image, commonly called a document, requires 658,068 bytes. An uncompressed PDF file of the document (created in Photoshop) requires 663,445 bytes. CCITT Group 3 compression reduces the file to 123,497 bytes, resulting in a compression ratio $C = 5.37$. CCITT Group 4 compression reduces the file to 110,456 bytes, increasing the compression ratio to about 6.

8.7 SYMBOL-BASED CODING

With reference to Tables 8.3–8.5, symbol-based coding is used in

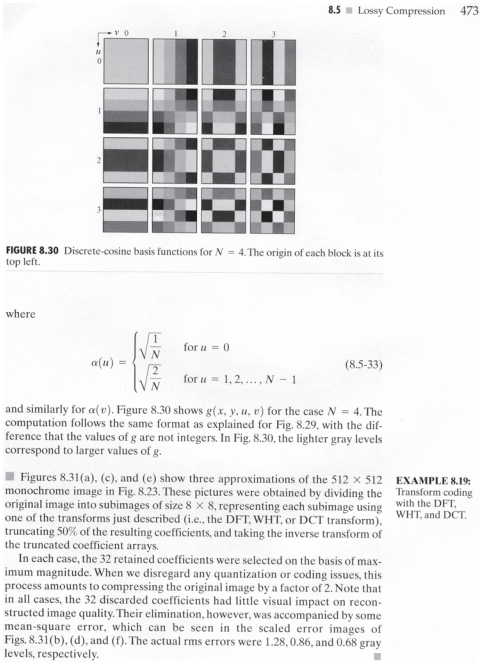
- JBIG2

compression.

In *symbol-* or *token-based* coding, an image is represented as a collection of frequently occurring subimages, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$, where each (x_i, y_i) pair specifies the location of a symbol in the image and *token* t_i is the address of the symbol or subimage in the dictionary. That is, each triplet represents an instance of a dictionary symbol in the image. Storing repeated symbols only once can compress images significantly, particularly in document storage and retrieval applications where the symbols are often character bitmaps that are repeated many times.

a b

FIGURE 8.16
A binary scan of a book page: (a) scaled to show the general page content; (b) scaled to show the binary pixels used in dithering.



8.5 ■ Lossy Compression 473

FIGURE 8.30 Discrete-cosine basis functions for $N = 4$. The origin of each block is at its top left.

where

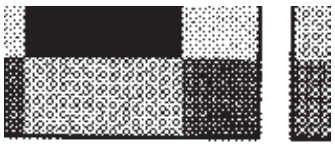
$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \ldots, N-1 \end{cases} \qquad (8.5\text{-}33)$$

and similarly for $\alpha(v)$. Figure 8.30 shows $g(x, y, u, v)$ for the case $N = 4$. The computation follows the same format as explained for Fig. 8.29, with the difference that the values of g are not integers. In Fig. 8.30, the lighter gray levels correspond to larger values of g.

EXAMPLE 8.19: Transform coding with the DFT, WHT, and DCT.

■ Figures 8.31(a), (c), and (e) show three approximations of the 512×512 monochrome image in Fig. 8.23. These pictures were obtained by dividing the original image into subimages of size 8×8, representing each subimage using one of the transforms just described (i.e., the DFT, WHT, or DCT transform), truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays.

In each case, the 32 retained coefficients were selected on the basis of maximum magnitude. When we disregard any quantization or coding issues, this process amounts to compressing the original image by a factor of 2. Note that in all cases, the 32 discarded coefficients had little visual impact on reconstructed image quality. Their elimination, however, was accompanied by some mean-square error, which can be seen in the scaled error images of Figs. 8.31(b), (d), and (f). The actual rms errors were 1.28, 0.86, and 0.68 gray levels, respectively. ■



r $N = 4$. Th

Consider the simple bilevel image in Fig. 8.17(a). It contains the single word, *banana*, which is composed of three unique symbols: a *b*, three *a*'s, and two *n*'s. Assuming that the *b* is the first symbol identified in the coding process, its 9×7 bitmap is stored in location 0 of the symbol dictionary. As Fig. 8.17(b) shows, the token identifying the *b* bitmap is 0. Thus, the first triplet in the encoded image's representation [see Fig. 8.17(c)] is (0, 2, 0), indicating that the upper-left corner (an arbitrary convention) of the rectangular bitmap representing the *b* symbol is to be placed at location (0, 2) in the decoded image. After the bitmaps for the *a* and *n* symbols have been identified and added to the dictionary, the remainder of the image can be encoded with five additional triplets. As long as the six triplets required to locate the symbols in the image, together with the three bitmaps required to define them, are smaller than the original image, compression occurs. In this case, the starting image has $9 \times 51 \times 1$ or 459 bits and, assuming that each triplet is composed of three bytes, the compressed representation has $(6 \times 3 \times 8) + [(9 \times 7) + (6 \times 7) + (6 \times 6)]$ or 285 bits; the resulting compression ratio $C = 1.61$. To decode the symbol-based representation in Fig. 8.17(c), you simply read the bitmaps of the symbols specified in the triplets from the symbol dictionary and place them at the spatial coordinates specified in each triplet.

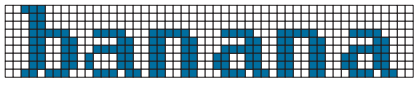
Symbol-based compression was proposed in the early 1970s (Ascher and Nagy [1974]), but has become practical only recently. Advances in symbol-matching algorithms (see Chapter 13) and increased CPU computer processing speeds have made it possible to both select dictionary symbols and to find where they occur in an image in a timely manner. And like many other compression methods, symbol-based decoding is significantly faster than encoding. Finally, we note that both the symbol bitmaps that are stored in the dictionary and the triplets used to reference them themselves can be encoded to further improve compression performance. If, as in Fig. 8.17, only exact symbol matches are allowed, the resulting compression is lossless; if small differences are permitted, some level of reconstruction error will be present.

JBIG2 COMPRESSION

JBIG2 is an international standard for bilevel image compression. By segmenting an image into overlapping and/or non-overlapping regions of text, halftone, and generic content, compression techniques that are specifically optimized for each type of content are employed:

a b c

FIGURE 8.17
(a) A bi-level document, (b) symbol dictionary, and (c) the triplets used to locate the symbols in the document.



Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0)
(3,10, 1)
(3, 18, 2)
(3, 26, 1)
(3, 34, 2)
(3, 42, 1)

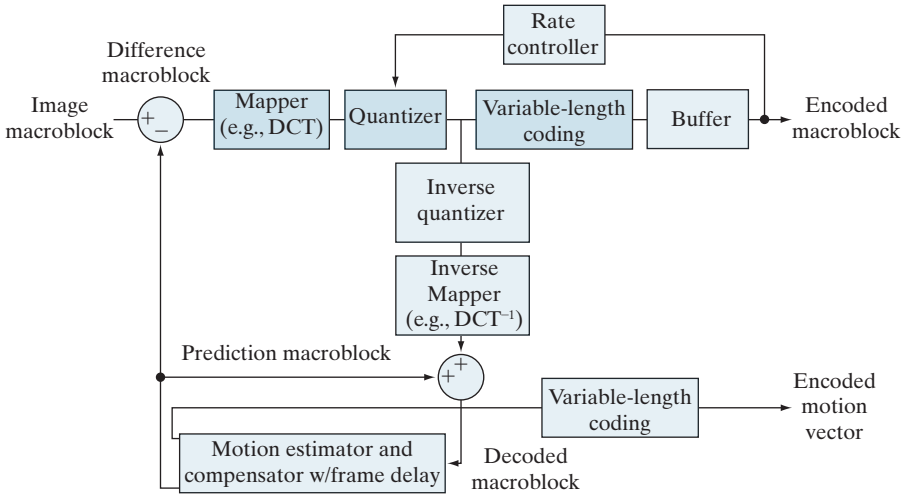


FIGURE 8.36
A typical motion compensated video encoder.

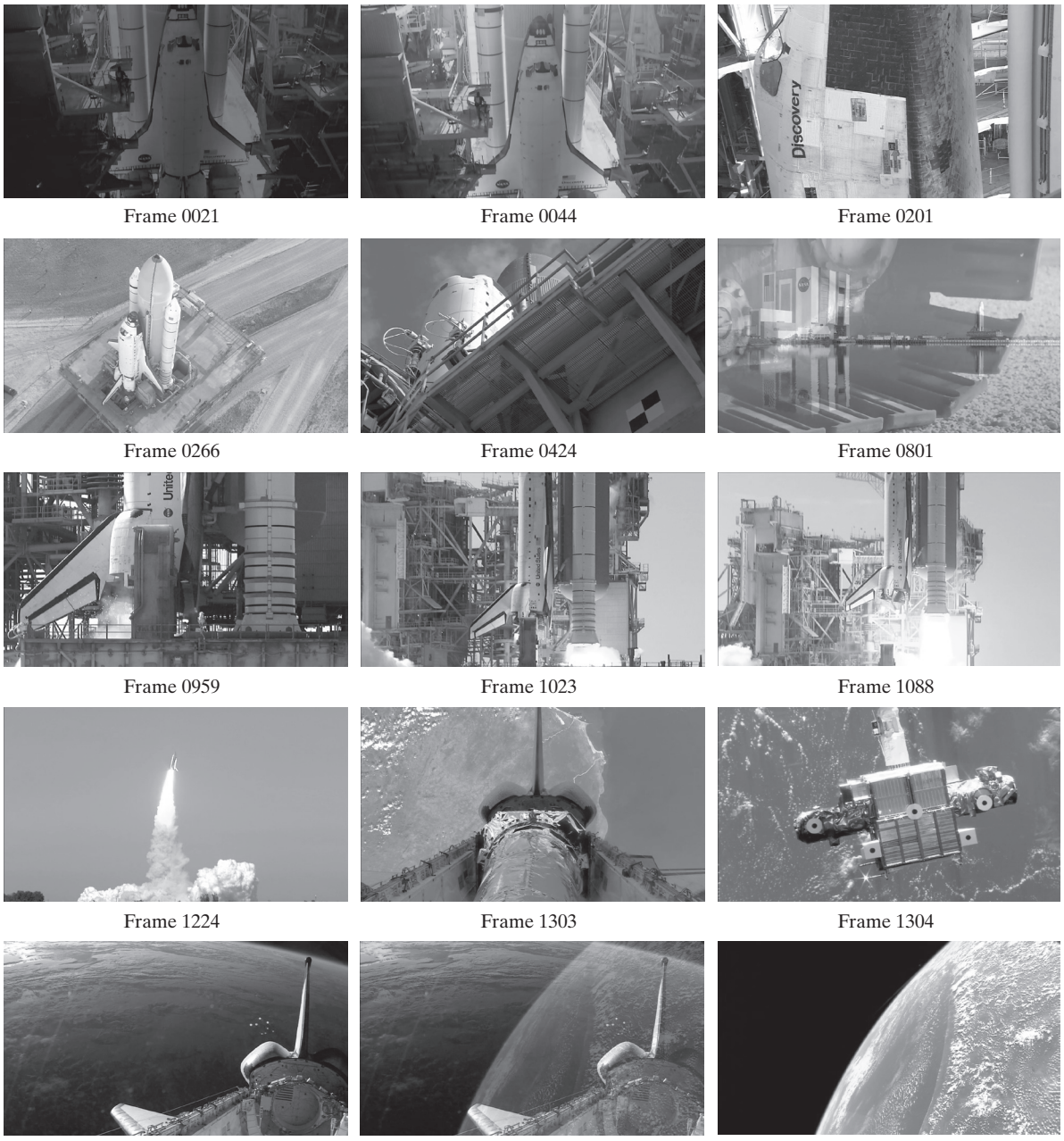
EXAMPLE 8.22: Video compression example.

We conclude our discussion of motion compensated predictive coding with an example illustrating the kind of compression that is possible with modern video compression methods. Figure 8.37 shows fifteen frames of a 1 minute HD (1280×720) full-color NASA video, parts of which have been used throughout this section. Although the images shown are monochrome, the video is a sequence of 1,829 full-color frames. Note that there are a variety of scenes, a great deal of motion, and multiple fade effects. For example, the video opens with a 150 frame fade-in from black, which includes frames 21 and 44 in Fig. 8.37, and concludes with a fade sequence containing frames 1595, 1609, and 1652 in Fig. 8.37, followed by a final fade to black. There are also several abrupt scene changes, like the change involving frames 1303 and 1304 in Fig. 8.37.

An H.264 compressed version of the NASA video stored as a Quicktime file (see Table 8.5) requires 44.56 MB of storage, plus another 1.39 MB for the associated audio. The video quality is excellent. About 5 GB of data would be needed to store the video frames as uncompressed full-color images. It should be noted that the video contains sequences involving both rotation and scale change (e.g., the sequence including frames 959, 1023, and 1088 in Fig. 8.37). The discussion in this section, however, has been limited to translation alone. (See the book website for the NASA video segment used in this example.)

LOSSY PREDICTIVE CODING

In this section, we add a quantizer to the lossless predictive coding model introduced earlier, and examine the trade-off between reconstruction accuracy and compression performance within the context of spatial predictors. As Fig. 8.38 shows, the quantizer, which replaces the nearest integer function of the error-free encoder, is inserted between the symbol encoder and the point at which the prediction error is formed. It maps the prediction error into a limited range of outputs, denoted $\dot{e}(n)$, which establish the amount of compression and distortion that occurs.



Frame 0021 Frame 0044 Frame 0201

Frame 0266 Frame 0424 Frame 0801

Frame 0959 Frame 1023 Frame 1088

Frame 1224 Frame 1303 Frame 1304

Frame 1595 Frame 1609 Frame 1652

FIGURE 8.37 Fifteen frames from an 1829-frame, 1-minute NASA video. The original video is in HD full color. (Courtesy of NASA.)

FIGURE 8.46 Four JPEG-2000 approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image. (Compare the results in rows 1 and 2 with the JPEG results in Fig. 8.29.).

A visual comparison of the error images in rows 1 and 2 of Fig. 8.46 with the corresponding images in Figs. 8.29(b) and (e) reveals a noticeable decrease of error in the JPEG-2000 results—3.86 and 5.77 intensity levels, as opposed to 5.4 and 10.7 intensity levels for the JPEG results. The computed errors favor the wavelet-based results at both compression levels. Besides decreasing reconstruction error, wavelet coding dramatically increases (in a subjective sense) image quality. Note that the blocking artifact that dominated the JPEG results [see Figs. 8.29(c) and (f)] is not present in Fig. 8.46. Finally, we note that the compression achieved in rows 3 and 4 of Fig. 8.46 is not practical with JPEG. JPEG-2000 provides useable images that are compressed by more than 100:1, with the most objectionable degradation being increased image blur.

8.12 DIGITAL IMAGE WATERMARKING

The methods and standards of Sections 8.2 through 8.11 make the distribution of images (in photographs or videos) on digital media and over the Internet practical. Unfortunately, the images so distributed can be copied repeatedly and without error, putting the rights of their owners at risk. Even when encrypted for distribution, images are unprotected after decryption. One way to discourage illegal duplication is to insert one or more items of information, collectively called a *watermark*, into potentially vulnerable images in such a way that the watermarks are inseparable from the images themselves. As integral parts of the watermarked images, they protect the rights of their owners in a variety of ways, including:

1. *Copyright identification*. Watermarks can provide information that serves as proof of ownership when the rights of the owner have been infringed.
2. *User identification* or *fingerprinting*. The identity of legal users can be encoded in watermarks and used to identify sources of illegal copies.
3. *Authenticity determination*. The presence of a watermark can guarantee that an image has not been altered, assuming the watermark is designed to be destroyed by any modification of the image.
4. *Automated monitoring*. Watermarks can be monitored by systems that track when and where images are used (e.g., programs that search the Web for images placed on Web pages). Monitoring is useful for royalty collection and/or the location of illegal users.
5. *Copy protection*. Watermarks can specify rules of image usage and copying (e.g., to DVD players).

In this section, we provide a brief overview of *digital image watermarking*, which is the process of inserting data into an image in such a way that it can be used to make an assertion about the image. The methods described have little in common with the compression techniques presented in the previous sections (although they do involve the coding of information). In fact, watermarking and compression are in some ways opposites. While the objective in compression is to reduce the amount of data used to represent images, the goal in watermarking is to add information and data (i.e., watermarks) to them. As will be seen in the remainder of the section, the watermarks themselves can be either visible or invisible.

A *visible watermark* is an opaque or semi-transparent subimage or image that is placed on top of another image (i.e., the image being watermarked) so that it is obvious to the viewer. Television networks often place visible watermarks (fashioned after their logos) in the upper or lower right-hand corner of the television screen. As the following example illustrates, visible watermarking typically is performed in the spatial domain.

EXAMPLE 8.29: A simple visible watermark.

The image in Fig. 8.47(b) is the lower right-hand quadrant of the image in Fig. 8.9(a) with a scaled version of the watermark in Fig. 8.47(a) overlaid on top of it. Letting f_w denote the watermarked image, we can express it as a linear combination of the unmarked image f and watermark w using

$$f_w = (1-\alpha)f + \alpha w \tag{8-68}$$

where constant α controls the relative visibility of the watermark and the underlying image. If α is 1, the watermark is opaque and the underlying image is completely obscured. As α approaches 0, more of the underlying image and less of the watermark is seen. In general, $0 < \alpha \leq 1$; in Fig. 8.47(b), $\alpha = 0.3$. Figure 8.47(c) is the computed difference (scaled in intensity) between the watermarked image in (b) and the unmarked image in Fig. 8.9(a). Intensity 128 represents a difference of 0. Note that the underlying image is clearly visible through the "semi-transparent" watermark. This is evident in both Fig. 8.47(b) and the difference image in Fig. 8.47(c).

Unlike the visible watermark of the previous example, *invisible watermarks* cannot be seen with the naked eye. They are imperceptible but can be recovered with an appropriate decoding algorithm. Invisibility is assured by inserting them as visually redundant information [information that the human visual system ignores or cannot

a
b c

FIGURE 8.47 A simple visible watermark: (a) watermark; (b) the watermarked image; and (c) the difference between the watermarked image and the original (non-watermarked) image.

8.29* Derive the Lloyd-Max decision and reconstruction levels for $L = 4$ and the uniform probability density function

$$p(s) = \begin{cases} \frac{1}{2A} & -A \le s \le A \\ 0 & \text{otherwise} \end{cases}$$

8.30 A radiologist from a well-known research hospital recently attended a medical conference at which a system that could transmit 4096×4096 12-bit digitized X-ray images over standard T1 (1.544 Mb/s) phone lines was exhibited. The system transmitted the images in a compressed format using a progressive technique in which a reasonably good approximation of the X-ray was first reconstructed at the viewing station, then refined gradually to produce an error-free display. The transmission of the data needed to generate the first approximation took approximately 5 or 6 s. Refinements were made every 5 or 6 s (on the average) for the next 1 min, with the first and last refinements having the most and least significant impact on the reconstructed X-ray, respectively. The physician was favorably impressed with the system, because she could begin her diagnosis by using the first approximation of the X-ray and complete it as the error-free reconstruction of the X-ray was being generated. Upon returning to her office, she submitted a purchase request to the hospital administrator. Unfortunately, the hospital was on a relatively tight budget, which recently had been stretched by the hiring of an aspiring young electrical engineering graduate. To appease the radiologist, the administrator gave the young engineer the task of designing such a system. (He thought it might be cheaper to design and build a similar system in-house. The hospital currently owned some of the elements of such a system, but the transmission of the raw X-ray data took more than 2 min.) The administrator asked the engineer to have an initial block diagram by the afternoon staff meeting. With little time and only a copy of *Digital Image Processing* from his recent school days in hand, the engineer was able to devise a system conceptually to satisfy the transmission and associated compression requirements. Construct a conceptual block diagram of such a system, specifying the compression techniques you would recommend.

8.31 Show that the lifting-based wavelet transform defined by Eq. (8-61) is equivalent to the traditional FWT filter bank implementation using the coefficients in Table 6.1. Define the filter coefficients in terms of α, β, γ, δ, and K.

8.32 Compute the quantization step sizes of the subbands for a JPEG-2000 encoded image in which derived quantization is used and 8 bits are allotted to the mantissa and exponent of the $2LL$ subband.

8.33 How would you add a visible watermark to an image in the frequency domain?

8.34* Design an invisible watermarking system based on the discrete Fourier transform.

8.35 Design an invisible watermarking system based on the discrete wavelet transform.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

8.1 Write a function **e = entropy4e(f,n)** to compute the amount of information in bits of matrix **f**, where **n** is the number of possible values for each element of **f**. (If matrix **f** is an 8-bit image, **n** would be 256.) Assume **f** to be the output of a zero-memory source so that Eq. (8-7) can be used. Make sure your function can handle negative pixel values. Use your function to compute the entropy of the image **lena.tif** from the book website.

8.2* Write a function **cr = compressionRatio4e(f,fc)** to compute the compression ratio of image **f** and compressed image **fc**. If **f** or **fc** is a string, assume that it is the name of a file; otherwise, **f** or **fc** is an image variable. (*Hint:* Do not use MATLAB's **whos** function, since it reports an extra 124 bytes for every field in a structure. Instead, add up the memory of every field.)

(a) Use your function to compute the com-

9 Morphological Image Processing

In form and feature, face and limb,
I grew so like my brother
That folks got taking me for him
And each for one another.

Henry Sambrook Leigh, Carols of Cockayne, The Twins

Preview

The word *morphology* commonly denotes a branch of biology that deals with the form and structure of animals and plants. We use the same word here in the context of *mathematical morphology* as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull. We are interested also in morphological techniques for pre- or postprocessing, such as morphological filtering, thinning, and pruning.

In the following sections, we will develop a number of fundamental concepts in mathematical morphology, and illustrate how they are applied in image processing. The material in this chapter begins a transition from methods whose inputs and outputs are images, to methods whose outputs are image attributes, for tasks such as object extraction and description. Morphology is one of several tools developed in the remainder of the book—such as segmentation, feature extraction, and object recognition—that form the foundation of techniques for extracting "meaning" from an image. The material in the following sections of this chapter deals with methods for processing both binary and grayscale images.

Upon completion of this chapter, readers should:

- Understand basic concepts of mathematical morphology, and how to apply them to digital image processing.
- Be familiar with the tools used for binary image morphology, including erosion, dilation, opening, closing, and how to combine them to generate more complex tools.
- Be able to develop algorithms based on binary image morphology for performing tasks such as morphological smoothing, edge detection, extracting connected components, and skeletonizing.
- Be familiar with how binary image morphology can be extended to grayscale images.
- Be able to develop algorithms for grayscale image processing for tasks such as textural segmentation, granulometry, computing grayscale image gradients, and others.

9.1 PRELIMINARIES

Before proceeding, you will find it helpful to review the discussion in Section 2.4 dealing with representing images, the discussion on connectivity in Section 2.5, and the discussion on sets in Section 2.6.

The language of mathematical morphology is set theory. As such, morphology offers a unified and powerful approach to numerous image processing problems. When working with images, sets in mathematical morphology represent objects in those images. In binary images, the sets in question are members of the 2-D integer space Z^2, where each element of a set is a tuple (2-D vector) whose coordinates are the coordinates of an object (typically foreground) pixel in the image. Grayscale digital images can be represented as sets whose components are in Z^3. In this case, two components of each element of the set refer to the coordinates of a pixel, and the third corresponds to its discrete intensity value. Sets in higher dimensional spaces can contain other image attributes, such as color and time-varying components.

Morphological operations are defined in terms of sets. In image processing, we use morphology with two types of sets of pixels: *objects* and *structuring elements* (SE's). Typically, objects are defined as sets of foreground pixels. Structuring elements can be specified in terms of both foreground and background pixels. In addition, structuring elements sometimes contain so-called "don't care" elements, denoted by $\times$, signifying that the value of that particular element in the SE does not matter. In this sense, the value can be ignored, or it can be made to fit a desired value in the evaluation of an expression; for example, it might take on the value of a pixel in an image in applications in which value matching is the objective.

Because the images with which we work are rectangular arrays, and sets in general are of arbitrary shape, applications of morphology in image processing require that sets be embedded in rectangular arrays. In forming such arrays, we assign a background value to all pixels that are not members of object sets. The top row in Fig. 9.1 shows an example. On the left are sets in the graphical format you are accustomed to seeing in book figures. In the center, the sets have been embedded in a rectangular background (white) to form a graphical image.[†] On the right, we show a digital image (notice the grid) which is the format we use for digital image processing.

Structuring elements are defined in the same manner, and the second row in Fig. 9.1 shows an example. There is an important difference between the way we represent digital images and digital structuring elements. Observe on the top right that there is a border of background pixels surrounding the objects, while there is none in the SE. As you will learn shortly, structuring elements are used in a form similar to spatial convolution kernels (see Fig. 3.34), and the image border just described is similar to the padding we discussed in Section 3.4 and 3.5. The operations are different in morphology, but the padding and sliding operations are the same as in convolution.

In addition to the set definitions given in Section 2.6, the concept of set reflection and translation are used extensively in morphology in connection with structuring elements. The *reflection* of a set (structuring element) B about its origin, denoted by $\hat{B}$, is defined as

[†] Sets are shown as drawings of objects (e.g. squares and triangles) of arbitrary shape. A graphical image contains sets that have been embedded into a background to form a rectangular array. When we intend for a drawing to be interpreted as a *digital* image (or structuring element), we include a grid in illustrations that might otherwise be ambiguous. Objects in all drawings are shaded, and the background is shown in white. When working with actual binary images, we say that objects are *foreground* pixels. All other pixels are *background*.

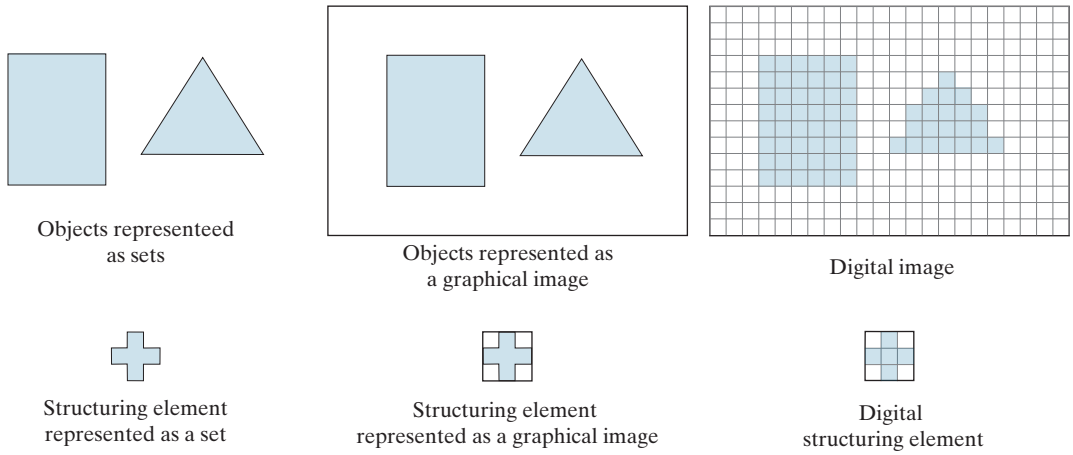


FIGURE 9.1 Top row. *Left:* Objects represented as graphical sets. *Center:* Objects embedded in a background to form a graphical image. *Right:* Object and background are digitized to form a digital image (note the grid). Second row: Example of a structuring element represented as a set, a graphical image, and finally as a digital SE.

$$\hat{B} = \{w \mid w = -b, \text{ for } b \in B\} \tag{9-1}$$

Reflection is the same operation we performed with kernels prior to spatial convolution, as explained in Section 3.4.

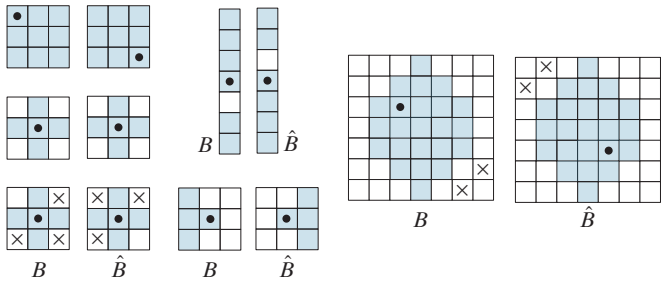
That is, if B is a set of points in 2-D, then $\hat{B}$ is the set of points in B whose (x, y) coordinates have been replaced by $(-x, -y)$. Figure 9.2 shows several examples of digital sets (structuring elements) and their reflection. The dot denotes the origin of the SE. Note that reflection consists simply of rotating an SE by 180° about its origin, and that all elements, including the background and don't care elements, are rotated.

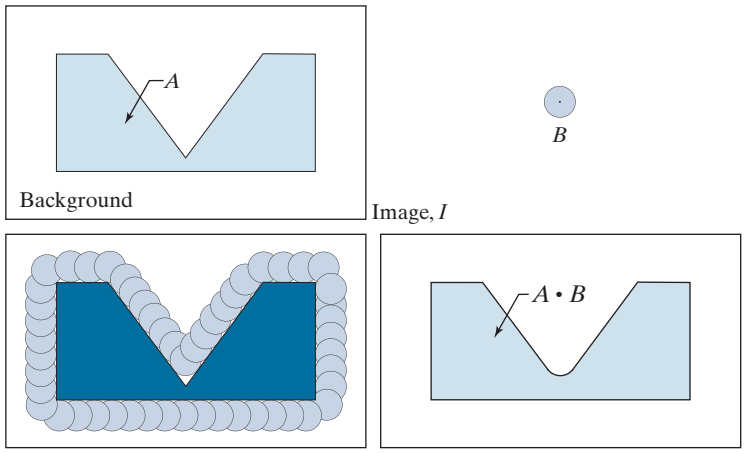
The *translation* of a set B by point $z = (z_1, z_2)$, denoted $(B)_z$, is defined as

$$(B)_z = \{c \mid c = b + z, \text{ for } b \in B\} \tag{9-2}$$

That is, if B is a set of pixels in 2-D, then $(B)_z$ is the set of pixels in B whose (x, y) coordinates have been replaced by $(x + z_1, y + z_2)$. This construct is used to translate (slide) a structuring element over an image, and each location perform a set

FIGURE 9.2 Structuring elements and their reflections about the origin (the ×'*s* are don't care elements, and the dots denote the origin). Reflection is rotation by 180° of an SE about its origin.





a b
c d

FIGURE 9.9
(a) Image I, composed of set (object) A, and background. (b) Structuring element B. (c) Translations of B such that B does not overlap any part of A. (A is shown dark for clarity.) (d) Closing of A by B.

Closing has a similar geometric interpretation, except that now we translate B *outside* A. The closing is then the *complement* of the union of all translations of B that *do not* overlap A. Figure 9.9 illustrates this concept. Note that the boundary of the closing is determined by the furthest points B could reach without going inside any part of A. Based on this interpretation, we can write the closing of A by B as

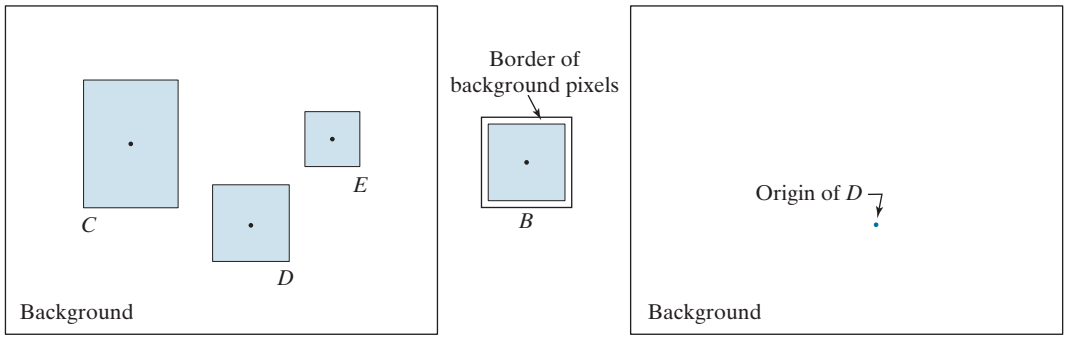
$$A \bullet B = \left[\bigcup \left\{(B)_z \,\middle|\, (B)_z \cap A = \varnothing\right\}\right]^c \tag{9-13}$$

EXAMPLE 9.3: Morphological opening and closing.

Figure 9.10 shows in more detail the process and properties of opening and closing. Unlike Figs. 9.8 and 9.9, whose main objectives are overall geometrical interpretations, this figure shows the individual processes and also pays more attention to the relationship between the scale of the final results and the size of the structuring elements.

Figure 9.10(a) shows an image containing a single object (set) A, and a disk structuring element. Figure 9.10(b) shows various positions of the structuring element during erosion. This process resulted in the disjoint set in Fig. 9.10(c). Note how the bridge between the two main sections was eliminated. Its width was thin in relation to the diameter of the structuring element, which could not be completely contained in this part of the set, thus violating the definition of erosion. The same was true of the two rightmost members of the object. Protruding elements where the disk did not fit were eliminated. Figure 9.10(d) shows the process of dilating the eroded set, and Fig. 9.10(e) shows the final result of opening. Morphological opening removes regions that cannot contain the structuring element, smoothes object contours, breaks thin connections, and removes thin protrusions.

Figures 9.10(f) through (i) show the results of closing A with the same structuring element. As with opening, closing also smoothes the contours of objects. However, unlike opening, closing tends to join narrow breaks, fills long thin gulfs, and fills objects smaller than the structuring element. In this example, the principal result of closing was that it filled the small gulf on the left of set A.



a b c

FIGURE 9.13 Same solution as in Fig. 9.12, but using Eq. (9-17) with a single structuring element.

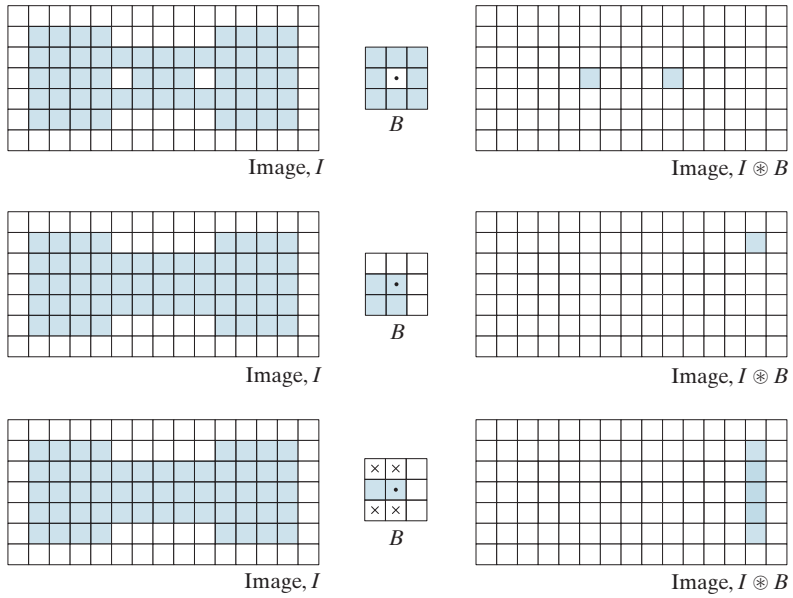
match. When the SE is centered on the bottom, right corner pixel, the role of the don't care elements is reversed, again resulting in a correct match. The other border pixels between the two corners were similarly detected by considering all don't care elements as foreground. Thus, using don't care elements increases the flexibility of structuring elements to perform multiple roles.

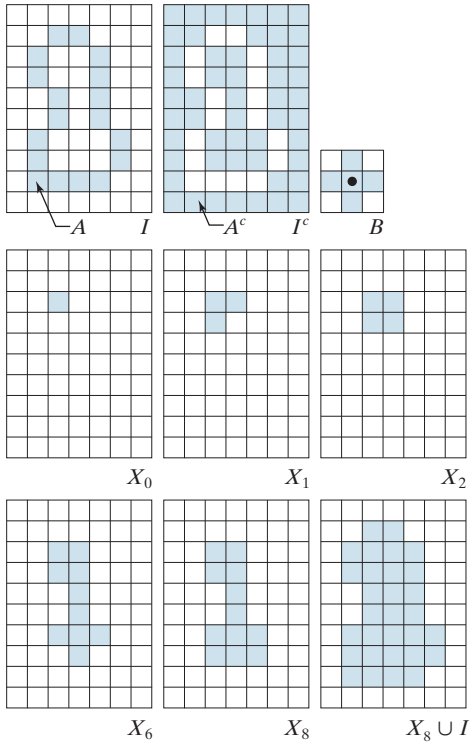
9.5 SOME BASIC MORPHOLOGICAL ALGORITHMS

With the preceding discussion as a foundation, we are now ready to consider some practical uses of morphology. When dealing with binary images, one of the principal applications of morphology is in extracting image components that are useful in the

a b c
d e f
g h i

FIGURE 9.14 Three examples of using a single structuring element and Eq. (9-17) to detect specific features. First row: detection of single-pixel holes. Second row: detection of an upper-right corner. Third row: detection of multiple features.





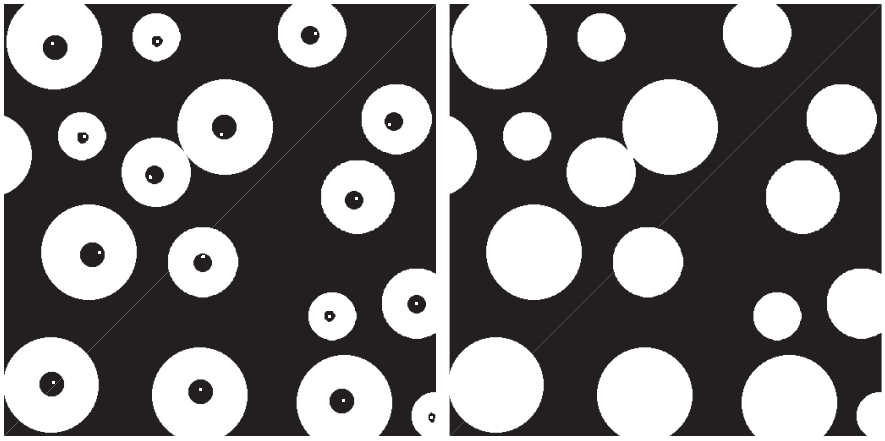
a b c
d e f
g h i

FIGURE 9.17
Hole filling. (a) Set A (shown shaded) contained in image I. (b) Complement of I. (c) Structuring element B. *Only* the foreground elements are used in computations (d) Initial point inside hole, set to 1. (e)–(h) Various steps of Eq. (9-19). (i) Final result [union of (a) and (h)].

EXTRACTION OF CONNECTED COMPONENTS

Connectivity and connected components are discussed in Section 2.5.

Being able to extract connected components from a binary image is central to many automated image analysis applications. Let A be a set of foreground pixels consisting of one or more connected components, and form an image X_0 (of the same size as I, the image containing A) whose elements are 0's (background values), except at each location known to correspond to a point in each connected component in A,



a b

FIGURE 9.18
(a) Binary image. The white dots inside the regions (shown enlarged for clarity) are the starting points for the hole-filling algorithm. (b) Result of filling all holes.

9.6 MORPHOLOGICAL RECONSTRUCTION

The morphological concepts discussed thus far involve a single image and one or more structuring elements. In this section, we discuss a powerful morphological transformation called *morphological reconstruction* that involves two images and a structuring element. One image, the *marker*, which we denote by F, contains the starting points for reconstruction. The other image, the *mask*, denoted by G, constrains (conditions) the reconstruction. The structuring element is used to define connectivity.[†] For 2-D applications, connectivity typically is defined as 8-connectivity, which is implied by a structuring element of size 3×3 whose elements are all 1's.

See Section 2.5 regarding connectivity.

GEODESIC DILATION AND EROSION

Central to morphological reconstruction are the concepts of geodesic dilation and geodesic erosion. Let F denote the marker image and G the mask image. We assume in this discussion that both are binary images and that $F \subseteq G$. The *geodesic dilation of size* 1 of the marker image with respect to the mask, denoted by $D_G^{(1)}(F)$, is defined as

$$D_G^{(1)}(F) = (F \oplus B) \cap G \tag{9-38}$$

where, as usual, $\cap$ denotes the set intersection (here $\cap$ may be interpreted as a logical AND because we are dealing with binary quantities). The *geodesic dilation of size n* of F with respect to G is defined as

$$D_G^{(n)}(F) = D_G^{(1)}\left(D_G^{(n-1)}(F)\right) \tag{9-39}$$

where $n \geq 1$ is an integer, and $D_G^{(0)}(F) = F$. In this recursive expression, the set intersection indicated in Eq. (9-38) is performed at each step.[‡] Note that the intersection operation guarantees that mask G will limit the growth (dilation) of marker F. Figure 9.28 shows a simple example of a geodesic dilation of size 1. The steps in the figure are a direct implementation of Eq. (9-38). Note that the marker F consists of just one point from the object in G. The idea is to grow (dilate) this point successively, masking of the result at each step by G. Continuing with this process would yield a result whose shape is influenced by the structure of G. In this simple case, the reconstruction would eventually result in an image identical to G (see Fig. 9.30).

The *geodesic erosion of size* 1 of marker F with respect to mask G is defined as

$$E_G^{(1)}(F) = (F \ominus B) \cup G \tag{9-40}$$

[†] In much of the literature on morphological reconstruction, the structuring element is tacitly assumed to be isotropic and typically is called an *elementary isotropic structuring element*. In the context of this chapter, an example of such an SE is a 3×3 array of 1's with the origin at the center.

[‡] Although it is more intuitive to develop morphological reconstruction methods using recursive formulations (as we do here), their practical implementation typically is based on more computationally efficient algorithms (see, for example, Vincent [1993] and Soille [2003]).

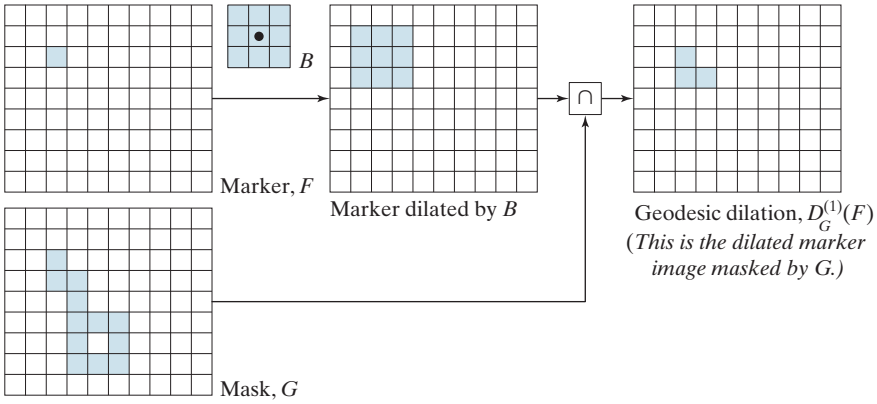


FIGURE 9.28 Illustration of a geodesic dilation of size 1. Note that the marker image contains a point from the object in G. If continued, subsequent dilations and maskings would eventually result in the object contained in G.

where $\cup$ denotes set union (or logical OR operation). The geodesic erosion of size n of F with respect to G is defined as

$$E_G^{(n)}(F) = E_G^{(1)}\left(E_G^{(n-1)}(F)\right) \tag{9-41}$$

where $n \geq 1$ is an integer and $E_G^{(0)}(F) = F$. The set union in Eq. (9-40) is performed at each step, and guarantees that geodesic erosion of an image remains greater than or equal to its mask image. As you might have expected from the forms in Eqs. (9-38) and (9-40), geodesic dilation and erosion are duals with respect to set complementation (see Problem 9.42). Figure 9.29 shows an example of a geodesic erosion of size 1. The steps in the figure are a direct implementation of Eq. (9-40).

Geodesic dilation and erosion converge after a finite number of iterative steps, because propagation or shrinking of the marker image is constrained by the mask.

MORPHOLOGICAL RECONSTRUCTION BY DILATION AND BY EROSION

Based on the preceding concepts, *morphological reconstruction by dilation* of a marker image F with respect to a mask image G, denoted $R_G^D(F)$, is defined as the geodesic dilation of F with respect to G, iterated until stability is achieved; that is,

$$R_G^D(F) = D_G^{(k)}(F) \tag{9-42}$$

with k such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

Figure 9.30 illustrates reconstruction by dilation. Figure 9.30(a) continues the process begun in Fig. 9.28. The next step in reconstruction after obtaining $D_G^{(1)}(F)$ is to dilate this result, then AND it with mask G to yield $D_G^{(2)}(F)$, as Fig. 9.30(b) shows. Dilation of $D_G^{(2)}(F)$ and masking with G then yields $D_G^{(3)}(F)$, and so on. This procedure is repeated until stability is reached. Carrying out this example one more step would give $D_G^{(5)}(F) = D_G^{(6)}(F)$, so the image, morphologically reconstructed by dilation, is given by $R_G^D(F) = D_G^{(5)}(F)$, as indicated in Eq. (9-42). The reconstructed image is identical to the mask, as expected.

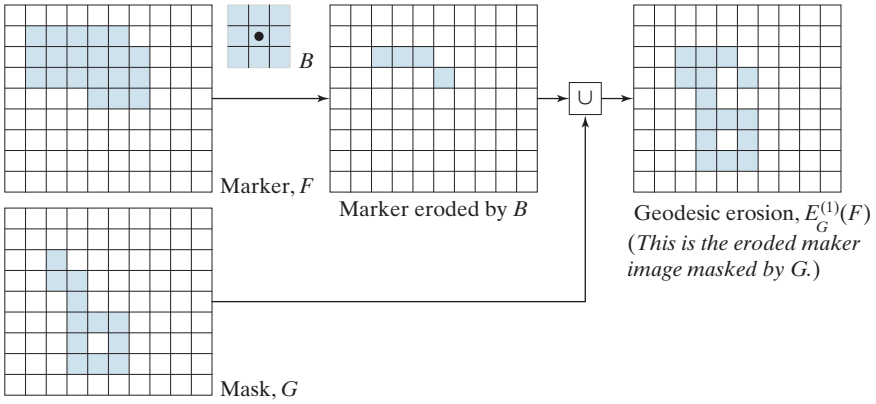


FIGURE 9.29 Illustration of a geodesic erosion of size 1.

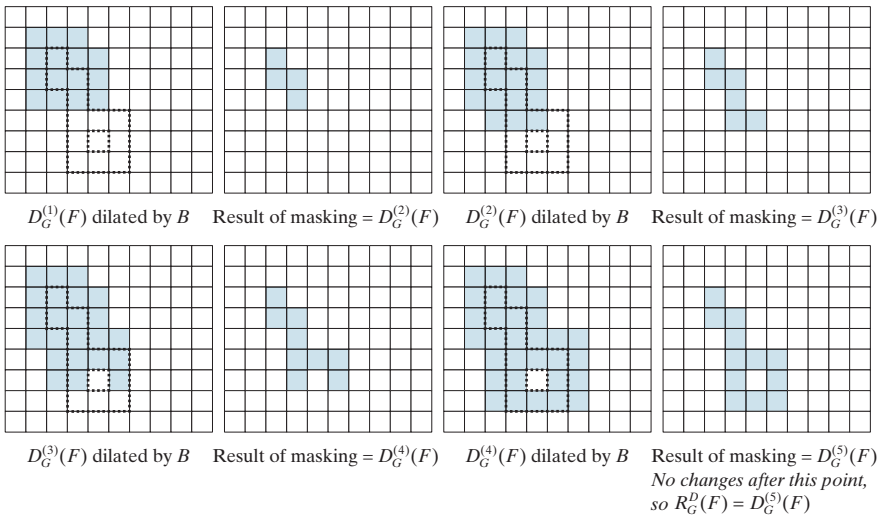
In a similar manner, the *morphological reconstruction by erosion* of a marker image F with respect to a mask image G, denoted $R_G^E(F)$, is defined as the geodesic erosion of F with respect to G, iterated until stability; that is,

$$R_G^E(F) = E_G^{(k)}(F) \tag{9-43}$$

with k such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$. As an exercise, generate a figure similar to Fig. 9.30 for morphological reconstruction by erosion. Reconstruction by dilation and erosion are duals with respect to set complementation (see Problem 9.43).

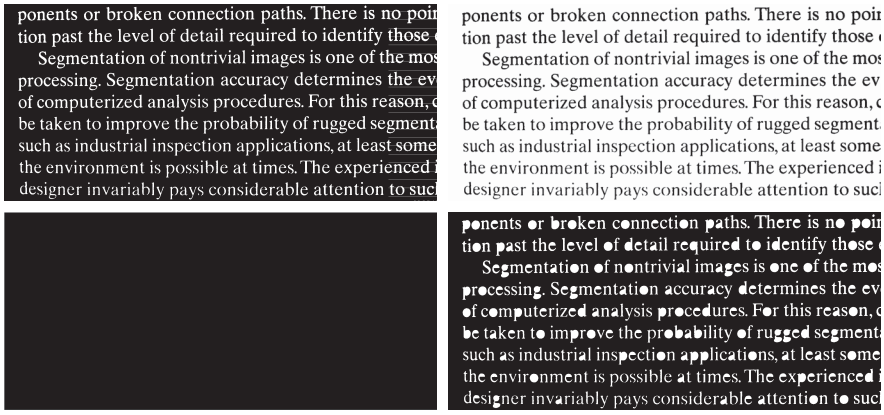
SAMPLE APPLICATIONS

Morphological reconstruction has a broad spectrum of practical applications, each determined by the selection of the marker and mask images, by the structuring



a b c d
e f g h

FIGURE 9.30 Illustration of morphological reconstruction by dilation. Sets $D_G^{(1)}(F)$, G, B and F are from Fig. 9.28. The mask (G) is shown dotted for reference.



a b
c d

FIGURE 9.33
(a) Text image of size 918×2018 pixels.
(b) Complement of (a) for use as a mask image.
(c) Marker image.
(d) Result of hole-filling using Eqs. (9-45) and (9-46).

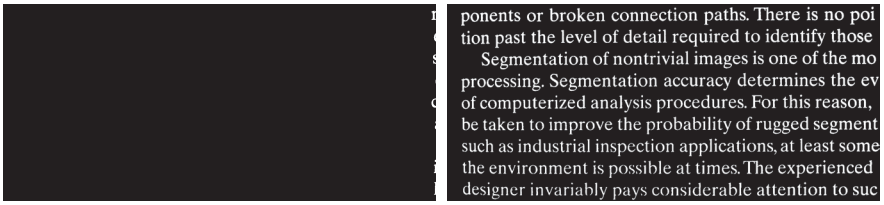
$$X = I - R_I^D(F) \tag{9-48}$$

to obtain an image, X, with no objects touching the border.

As an example, consider the original text image from Fig. 9.31(a) again. Figure 9.34(a) shows the reconstruction $R_I^D(F)$ obtained using a 3×3 structuring element of 1's. The objects touching the border of the original image are visible in the right side of Fig. 9.34(a). Figure 9.34(b) shows image X, computed using Eq. (9-48). If the task at hand were automated character recognition, having an image in which no characters touch the border is most useful because the problem of having to recognize partial characters (a difficult task at best) is avoided.

9.7 SUMMARY OF MORPHOLOGICAL OPERATIONS ON BINARY IMAGES

Figure 9.35 summarizes the types of structuring elements used in the various binary morphological methods discussed thus far. The shaded elements are foreground values (typically denoted by 1's in numerical arrays), the elements in white are background values (typically denoted by 0's), and the ×'s are "don't care" elements. Table 9.1 summarizes the binary morphological results developed in the preceding sections. The Roman numerals in the third column of Table 9.1 refer to the structuring elements in Fig. 9.35.



a b

FIGURE 9.34
(a) Reconstruction by dilation of marker image. (b) Image with no objects touching the border. The original image is Fig. 9.31(a).

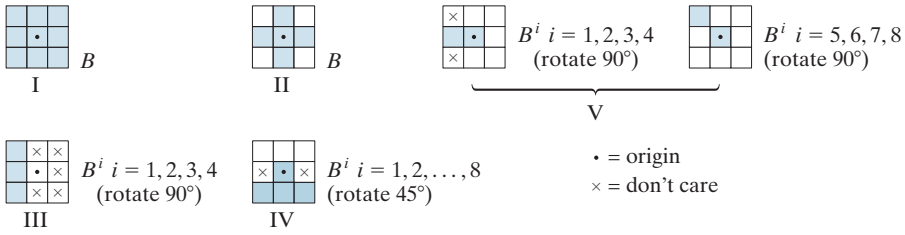


FIGURE 9.35 Five basic types of structuring elements used for binary morphology.

9.8 GRAYSCALE MORPHOLOGY

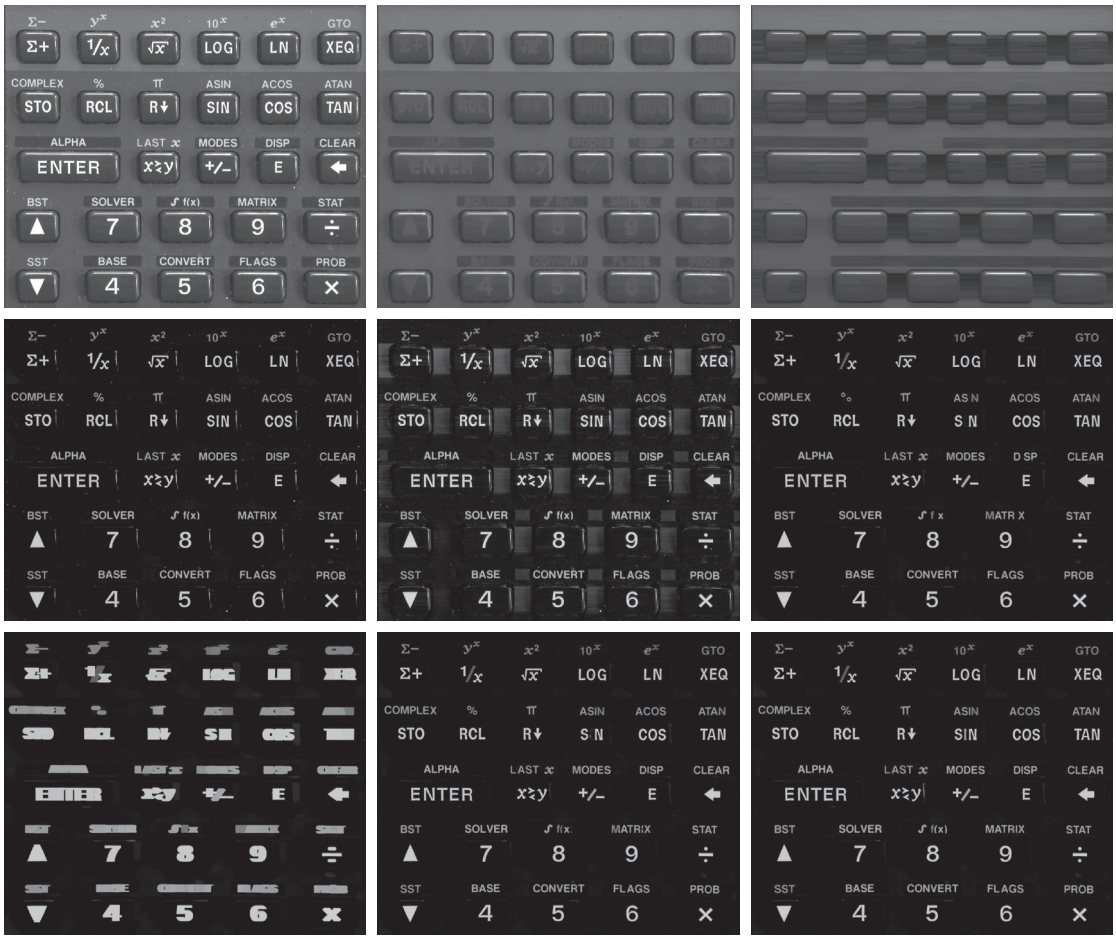
In this section, we extend to grayscale images the basic operations of dilation, erosion, opening, and closing. We then use these operations to develop several basic grayscale morphological algorithms. Throughout the discussion that follows, we deal with digital functions of the form $f(x, y)$ and $b(x, y)$, where $f(x, y)$ is a grayscale image and $b(x, y)$ is a structuring element. The assumption is that these functions are discrete in the sense defined in Section 2.4. That is, if Z denotes the set of real integers, then the coordinates (x, y) are integers from the Cartesian product Z^2, and $f(x, y)$ and $b(x, y)$ are functions that assign an intensity value (a real number from the set of real numbers, R) to each distinct pair of coordinates (x, y). If the intensity levels are integers also, then Z replaces R.

Structuring elements in grayscale morphology perform the same basic functions as their binary counterparts: They are used as "probes" to examine a given image for specific properties. Structuring elements in grayscale morphology belong to one of two categories: *nonflat* and *flat*. Figure 9.36 shows an example of each. Figure 9.36(a) is a hemispherical grayscale SE shown as an image, and Fig. 9.36(c) is a horizontal intensity profile through its center. Figure 9.34(b) shows a flat structuring element in the shape of a disk, and Fig. 9.36(d) is its corresponding intensity profile. (The shape of this profile explains the origin of the word "flat.") The elements in Fig. 9.36 are shown as continuous quantities for clarity; their computer implementation is based on digital approximations. Because of a number of difficulties discussed later in this section, grayscale nonflat SEs are not used frequently in practice. Finally, we mention that, as in the binary case, the origin of grayscale structuring elements must be clearly identified. Unless mentioned otherwise, all the examples in this section are based on symmetrical, flat structuring elements of unit height whose origins are at the center. The reflection of an SE in grayscale morphology is as defined in Section 9.1; we denote it in the following discussion by $\hat{b}(x, y) = b(-x, -y)$.

GRAYSCALE EROSION AND DILATION

The *grayscale erosion* of f by a flat structuring element b at location (x, y) is defined as the *minimum* value of the image in the region coincident with $b(x, y)$ when the origin of b is at (x, y). In equation form, the erosion at (x, y) of an image f by a structuring element b is given as

$$[f \ominus b](x, y) = \min_{(s, t) \in b} \{f(x + s, y + t)\} \tag{9-49}$$



a b c
d e f
g h i

FIGURE 9.46 (a) Original image of size 1134×1360 pixels. (b) Opening by reconstruction of (a), using a structuring element consisting of a horizontal line 71 pixels long in the erosion. (c) Opening of (a) using the same SE. (d) Top-hat by reconstruction. (e) Result of applying just a top-hat transformation. (f) Opening by reconstruction of (d), using a horizontal line 11 pixels long. (g) Dilation of (f) using a horizontal line 21 pixels long. (h) Minimum of (d) and (g). (i) Final reconstruction result. (Images courtesy of Dr. Steve Eddins, MathWorks, Inc.)

constant intensity. The solution of this problem is a good illustration of the power of grayscale morphology. We begin by suppressing the horizontal reflection on the top of the keys. The reflections are wider than any single character in the image, so we should be able to suppress them by performing an opening by reconstruction using a long horizontal line in the erosion operation. This operation will yield the background containing the keys and their reflections. Subtracting this from the original image (i.e., performing a top-hat by reconstruction) will eliminate the horizontal reflections and variations in background from the original image.

(*Hint:* Use proof by induction.)

(a)* $D_g^{(n)}(f) = \left[E_{g^c}^{(1)}[E_{g^c}^{(n-1)}(f^c)]\right]^c$. Assume a symmetric structuring element.

(b) $E_g^{(n)}(f) = \left[D_{g^c}^{(1)}[D_{g^c}^{(n-1)}(f^c)]\right]^c$. Assume a symmetric structuring element.

9.48 Prove the validity of the following grayscale morphological expressions.

(a)* $R_g^D(f) = \left[R_{g^c}^E\left(f^c\right)\right]^c$.

(b) $R_g^E(f) = \left[R_{g^c}^D\left(f^c\right)\right]^c$.

9.49 Prove the validity of the following grayscale morphological expressions.

(a)* $(f \ominus nb)^c = \left(f^c \oplus n\hat{b}\right)$, where $(f \ominus nb)$ indicates n successive erosions, starting with b.

(b) $(f \oplus nb)^c = \left(f^c \ominus n\hat{b}\right)$.

9.50 Prove the validity of the following grayscale morphological expressions. Recall that $f^c(x,y) = -f(x,y)$ and that $\hat{b}(x,y) = b(-x,-y)$. Assume a symmetric structuring element.

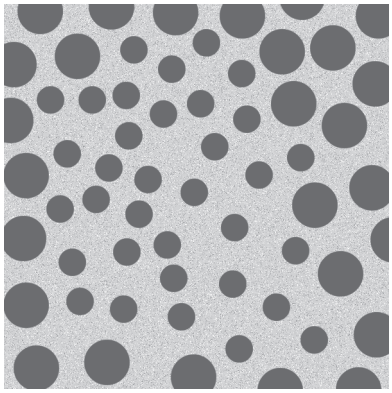
(a)* $O_R^{(n)}(f) = \left[C_R^{(n)}\left(f^c\right)\right]^c$.

(b) $C_R^{(n)}(f) = \left[O_R^{(n)}\left(f^c\right)\right]^c$.

9.51 Consider the image below, which shows a region of small circles enclosed by a region of larger circles.

(a) Would you expect the method used to generate Fig. 9.45(d) to work with this image also? Explain your reasoning, including any assumptions that you need to make for the method to work.

(b)* If your answer to (a) is yes, sketch what the boundary will look like.



9.52 A grayscale image, $f(x,y)$, is corrupted by nonoverlapping noise spikes that can be modeled as small flat disks of radii $R_{\min} \le R \le R_{\max}$ and amplitude $A_{\min} \le A \le A_{\max}$.

(a)* Develop a morphological filtering approach for denoising the image.

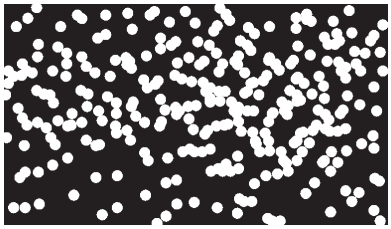
(b) Repeat (a), but now assume that there is touching and overlapping of, at most, four noise spikes appearing either as an array of 2-by-2 spikes, or 4-by-1 spikes.

9.53 A preprocessing step in an application of microscopy is concerned with the issue of isolating individual round particles from similar particles that overlap in groups of two or more particles (see the following image). Assuming that all particles are of the same size, propose a morphological algorithm that produces three images consisting respectively of:

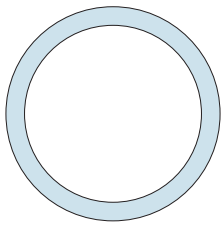
(a)* Only particles that have merged with the boundary of the image.

(b) Only overlapping particles.

(c) Only nonoverlapping particles.



9.54 A high-technology manufacturing plant is awarded a government contract to manufacture high-precision washers of the form shown:



The terms of the contract require that the shape of all washers be inspected by an imaging system. In this context, shape inspection refers to deviations from round on the inner and outer edges of

the washers. You may assume the following: (1) A "golden" (perfect with respect to the problem) image of an acceptable washer is available; and (2) the imaging and positioning components ultimately used in the system will have an accuracy high enough to allow you to ignore errors due to digitalization and positioning. You are hired as a consultant to help specify the visual inspection part of the system. Propose a solution based on morphological/logical operations.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

9.1* Numerous morphological functions are based on moving the center of a structuring element (SE) over an image I and, at each location (x, y), determining how well the elements of the SE match the pixels of the corresponding neighborhood of I centered at (x, y). This is similar to the mechanics of convolution and correlation discussed in Section 3.4 (see Fig. 3.34). Let **I** be a binary image of size **M**×**N** and **B** an SE of size **m**×**n** (**m** and **n** odd) whose origin is at its canter. The elements of **B** can be: 0, corresponding to the background of **I**; 1, corresponding to the foreground; or any other value (e.g., any integer other than 0 or 1) corresponding to "don't care" values. As in convolution and correlation, **I** must be padded. To accommodate all possible excursions of **B**, pad **I** with **m** rows of **padval** above and below and **n** columns to the left and right. The padding value can be 0 (the default) or 1. The padded image, **Ip**, will be of size (**M**+**2m**)×(**N**+**2n**).

(a) Write a function, **S = morphoMatch4e(I,B,padval, mode)** that finds all matches of **B** in **I**. Output **S** has elements with three possible values: 0, meaning no matches; 0.5, meaning partial matches; and 1 meaning a perfect match. Thus, a value of 1 at coordinates **(x, y)** in **S** means that the center of **B** was at **(x, y)** when **B** and the subimage of **Ip** directly under **B** were identical. In a partial match, at least one element of **B** matches a corresponding element in **Ip**. When **S** is 0 at **(x, y)**, no elements of **B** and the corresponding elements of the subimage were equal. Elements of **B** that have "don't care" values are always forced to match their corresponding elements in **Ip**. If **mode = 'full'**, **S** will be of the same size as **Ip**. If **mode = 'same'** (the default), **S** is cropped to the same size as **I**. If **mode** is included in the input argument, **padval** must be provided also.

You can implement this function in two basic ways. If you do not have the Image Processing Toolbox in your MATLAB installation, use **for** loops. If you do have the toolbox, you may *optionally* write the function using toolbox function **colfilt**, which implements sliding neighborhoods. The first approach is the simplest (but it generally is slower). The second approach is much more difficult, but it is faster and more elegant. We give solutions using both approaches. The solution using **colfilt** is called **morphoMatch4e**. The solution using loops is called **morphoMatchLoops4e**. If you implement only the loops solution, name it **morphoMatch4e** for use in later projects.

(b) Function **morphoMatch4e** is the foundation for most of the functions you will be writing in the following projects, so test it extensively with synthetic images of your choice. In your tests, make sure you use rectangular arrays (i.e., not square) for both **I** and **B**.

9.2 Erosion and dilation.

(a)* Write a function **E = morphoErode4e(I,B,padval)** for performing morphological erosion of binary image **I** by a structuring element **B**. The specifications for **I**, **B**, and **padval**, are the same as in Project 9.1, except that all elements of **B** should be 1. A value of **padval = 1** is used, for example, when eroding the complement of **I**. Because we assume that the background is by default 0, complementing **I** turns the background into 1, so the border has to be padded with 1's in such cases. (*Hint:* Use function **morphoMatch4e** from Project 9.1.)

10 Image Segmentation I
Edge Detection, Thresholding, and Region Detection

The whole is equal to the sum of its parts.
Euclid

The whole is greater than the sum of its parts.
Max Wertheimer

Preview

The material in the previous chapter began a transition from image processing methods whose inputs and outputs are images, to methods in which the inputs are images but the outputs are attributes extracted from those images. Most of the segmentation algorithms in this chapter are based on one of two basic properties of image intensity values: *discontinuity* and *similarity*. In the first category, the approach is to partition an image into regions based on abrupt changes in intensity, such as edges. Approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding, region growing, and region splitting and merging are examples of methods in this category. We show that improvements in segmentation performance can be achieved by combining methods from distinct categories, such as techniques in which edge detection is combined with thresholding. We discuss also image segmentation using clustering and superpixels, and give an introduction to graph cuts, an approach ideally suited for extracting the principal regions of an image. This is followed by a discussion of image segmentation based on morphology, an approach that combines several of the attributes of segmentation based on the techniques presented in the first part of the chapter. We conclude the chapter with a brief discussion on the use of motion cues for segmentation.

Upon completion of this chapter, readers should:

- Understand the characteristics of various types of edges found in practice.
- Understand how to use spatial filtering for edge detection.
- Be familiar with other types of edge detection methods that go beyond spatial filtering.
- Understand image thresholding using several different approaches.
- Know how to combine thresholding and spatial filtering to improve segmentation.
- Be familiar with region-based segmentation, including clustering and superpixels.
- Understand how graph cuts and morphological watersheds are used for segmentation.
- Be familiar with basic techniques for utilizing motion in image segmentation.

10.1 FUNDAMENTALS

Let R represent the entire spatial region occupied by an image. We may view image segmentation as a process that partitions R into n subregions, $R_1, R_2, \ldots, R_n$, such that

(a) $\bigcup_{i=1}^{n} R_i = R$.

(b) R_i is a connected set, for $i = 0, 1, 2, \ldots, n$.

(c) $R_i \cap R_j = \varnothing$ for all i and j, $i \neq j$.

(d) $Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \ldots, n$.

(e) $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j.

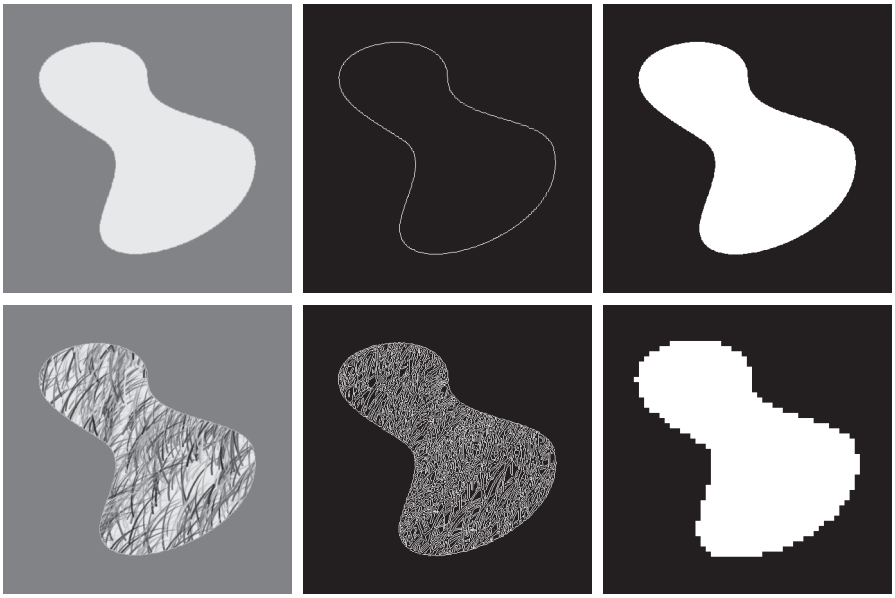
where $Q(R_k)$ is a logical predicate defined over the points in set R_k, and $\varnothing$ is the null set. The symbols $\cup$ and $\cap$ represent set union and intersection, respectively, as defined in Section 2.6. Two regions R_i and R_j are said to be *adjacent* if their union forms a connected set, as defined in Section 2.5. If the set formed by the union of two regions is not connected, the regions are said to *disjoint*.

Condition (a) indicates that the segmentation must be *complete*, in the sense that every pixel must be in a region. Condition (b) requires that points in a region be connected in some predefined sense (e.g., the points must be 8-connected). Condition (c) says that the regions must be disjoint. Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region—for example, $Q(R_i) = \text{TRUE}$ if all pixels in R_i have the same intensity. Finally, condition (e) indicates that two adjacent regions R_i and R_j must be different in the sense of predicate Q.[†]

Thus, we see that the fundamental problem in segmentation is to partition an image into regions that satisfy the preceding conditions. Segmentation algorithms for monochrome images generally are based on one of two basic categories dealing with properties of intensity values: *discontinuity* and *similarity*. In the first category, we assume that boundaries of regions are sufficiently different from each other, and from the background, to allow boundary detection based on local discontinuities in intensity. *Edge-based* segmentation is the principal approach used in this category. *Region-based* segmentation approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

Figure 10.1 illustrates the preceding concepts. Figure 10.1(a) shows an image of a region of constant intensity superimposed on a darker background, also of constant intensity. These two regions comprise the overall image. Figure 10.1(b) shows the result of computing the boundary of the inner region based on intensity discontinuities. Points on the inside and outside of the boundary are black (zero) because there are no discontinuities in intensity in those regions. To segment the image, we assign one level (say, white) to the pixels on or inside the boundary, and another level (e.g., black) to all points exterior to the boundary. Figure 10.1(c) shows the result of such a procedure. We see that conditions (a) through (c) stated at the beginning of this

[†] In general, Q can be a compound expression such as, "$Q(R_i) = \text{TRUE}$ if the average intensity of the pixels in region R_i is less than m_i AND if the standard deviation of their intensity is greater than σ_i," where m_i and σ_i are specified constants.



a b c
d e f

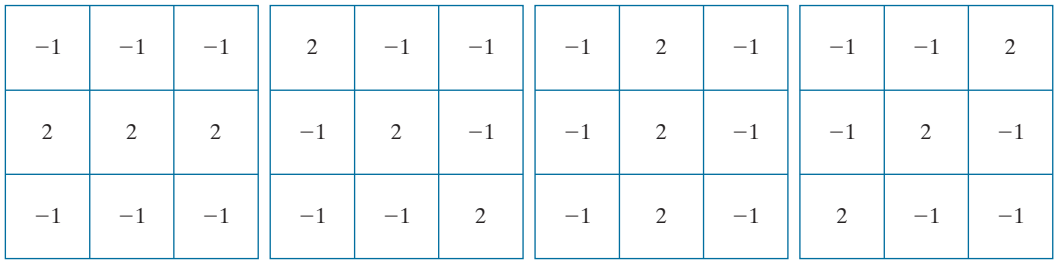
FIGURE 10.1
(a) Image of a constant intensity region. (b) Boundary based on intensity discontinuities. (c) Result of segmentation. (d) Image of a texture region. (e) Result of intensity discontinuity computations (note the large number of small edges). (f) Result of segmentation based on region properties.

section are satisfied by this result. The predicate of condition (d) is: If a pixel is on, or inside the boundary, label it white; otherwise, label it black. We see that this predicate is TRUE for the points labeled black or white in Fig. 10.1(c). Similarly, the two segmented regions (object and background) satisfy condition (e).

The next three images illustrate region-based segmentation. Figure 10.1(d) is similar to Fig. 10.1(a), but the intensities of the inner region form a textured pattern. Figure 10.1(e) shows the result of computing intensity discontinuities in this image. The numerous spurious changes in intensity make it difficult to identify a unique boundary for the original image because many of the nonzero intensity changes are connected to the boundary, so edge-based segmentation is not a suitable approach. However, we note that the outer region is constant, so all we need to solve this segmentation problem is a predicate that differentiates between textured and constant regions. The standard deviation of pixel values is a measure that accomplishes this because it is nonzero in areas of the texture region, and zero otherwise. Figure 10.1(f) shows the result of dividing the original image into subregions of size 8×8. Each subregion was then labeled white if the standard deviation of its pixels was positive (i.e., if the predicate was TRUE), and zero otherwise. The result has a "blocky" appearance around the edge of the region because groups of 8×8 squares were labeled with the same intensity (smaller squares would have given a smoother region boundary). Finally, note that these results also satisfy the five segmentation conditions stated at the beginning of this section.

10.2 POINT, LINE, AND EDGE DETECTION

The focus of this section is on segmentation methods that are based on detecting sharp, *local* changes in intensity. The three types of image characteristics in which



a b c d

FIGURE 10.6 Line detection kernels. Detection angles are with respect to the axis system in Fig. 2.19, with positive angles measured counterclockwise with respect to the (vertical) x-axis.

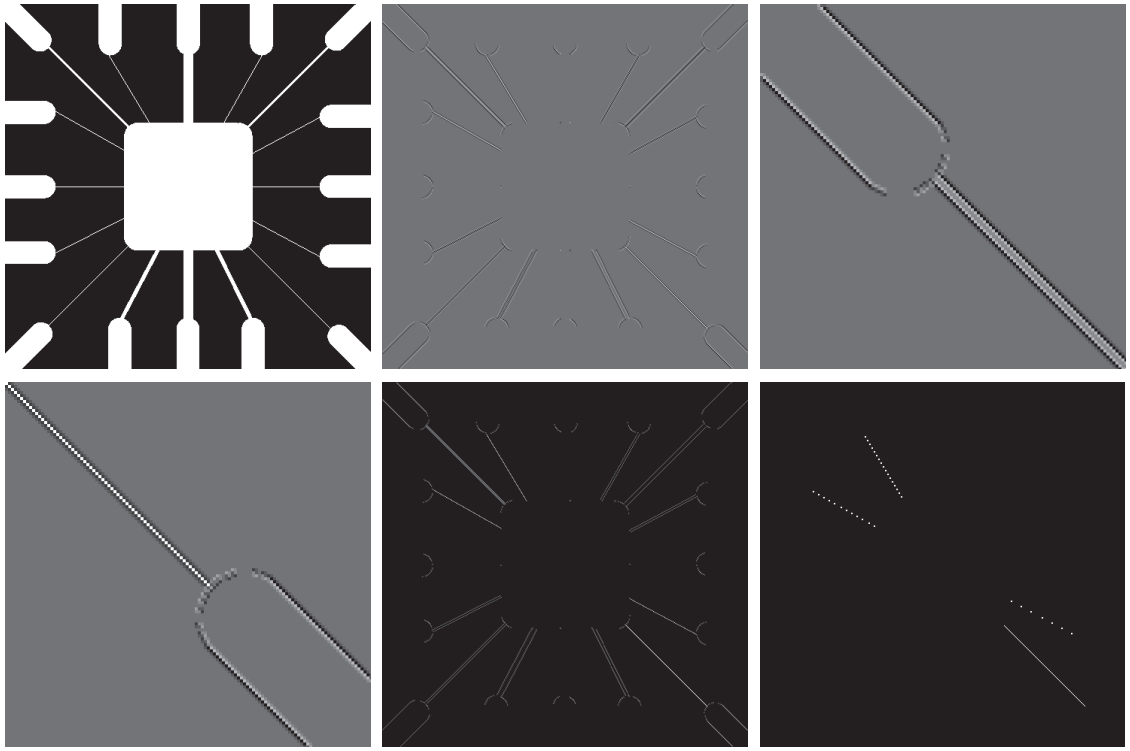
point is said to be more likely associated with a horizontal line. If we are interested in detecting all the lines in an image in the direction defined by a given kernel, we simply run the kernel through the image and threshold the absolute value of the result, as in Eq. (10-15). The nonzero points remaining after thresholding are the strongest responses which, for lines one pixel thick, correspond closest to the direction defined by the kernel. The following example illustrates this procedure.

EXAMPLE 10.3: Detecting lines in specified directions.

Figure 10.7(a) shows the image used in the previous example. Suppose that we are interested in finding all the lines that are one pixel thick and oriented at +45°. For this purpose, we use the kernel in Fig. 10.6(b). Figure 10.7(b) is the result of filtering the image with that kernel. As before, the shades darker than the gray background in Fig. 10.7(b) correspond to negative values. There are two principal segments in the image oriented in the +45° direction, one in the top left and one at the bottom right. Figures 10.7(c) and (d) show zoomed sections of Fig. 10.7(b) corresponding to these two areas. The straight line segment in Fig. 10.7(d) is brighter than the segment in Fig. 10.7(c) because the line segment in the bottom right of Fig. 10.7(a) is one pixel thick, while the one at the top left is not. The kernel is "tuned" to detect one-pixel-thick lines in the +45° direction, so we expect its response to be stronger when such lines are detected. Figure 10.7(e) shows the positive values of Fig. 10.7(b). Because we are interested in the strongest response, we let T equal 254 (the maximum value in Fig. 10.7(e) minus one). Figure 10.7(f) shows in white the points whose values satisfied the condition $g > T$, where g is the image in Fig. 10.7(e). The isolated points in the figure are points that also had similarly strong responses to the kernel. In the original image, these points and their immediate neighbors are oriented in such a way that the kernel produced a maximum response at those locations. These isolated points can be detected using the kernel in Fig. 10.4(a) and then deleted, or they can be deleted using morphological operators, as discussed in the last chapter.

EDGE MODELS

Edge detection is an approach used frequently for segmenting images based on abrupt (local) changes in intensity. We begin by introducing several ways to model edges and then discuss a number of approaches for edge detection.



a b c
d e f

FIGURE 10.7 (a) Image of a wire-bond template. (b) Result of processing with the +45° line detector kernel in Fig. 10.6. (c) Zoomed view of the top left region of (b). (d) Zoomed view of the bottom right region of (b). (e) The image in (b) with all negative values set to zero. (f) All points (in white) whose values satisfied the condition $g > T$, where g is the image in (e) and $T = 254$ (the maximum pixel value in the image minus 1). (The points in (f) were enlarged to make them easier to see.)

Edge models are classified according to their intensity profiles. A *step edge* is characterized by a transition between two intensity levels occurring ideally over the distance of one pixel. Figure 10.8(a) shows a section of a vertical step edge and a horizontal intensity profile through the edge. Step edges occur, for example, in images generated by a computer for use in areas such as solid modeling and animation. These clean, *ideal* edges can occur over the distance of one pixel, provided that no additional processing (such as smoothing) is used to make them look "real." Digital step edges are used frequently as edge models in algorithm development. For example, the Canny edge detection algorithm discussed later in this section was derived originally using a step-edge model.

In practice, digital images have edges that are blurred and noisy, with the degree of blurring determined principally by limitations in the focusing mechanism (e.g., lenses in the case of optical images), and the noise level determined principally by the electronic components of the imaging system. In such situations, edges are more

a b c

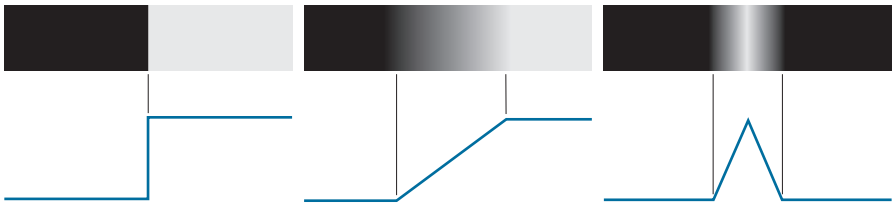


FIGURE 10.8
From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

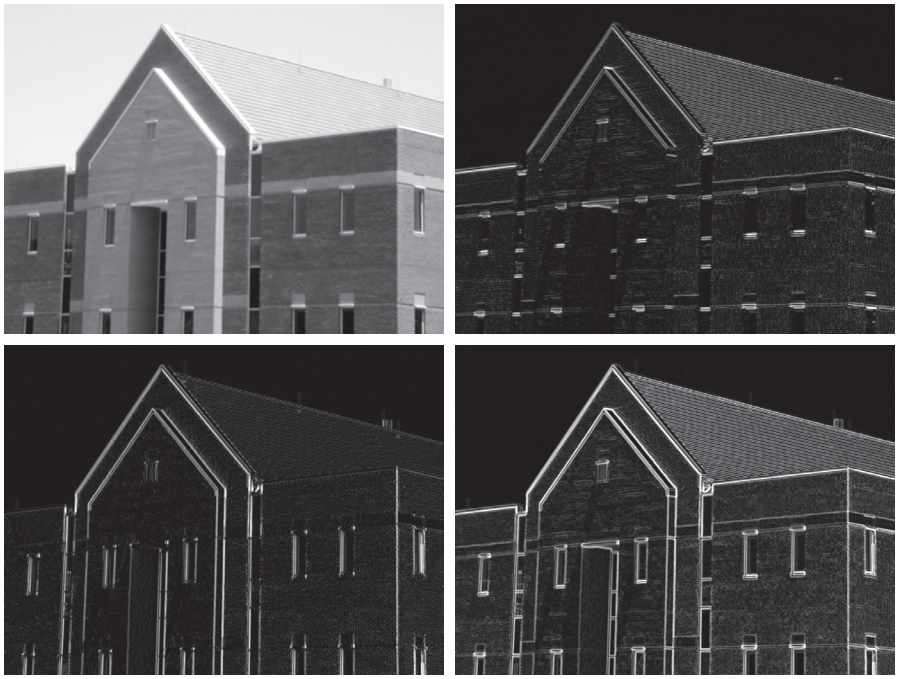
closely modeled as having an intensity *ramp* profile, such as the edge in Fig. 10.8(b). The slope of the ramp is inversely proportional to the degree to which the edge is blurred. In this model, we no longer have a single "edge point" along the profile. Instead, an edge point now is any point contained in the ramp, and an edge segment would then be a set of such points that are connected.

A third type of edge is the so-called *roof edge*, having the characteristics illustrated in Fig. 10.8(c). Roof edges are models of lines through a region, with the base (width) of the edge being determined by the thickness and sharpness of the line. In the limit, when its base is one pixel wide, a roof edge is nothing more than a one-pixel-thick line running through a region in an image. Roof edges arise, for example, in range imaging, when thin objects (such as pipes) are closer to the sensor than the background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 10.8(c). Other areas in which roof edges appear routinely are in the digitization of line drawings and also in satellite images, where thin features, such as roads, can be modeled by this type of edge.

It is not unusual to find images that contain all three types of edges. Although blurring and noise result in deviations from the ideal shapes, edges in images that are reasonably sharp and have a moderate amount of noise do resemble the characteristics of the edge models in Fig. 10.8, as the profiles in Fig. 10.9 illustrate. What the models in Fig. 10.8 allow us to do is write mathematical expressions for edges in the development of image processing algorithms. The performance of these algorithms will depend on the differences between actual edges and the models used in developing the algorithms.

Figure 10.10(a) shows the image from which the segment in Fig. 10.8(b) was extracted. Figure 10.10(b) shows a horizontal intensity profile. This figure shows also the first and second derivatives of the intensity profile. Moving from left to right along the intensity profile, we note that the first derivative is positive at the onset of the ramp and at points on the ramp, and it is zero in areas of constant intensity. The second derivative is positive at the beginning of the ramp, negative at the end of the ramp, zero at points on the ramp, and zero at points of constant intensity. The signs of the derivatives just discussed would be reversed for an edge that transitions from light to dark. The intersection between the zero intensity axis and a line extending between the extrema of the second derivative marks a point called the *zero crossing* of the second derivative.

We conclude from these observations that the *magnitude* of the first derivative can be used to detect the presence of an edge at a point in an image. Similarly, the *sign* of the second derivative can be used to determine whether an edge pixel lies on

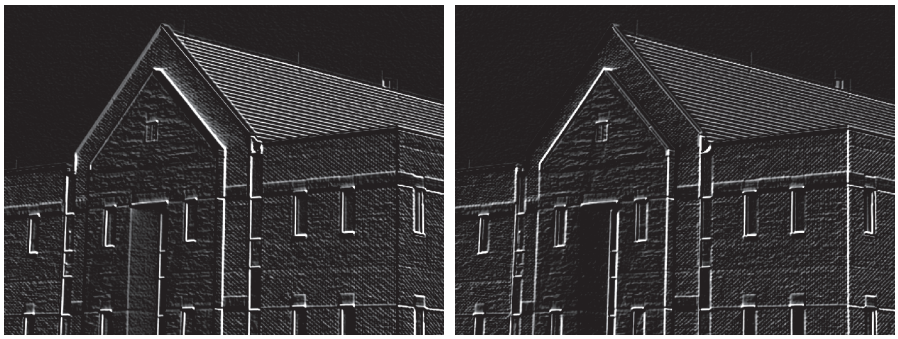


a b
c d

FIGURE 10.18
Same sequence as in Fig. 10.16, but with the original image smoothed using a 5×5 averaging kernel prior to edge detection.

black. Comparing this image with Fig. 10.16(d), we see that there are fewer edges in the thresholded image, and that the edges in this image are much sharper (see, for example, the edges in the roof tile). On the other hand, numerous edges, such as the sloping line defining the far edge of the roof (see arrow), are broken in the thresholded image.

When interest lies both in highlighting the principal edges and on maintaining as much connectivity as possible, it is common practice to use both smoothing and thresholding. Figure 10.20(b) shows the result of thresholding Fig. 10.18(d), which is the gradient of the smoothed image. This result shows a reduced number of broken edges; for instance, compare the corresponding edges identified by the arrows in Figs. 10.20(a) and (b).



a b

FIGURE 10.19
Diagonal edge detection.
(a) Result of using the Kirsch kernel in Fig. 10.15(c).
(b) Result of using the kernel in Fig. 10.15(d). The input image in both cases was Fig. 10.18(a).

a b

FIGURE 10.20
(a) Result of thresholding Fig. 10.16(d), the gradient of the original image. (b) Result of thresholding Fig. 10.18(d), the gradient of the smoothed image.

MORE ADVANCED TECHNIQUES FOR EDGE DETECTION

The edge-detection methods discussed in the previous subsections are based on filtering an image with one or more kernels, with no provisions made for edge characteristics and noise content. In this section, we discuss more advanced techniques that attempt to improve on simple edge-detection methods by taking into account factors such as image noise and the nature of edges themselves.

The Marr-Hildreth Edge Detector

One of the earliest successful attempts at incorporating more sophisticated analysis into the edge-finding process is attributed to Marr and Hildreth [1980]. Edge-detection methods in use at the time were based on small operators, such as the Sobel kernels discussed earlier. Marr and Hildreth argued (1) that intensity changes are not independent of image scale, implying that their detection requires using operators of different sizes; and (2) that a sudden intensity change will give rise to a peak or trough in the first derivative or, equivalently, to a zero crossing in the second derivative (as we saw in Fig. 10.10).

These ideas suggest that an operator used for edge detection should have two salient features. First and foremost, it should be a differential operator capable of computing a digital approximation of the first or second derivative at every point in the image. Second, it should be capable of being "tuned" to act at any desired scale, so that large operators can be used to detect blurry edges and small operators to detect sharply focused fine detail.

Marr and Hildreth suggested that the most satisfactory operator fulfilling these conditions is the filter $\nabla^2 G$ where, as defined in Section 3.6, ∇^2 is the Laplacian, and G is the 2-D Gaussian function

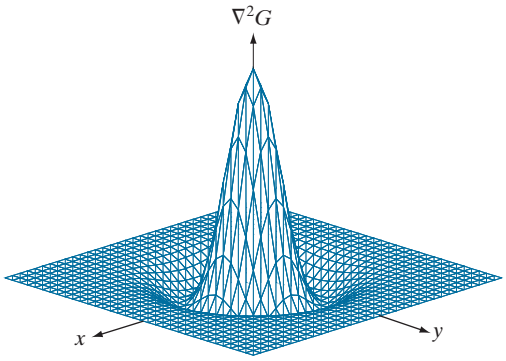
Equation (10-27) differs from the definition of a Gaussian function by a multiplicative constant [see Eq. (3-54)]. Here, we are interested only in the general shape of the Gaussian function.

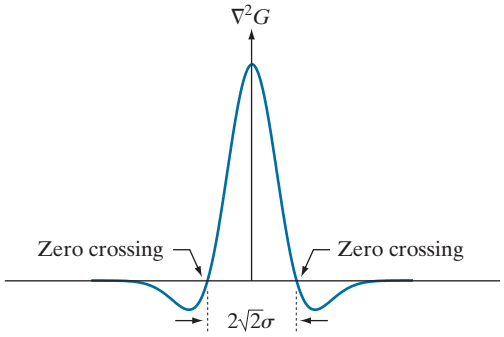
$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{10-27}$$

with standard deviation σ (sometimes σ is called the *space constant* in this context). We find an expression for $\nabla^2 G$ by applying the Laplacian to Eq. (10-27):

a b
c d

FIGURE 10.21
(a) 3-D plot of the *negative* of the LoG.
(b) Negative of the LoG displayed as an image.
(c) Cross section of (a) showing zero crossings.
(d) 5×5 kernel approximation to the shape in (a). The negative of this kernel would be used in practice.





0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

direction, thus avoiding having to use multiple kernels to calculate the strongest response at any point in the image.

The Marr-Hildreth algorithm consists of convolving the LoG kernel with an input image,

$$g(x,y) = \left[\nabla^2 G(x,y)\right] \star f(x,y) \tag{10-30}$$

This expression is implemented in the spatial domain using Eq. (3-44). It can be implemented also in the frequency domain using Eq. (4-104).

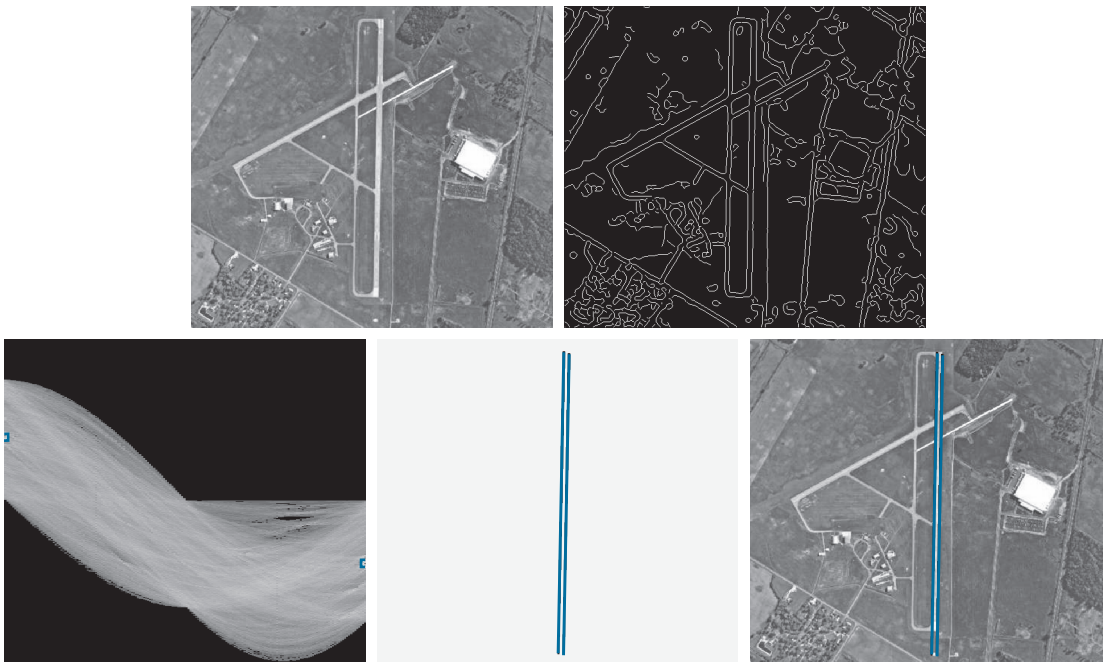
and then finding the zero crossings of $g(x,y)$ to determine the locations of edges in $f(x,y)$. Because the Laplacian and convolution are linear processes, we can write Eq. (10-30) as

$$g(x,y) = \nabla^2 \left[G(x,y) \star f(x,y)\right] \tag{10-31}$$

indicating that we can smooth the image first with a Gaussian filter and then compute the Laplacian of the result. These two equations give identical results.

The Marr-Hildreth edge-detection algorithm may be summarized as follows:

1. Filter the input image with an $n \times n$ Gaussian lowpass kernel obtained by sampling Eq. (10-27).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 kernel in Fig. 10.4(a). [Steps 1 and 2 implement Eq. (10-31).]
3. Find the zero crossings of the image from Step 2.



a b
c d e

FIGURE 10.31 (a) A 502×564 aerial image of an airport. (b) Edge map obtained using Canny's algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes. (e) Lines superimposed on the original image.

orientations of runways throughout the world are available in flight charts, and the direction of travel is easily obtainable using GPS (Global Positioning System) information. This information also could be used to compute the distance between the vehicle and the runway, thus allowing estimates of parameters such as expected length of lines relative to image size, as we did in this example.

10.3 THRESHOLDING

Because of its intuitive properties, simplicity of implementation, and computational speed, image thresholding enjoys a central position in applications of image segmentation. Thresholding was introduced in Section 3.1, and we have used it in various discussions since then. In this section, we discuss thresholding in a more formal way, and develop techniques that are considerably more general than what has been presented thus far.

FOUNDATION

In the previous section, regions were identified by first finding edge segments, then attempting to link the segments into boundaries. In this section, we discuss

techniques for partitioning images directly into regions based on intensity values and/or properties of these values.

The Basics of Intensity Thresholding

Suppose that the intensity histogram in Fig. 10.32(a) corresponds to an image, $f(x, y)$, composed of light objects on a dark background, in such a way that object and background pixels have intensity values grouped into two dominant modes. One obvious way to extract the objects from the background is to select a threshold, T, that separates these modes. Then, any point (x, y) in the image at which $f(x, y) > T$ is called an *object point*. Otherwise, the point is called a *background* point. In other words, the segmented image, denoted by $g(x, y)$, is given by

Remember, $f(x, y)$ denotes the intensity of f at coordinates (x, y).

Although we follow convention in using 0 intensity for the background and 1 for object pixels, any two distinct values can be used in Eq. (10-46).

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \tag{10-46}$$

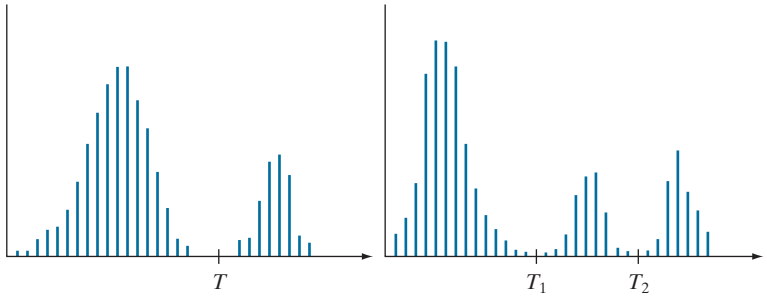
When T is a constant applicable over an entire image, the process given in this equation is referred to as *global thresholding*. When the value of T changes over an image, we use the term *variable thresholding*. The terms *local* or *regional* thresholding are used sometimes to denote variable thresholding in which the value of T at any point (x, y) in an image depends on properties of a neighborhood of (x, y) (for example, the average intensity of the pixels in the neighborhood). If T depends on the spatial coordinates (x, y) themselves, then variable thresholding is often referred to as *dynamic* or *adaptive* thresholding. Use of these terms is not universal.

Figure 10.32(b) shows a more difficult thresholding problem involving a histogram with three dominant modes corresponding, for example, to two types of light objects on a dark background. Here, *multiple thresholding* classifies a point (x, y) as belonging to the background if $f(x, y) \leq T_1$, to one object class if $T_1 < f(x, y) \leq T_2$, and to the other object class if $f(x, y) > T_2$. That is, the segmented image is given by

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases} \tag{10-47}$$

a b

FIGURE 10.32 Intensity histograms that can be partitioned (a) by a single threshold, and (b) by dual thresholds.

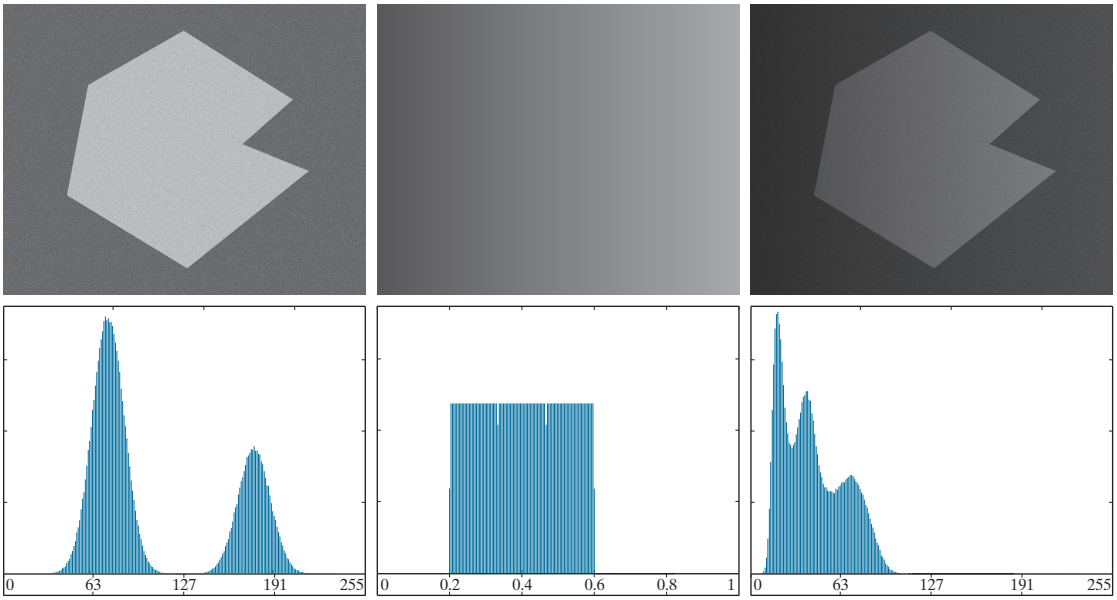


[see Fig. 10.33(e)], but their separation is enough so that the depth of the valley between them is sufficient to make the modes easy to separate. A threshold placed midway between the two peaks would do the job. Figure 10.33(c) shows the result of corrupting the image with Gaussian noise of zero mean and a standard deviation of 50 intensity levels. As the histogram in Fig. 10.33(f) shows, the situation is much more serious now, as there is no way to differentiate between the two modes. Without additional processing (such as the methods discussed later in this section) we have little hope of finding a suitable threshold for segmenting this image.

The Role of Illumination and Reflectance in Image Thresholding

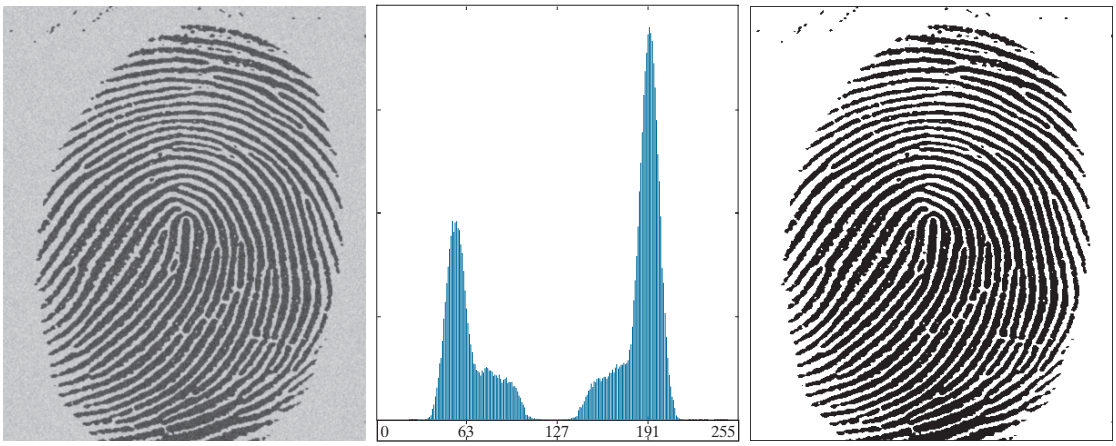
Figure 10.34 illustrates the effect that illumination can have on the histogram of an image. Figure 10.34(a) is the noisy image from Fig. 10.33(b), and Fig. 10.34(d) shows its histogram. As before, this image is easily segmentable with a single threshold. With reference to the image formation model discussed in Section 2.3, suppose that we multiply the image in Fig. 10.34(a) by a nonuniform intensity function, such as the intensity ramp in Fig. 10.37(b), whose histogram is shown in Fig. 10.34(e). Figure 10.34(c) shows the product of these two images, and Fig. 10.34(f) is the resulting histogram. The deep valley between peaks was corrupted to the point where separation of the modes without additional processing (to be discussed later in this section) is no longer possible. Similar results would be obtained if the illumination was

In theory, the histogram of a ramp image is uniform. In practice, the degree of uniformity depends on the size of the image and number of intensity levels.



a b c
d e f

FIGURE 10.34 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d) through (f) Corresponding histograms.



a b c

FIGURE 10.35 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (thin image border added for clarity). (Original image courtesy of the National Institute of Standards and Technology.).

initial choice for T). If this condition is met, the algorithm converges in a finite number of steps, whether or not the modes are separable (see Problem 10.32).

EXAMPLE 10.13: Global thresholding.

Figure 10.35 shows an example of segmentation using the preceding iterative algorithm. Figure 10.35(a) is the original image and Fig. 10.35(b) is the image histogram, showing a distinct valley. Application of the basic global algorithm resulted in the threshold $T = 125.4$ after three iterations, starting with T equal to the average intensity of the image, and using $\Delta T = 0$. Figure 10.35(c) shows the result obtained using $T = 125$ to segment the original image. As expected from the clear separation of modes in the histogram, the segmentation between object and background was perfect.

OPTIMUM GLOBAL THRESHOLDING USING OTSU'S METHOD

Thresholding may be viewed as a statistical-decision theory problem whose objective is to minimize the average error incurred in assigning pixels to two or more groups (also called *classes*). This problem is known to have an elegant closed-form solution known as the *Bayes decision function* (see Section 13.4). The solution is based on only two parameters: the probability density function (PDF) of the intensity levels of each class, and the probability that each class occurs in a given application. Unfortunately, estimating PDFs is not a trivial matter, so the problem usually is simplified by making workable assumptions about the form of the PDFs, such as assuming that they are Gaussian functions. Even with simplifications, the process of implementing solutions using these assumptions can be complex and not always well-suited for real-time applications.

The approach in the following discussion, called *Otsu's method* (Otsu [1979]), is an attractive alternative. The method is optimum in the sense that it maximizes the

between-class variance, a well-known measure used in statistical discriminant analysis. The basic idea is that properly thresholded classes should be distinct with respect to the intensity values of their pixels and, conversely, that a threshold giving the best separation between classes in terms of their intensity values would be the best (optimum) threshold. In addition to its optimality, Otsu's method has the important property that it is based entirely on computations performed on the histogram of an image, an easily obtainable 1-D array (see Section 3.3).

Let $\{0, 1, 2, \ldots, L-1\}$ denote the set of L distinct integer intensity levels in a digital image of size $M \times N$ pixels, and let n_i denote the number of pixels with intensity i. The total number, MN, of pixels in the image is $MN = n_0 + n_1 + n_2 + \cdots + n_{L-1}$. The normalized histogram (see Section 3.3) has components $p_i = n_i/MN$, from which it follows that

$$\sum_{i=0}^{L-1} p_i = 1 \qquad p_i \geq 0 \tag{10-48}$$

Now, suppose that we select a threshold $T(k) = k,\ 0 < k < L-1$, and use it to threshold the input image into two classes, c_1 and c_2, where c_1 consists of all the pixels in the image with intensity values in the range $[0, k]$ and c_2 consists of the pixels with values in the range $[k+1, L-1]$. Using this threshold, the probability, $P_1(k)$, that a pixel is assigned to (i.e., thresholded into) class c_1 is given by the cumulative sum

$$P_1(k) = \sum_{i=0}^{k} p_i \tag{10-49}$$

Viewed another way, this is the probability of class c_1 occurring. For example, if we set $k = 0$, the probability of class c_1 having any pixels assigned to it is zero. Similarly, the probability of class c_2 occurring is

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k) \tag{10-50}$$

From Eq. (3-36), the *mean intensity* value of the pixels in c_1 is

$$\begin{aligned} m_1(k) &= \sum_{i=0}^{k} iP(i/c_1) = \sum_{i=0}^{k} iP(c_1/i)P(i)/P(c_1) \\ &= \frac{1}{P_1(k)} \sum_{i=0}^{k} i p_i \end{aligned} \tag{10-51}$$

where $P_1(k)$ is given by Eq. (10-49). The term $P(i/c_1)$ in Eq. (10-51) is the probability of intensity value i, given that i comes from class c_1. The rightmost term in the first line of the equation follows from Bayes' formula (see Section 2.6):

$$P(A/B) = P(B/A)P(A)/P(B)$$

The second line follows from the fact that $P(c_1/i)$, the probability of c_1 given i, is 1 because we are dealing only with values of i from class c_1. Also, $P(i)$ is the probability of the ith value, which is the ith component of the histogram, p_i. Finally, $P(c_1)$ is the probability of class c_1 which, from Eq. (10-49), is equal to $P_1(k)$.

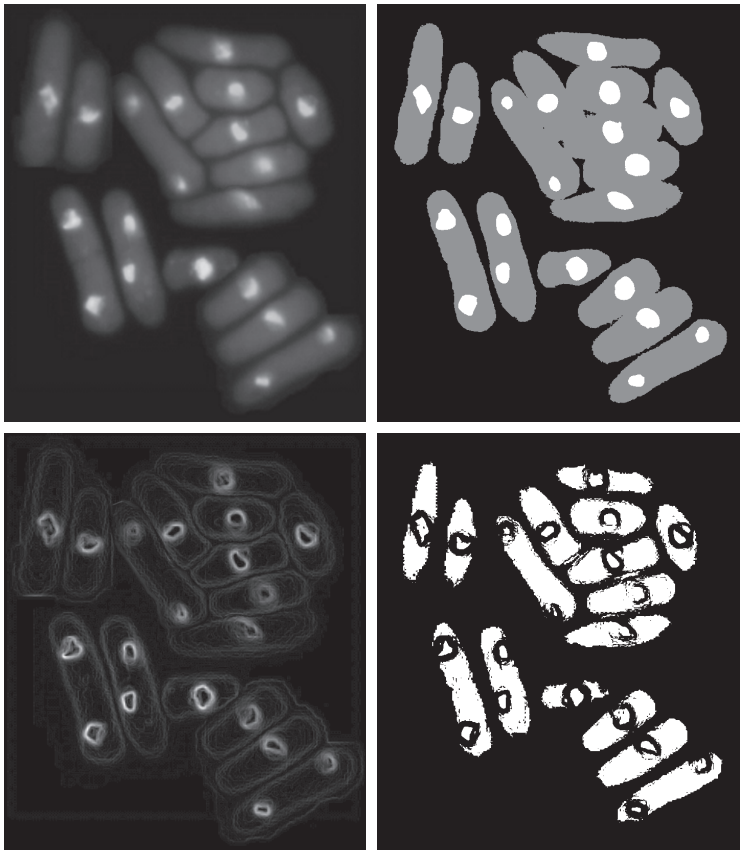
where Q is a *predicate* based on parameters computed using the pixels in neighborhood S_{xy}. For example, consider the following predicate, $Q(\sigma_{xy}, m_{xy})$, based on the local mean and standard deviation:

$$Q(\sigma_{xy}, m_{xy}) = \begin{cases} \text{TRUE} & \text{if } f(x,y) > a\sigma_{xy} \text{ AND } f(x,y) > bm_{xy} \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (10\text{-}82)$$

Note that Eq. (10-80) is a special case of Eq. (10-81), obtained by letting Q be TRUE if $f(x,y) > T_{xy}$ and FALSE otherwise. In this case, the predicate is based simply on the intensity at a point.

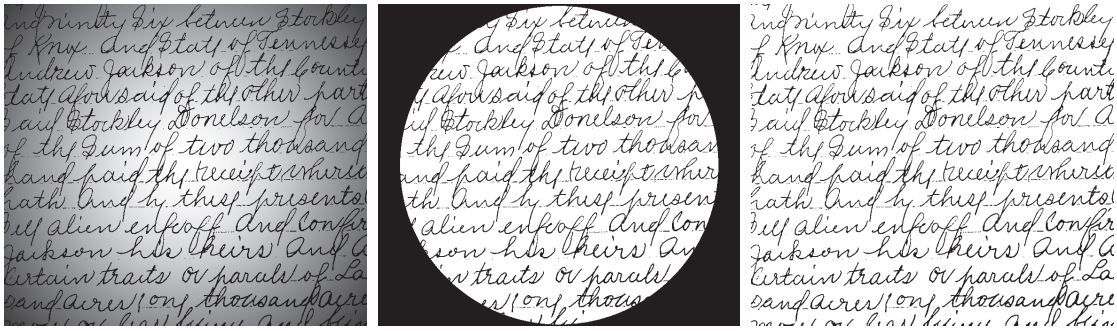
EXAMPLE 10.18: Variable thresholding based on local image properties.

Figure 10.43(a) shows the yeast image from Example 10.16. This image has three predominant intensity levels, so it is reasonable to assume that perhaps dual thresholding could be a good segmentation approach. Figure 10.43(b) is the result of using the dual thresholding method summarized in Eq. (10-76). As the figure shows, it was possible to isolate the bright areas from the background, but the mid-gray regions on the right side of the image were not segmented (i.e., separated) properly. To illustrate the use



a b
c d

FIGURE 10.43
(a) Image from Fig. 10.40. (b) Image segmented using the dual thresholding approach given by Eq. (10-76). (c) Image of local standard deviations. (d) Result obtained using local thresholding.



a b c

FIGURE 10.44 (a) Text image corrupted by spot shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages.

As another illustration of the effectiveness of this segmentation approach, we used the same parameters as in the previous paragraph to segment the image in Fig. 10.45(a), which is corrupted by a sinusoidal intensity variation typical of the variations that may occur when the power supply in a document scanner is not properly grounded. As Figs. 10.45(b) and (c) show, the segmentation results are comparable to those in Fig. 10.44.

Note that successful segmentation results were obtained in both cases using the same values for n and c, which shows the relative ruggedness of the approach. In general, thresholding based on moving averages works well when the objects of interest are small (or thin) with respect to the image size, a condition satisfied by images of typed or handwritten text.

10.4 SEGMENTATION BY REGION GROWING AND BY REGION SPLITTING AND MERGING

You should review the terminology introduced in Section 10.1 before proceeding.

As we discussed in Section 10.1, the objective of segmentation is to partition an image into regions. In Section 10.2, we approached this problem by attempting to find boundaries between regions based on discontinuities in intensity levels, whereas in Section 10.3, segmentation was accomplished via thresholds based on the distribution of pixel properties, such as intensity values or color. In this section and in Sections 10.5 and 10.6, we discuss segmentation techniques that find the regions directly. In Section 10.7, we will discuss a method that finds the regions and their boundaries simultaneously.

REGION GROWING

As its name implies, *region growing* is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach is to start with a set of "seed" points, and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed (such as ranges of intensity or color).

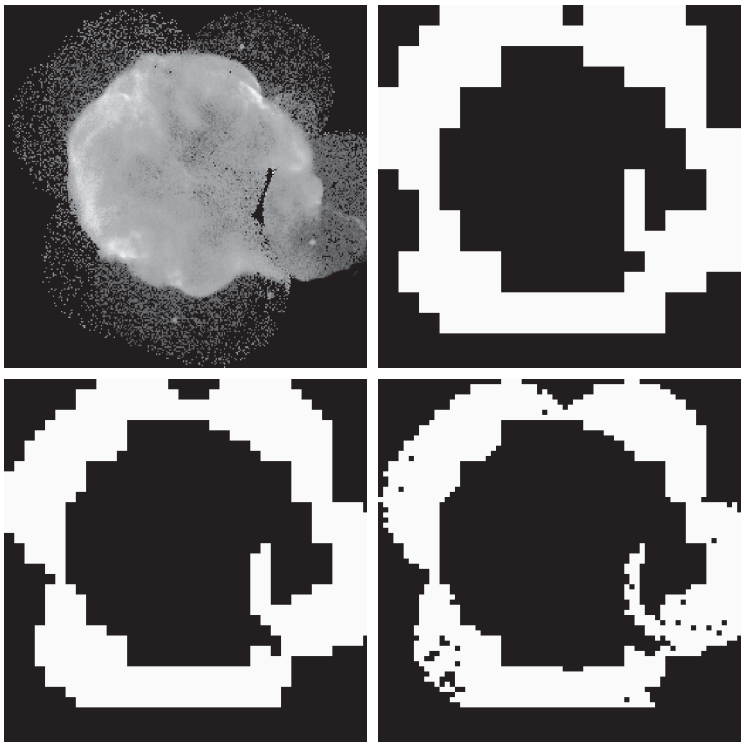
Selecting a set of one or more starting points can often be based on the nature of the problem, as we show later in Example 10.20. When a priori information is not

3. Stop when no further merging is possible.

Numerous variations of this basic theme are possible. For example, a significant simplification results if in Step 2 we allow merging of any two adjacent regions R_j and R_k if each one satisfies the predicate individually. This results in a much simpler (and faster) algorithm, because testing of the predicate is limited to individual quadregions. As the following example shows, this simplification is still capable of yielding good segmentation results.

EXAMPLE 10.21: Segmentation by region splitting and merging.

Figure 10.48(a) shows a 566×566 X-ray image of the Cygnus Loop supernova. The objective of this example is to segment (extract from the image) the "ring" of less dense matter surrounding the dense inner region. The region of interest has some obvious characteristics that should help in its segmentation. First, we note that the data in this region has a random nature, indicating that its standard deviation should be greater than the standard deviation of the background (which is near 0) and of the large central region, which is smooth. Similarly, the mean value (average intensity) of a region containing data from the outer ring should be greater than the mean of the darker background and less than the mean of the lighter central region. Thus, we should be able to segment the region of interest using the following predicate:



a b
c d

FIGURE 10.48
(a) Image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope. (b) through (d) Results of limiting the smallest allowed quadregion to be of sizes of 32×32, 16×16, and 8×8 pixels, respectively. (Original image courtesy of NASA.)

$$Q(R) = \begin{cases} \text{TRUE} & \text{if } \sigma_R > a \text{ AND } 0 < m_R < b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where σ_R and m_R are the standard deviation and mean of the region being processed, and a and b are nonnegative constants.

Analysis of several regions in the outer area of interest revealed that the mean intensity of pixels in those regions did not exceed 125, and the standard deviation was always greater than 10. Figures 10.48(b) through (d) show the results obtained using these values for a and b, and varying the minimum size allowed for the quadregions from 32 to 8. The pixels in a quadregion that satisfied the predicate were set to white; all others in that region were set to black. The best result in terms of capturing the shape of the outer region was obtained using quadregions of size 16×16. The small black squares in Fig. 10.48(d) are quadregions of size 8×8 whose pixels did not satisfy the predicate. Using smaller quadregions would result in increasing numbers of such black regions. Using regions larger than the one illustrated here would result in a more "block-like" segmentation. Note that in all cases the segmented region (white pixels) was a connected region that completely separates the inner, smoother region from the background. Thus, the segmentation effectively partitioned the image into three distinct areas that correspond to the three principal features in the image: background, a dense region, and a sparse region. Using any of the white regions in Fig. 10.48 as a mask would make it a relatively simple task to extract these regions from the original image (see Problem 10.45). As in Example 10.20, these results could not have been obtained using edge- or threshold-based segmentation.

As used in the preceding example, properties based on the mean and standard deviation of pixel intensities in a region attempt to quantify the texture of the region (see Section 12.3 for a discussion on texture). The concept of texture segmentation is based on using measures of texture in the predicates. In other words, we can perform texture segmentation by any of the methods discussed in this section simply by specifying predicates based on texture content.

10.5 REGION SEGMENTATION USING CLUSTERING AND SUPERPIXELS

In this section, we discuss two related approaches to region segmentation. The first is a classical approach based on seeking clusters in data, related to such variables as intensity and color. The second approach is significantly more modern, and is based on using clustering to extract "superpixels" from an image.

A more general form of clustering is *unsupervised clustering*, in which a clustering algorithm attempts to find a meaningful set of clusters in a given set of samples. We do not address this topic, as our focus in this brief introduction is only to illustrate how *supervised clustering* is used for image segmentation.

REGION SEGMENTATION USING K-MEANS CLUSTERING

The basic idea behind the clustering approach used in this chapter is to partition a set, Q, of observations into a specified number, k, of clusters. In k-means clustering, each observation is assigned to the cluster with the nearest mean (hence the name of the method), and each mean is called the *prototype* of its cluster. A *k-means algorithm* is an iterative procedure that successively refines the means until convergence is achieved.

Let $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_Q\}$ be set of vector observations (samples). These vectors have the form

a b c

FIGURE 10.51 (a) Original image. (b) Image composed of 40,000 superpixels. (c) Difference between (a) and (b).

some minor differences in areas around sharp edges. But remember, the superpixel image has an order of magnitude fewer elements than the original and, if needed, contrast differences are easily remedied by histogram processing.

As a final illustration, we show the results of severely decreasing the number of superpixels to 1,000, 500, and 250. The results in Fig. 10.52, show a significant loss of detail compared to Fig. 10.50(a), but the first two images contain most of the detail relevant to the image description discussed earlier. A notable difference is that two of the three small carvings on the fence in the back were eliminated. The 250-element superpixel image even lost the third. However, the boundaries between the principal regions, as well as the basic topology of the images, were preserved.

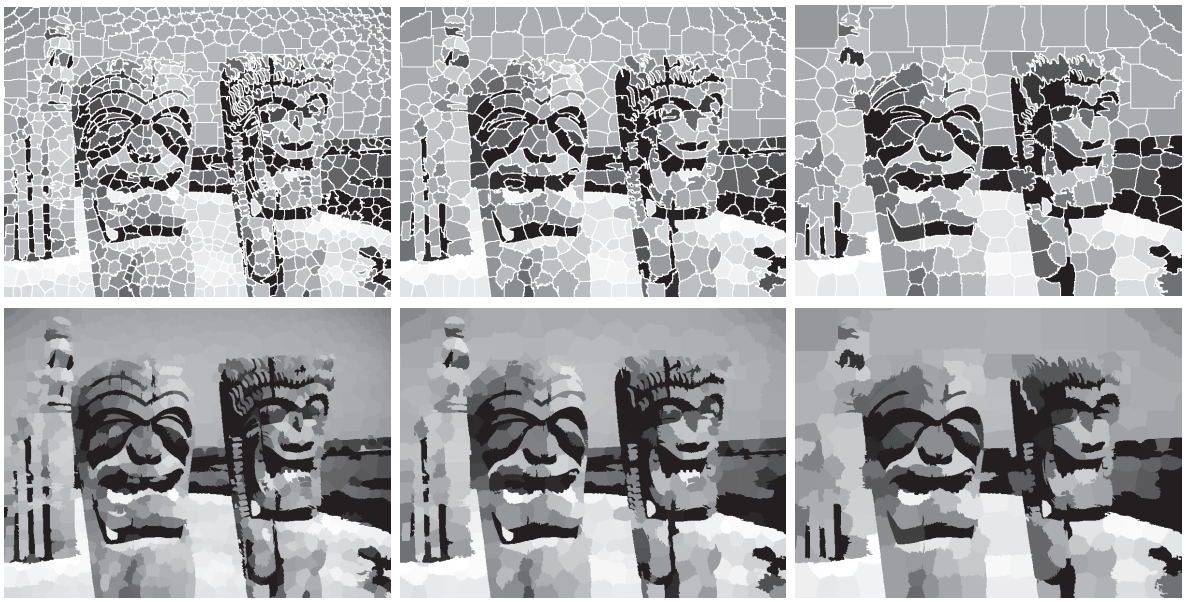
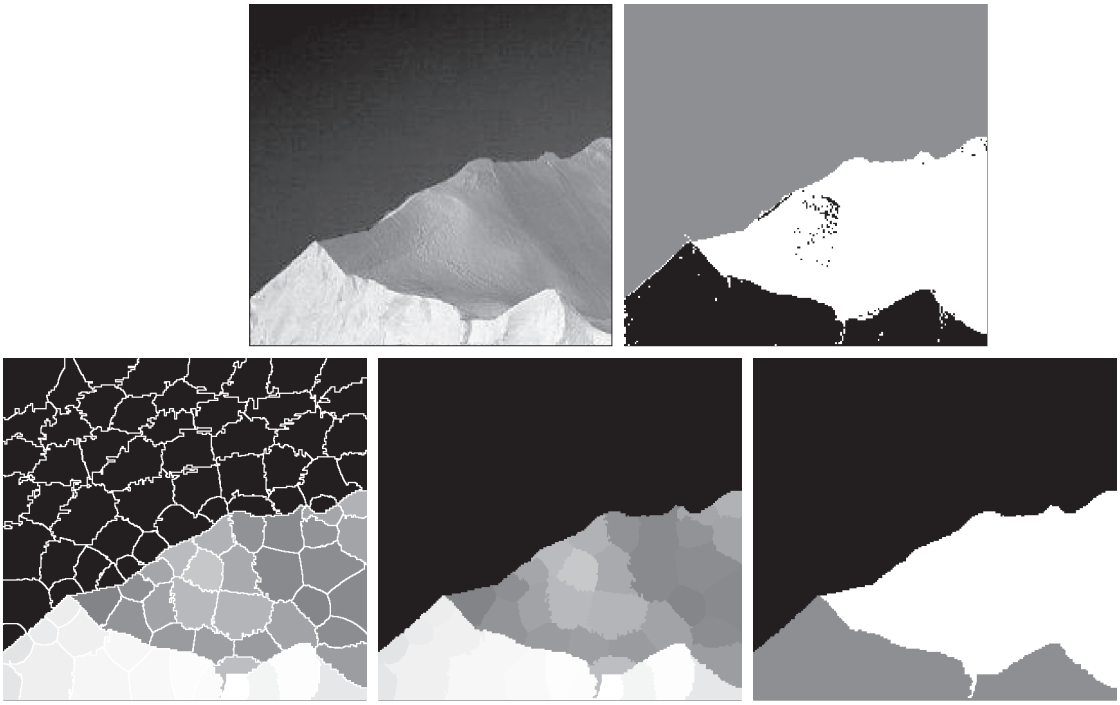


FIGURE 10.52 Top row: Results of using 1,000, 500, and 250 superpixels in the representation of Fig. 10.50(a). As before, the boundaries between superpixels are superimposed on the images for reference. Bottom row: Superpixel images.



a b
c d e

FIGURE 10.53 (a) Image of size 533×566 (301,678) pixels. (b) Image segmented using the k-means algorithm. (c) 100-element superpixel image showing boundaries for reference. (d) Same image without boundaries. (e) Superpixel image (d) segmented using the k-means algorithm. (Original image courtesy of NOAA.)

than is needed for a proper segmentation. In terms of computational advantage, consider that generating Fig. 10.53(b) required individual processing of over 300K pixels, while (e) required processing of 100 pixels with considerably fewer shades of gray.

Figure 10.54(a) is the same as Fig. 10.49(a), and Fig. 10.54(b) is a 95K-superpixel image (about 20% of the number of pixels in the original image). The original and the superpixel images are quite close visually. Although the number of superpixels is significantly smaller than the number of pixels in the original, they have basically the same content. For example, Fig. 10.54(c) is the result of segmenting Fig. 10.54(b) using the same k-means approach we used to generate Fig. 10.49(b), which is a segmentation of the original image. You can see by comparing Figs. 10.54(c) and 10.42(b) that the result using superpixels is superior. An added bonus is that the computational load of segmenting the superpixel image was significantly less.

10.6 REGION SEGMENTATION USING GRAPH CUTS

In this section, we discuss an approach for partitioning an image into regions by expressing the pixels of the image as nodes of a graph, and then finding an optimum partition (*cut*) of the graph into groups of nodes. Optimality is based on criteria whose

Superpixels are also well suited for use as graph nodes. Thus, when we refer in this section to "pixels" in an image, we are, by implication, also referring to superpixels.

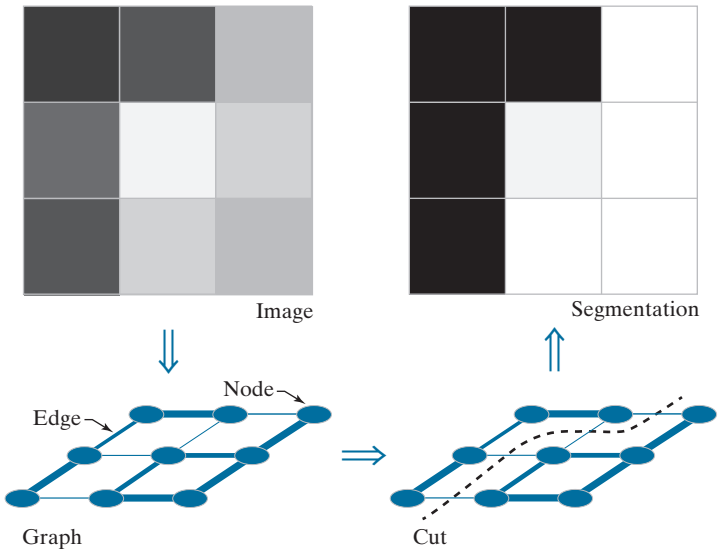
The essence of the material in this section is to represent an image to be segmented as a weighted, undirected graph, where the nodes of the graph are the pixels in the image, and an edge is formed between every pair of nodes. The weight, $w(i, j)$, of each edge is a function of the similarity between nodes i and j. We then seek to partition the nodes of the graph into disjoint subsets $V_1, V_2, \ldots, V_K$ where, by some measure, the similarity among the nodes within a subset is high, and the similarity across the nodes of different subsets is low. The nodes of the partitioned subsets correspond to the regions in the segmented image.

Set V is partitioned into subsets by cutting the graph. A *cut* of a graph is a partition of V into two subsets A and B such that

$$A \cup B = V \text{ and } A \cap B = \varnothing \tag{10-96}$$

where the cut is implemented by removing the edges connecting subgraphs A and B. There are two key aspects of using graph cuts for image segmentation: (1) how to associate a graph with an image; and (2) how to cut the graph in a way that makes sense in terms of partitioning the image into background and foreground (object) pixels. We address these two questions next.

Figure 10.55 shows a simplified approach for generating a graph from an image. The nodes of the graph correspond to the pixels in the image and, to keep the explanation simple, we allow edges only between adjacent pixels using 4-connectivity, which means that there are no diagonal edges linking the pixels. But, keep in mind that, in general, edges are specified between every pair of pixels. The weights for the edges typically are formed from spatial relationships (for example, distance from the vertex pixel) and intensity measures (for example, texture and color), consistent with exhibiting similarity between pixels. In this simple example, we define the degree of similarity between two pixels as the inverse of the difference in their intensities.



a b
c d

FIGURE 10.55
(a) A 3×3 image.
(c) A corresponding graph.
(d) Graph cut.
(c) Segmented image.

a b c

FIGURE 10.58 (a) Image of size 600×600 pixels. (b) Image smoothed with a 25×25 box kernel. (c) Graph cut segmentation obtained by specifying two regions.

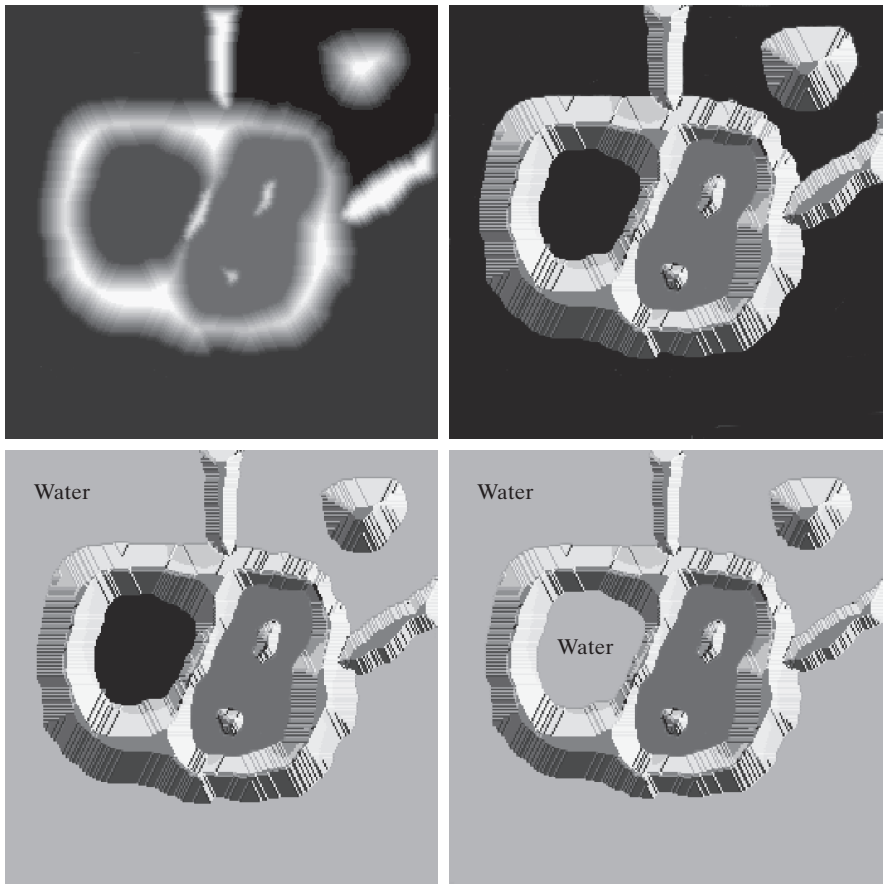
EXAMPLE 10.25: Segmentation using graph cuts.

Graph cuts are ideally suited for obtaining a rough segmentation of the principal regions in an image. Figure 10.58 shows a typical result. Figure 10.58(a) is the familiar building image. Consistent with the idea of extracting the principal regions of an image, Fig. 10.58(b) shows the image smoothed with a simple 25×25 box kernel. Observe how the fine detail is smoothed out, leaving only major regional features such as the facade and sky. Figure 10.58(c) is the result of segmentation using the graph cut algorithm just developed, with weights of the form discussed in the previous example, and allowing only two partitions. Note how well the region corresponding to the building was extracted, with none of the details characteristic of the methods discussed earlier in this chapter. In fact, it would have been nearly impossible to obtain comparable results using any of the methods we have discussed thus far without significant additional processing. This type of result is ideal for tasks such as providing broad cues for autonomous navigation, for searching image databases, and for low-level image analysis.

Figure 10.59 shows another example in which the image was smoothed with the same kernel. Here, we first specified two regions, resulting in Fig. 10.59(c). Note the fidelity of the separation between the iceberg and the background. Figure 10.59(d) is the result of specifying three regions in the segmentation. We see by comparing Figs. 10.59(d) and Fig. 10.42(c) that the graph-cut approach yielded a much more accurate segmentation, in the sense that none of the pixels within the regions were mislabeled as belonging to another region.

10.7 SEGMENTATION USING MORPHOLOGICAL WATERSHEDS

Thus far, we have discussed segmentation based on three principal concepts: edge detection, thresholding, and region extraction. Each of these approaches was found to have advantages (for example, speed in the case of global thresholding) and disadvantages (for example, the need for post-processing, such as edge linking, in edge-based segmentation). In this section, we discuss an approach based on the concept of so-called *morphological watersheds*. Segmentation by watersheds embodies many of the concepts of the other three approaches and, as such, often produces more stable



a c
b d

FIGURE 10.60
(a) Original image. (b) Topographic view. Only the background is *black*. The basin on the left is slightly lighter than black. (c) and (d) Two stages of flooding. All constant dark values of gray are intensities in the original image. Only constant *light gray* represents "water." (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.) (*Continued on next page*.)

from spilling out through the edges of the image, we imagine the perimeter of the entire topography (image) being enclosed by dams that are higher than the highest possible mountain, whose value is determined by the highest possible intensity value in the input image.

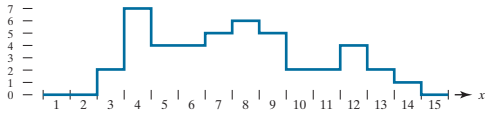
Suppose that a hole is punched in each regional minimum [shown as dark areas in Fig. 10.60(b)] and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. Figure 10.60(c) shows the first stage of flooding, where the "water," shown in light gray, has covered only areas that correspond to the black *background* in the image. In Figs. 10.60(d) and (e) we see that the water now has risen into the first and second catchment basins, respectively. As the water continues to rise, it will eventually overflow from one catchment basin into another. The first indication of this is shown in 10.60(f). Here, water from the lower part of the left basin overflowed into the basin on the right, and a short "dam" (consisting of single pixels) was built to prevent water from merging at that level of flooding (the mathematical details of dam building are discussed in the following section). The effect is more pronounced as water continues to rise, as shown in Fig. 10.60(g). This

Because of neighboring contrast, the leftmost basin in Fig. 10.60(c) appears black, but it is a few shades lighter than the black background. The mid-gray in the second basin is a natural gray from the image in (a).

$C_n(M_i)$ and $T[n]$ either increases or remains the same as n increases.

10.50 You saw in Section 10.7 that the boundaries obtained using the watershed segmentation algorithm form closed loops (for example, see Figs. 10.62 and 10.64). Advance an argument that establishes whether or not closed boundaries *always* result from application of this algorithm.

10.51* Give a step-by-step implementation of the dam-building procedure for the one-dimensional intensity cross section shown below. Show a drawing of the cross section at each step, showing "water" levels and dams constructed.



10.52 What would the negative ADI image shown in Fig. 10.65(c) look like if we tested against T (instead of testing against $-T$) in Eq. (10-117)?

10.53 Are the following statements true or false? Explain the reason for your answer in each.

(a)* The nonzero entries in the absolute ADI continue to grow in dimension, provided that the object is moving.

(b) The nonzero entries in the positive ADI always occupy the same area, regardless of the motion undergone by the object.

(c) The nonzero entries in the negative ADI continue to grow in dimension, provided that the object is moving.

10.54 Suppose that in Example 10.29 motion along the x-axis is set to zero. The object now moves only along the y-axis at 1 pixel per frame for 32 frames and then (instantaneously) reverses direction and moves in exactly the opposite direction for another 32 frames. What would Figs. 10.69(a) and (b) look like under these conditions?

10.55* Advance an argument that demonstrates that when the signs of S_{1x} and S_{2x} in Eqs. (10-125) and (10-126) are the same, velocity component V_1 is positive.

10.56 An automated pharmaceutical plant uses image processing to measure the shapes of medication tablets for the purpose of quality control. The segmentation stage of the system is based on Otsu's method. The speed of the inspection lines is so high that a very high rate flash illumination is required to "stop" motion. When new, the illumination lamps project a uniform pattern of light. However, as the lamps age, the illumination pattern deteriorates as a function of time and spatial coordinates according to the equation

$$i(x, y) = A(t) - t^2 e^{-[(x - M/2)^2 + (y - N/2)^2]}$$

where $(M/2, N/2)$ is the center of the viewing area and t is time measured in increments of months. The lamps are still experimental and the behavior of $A(t)$ is not fully understood by the manufacturer. All that is known is that, during the life of the lamps, $A(t)$ is always greater than the negative component in the preceding equation because illumination cannot be negative. It has been observed that Otsu's algorithm works well when the lamps are new, and their pattern of illumination is nearly constant over the entire image. However, segmentation performance deteriorates with time. Being experimental, the lamps are exceptionally expensive, so you are employed as a consultant to help solve the problem using digital image processing techniques to compensate for the changes in illumination, and thus extend the useful life of the lamps. You are given flexibility to install any special markers or other visual cues in the viewing area of the imaging cameras. Propose a solution in sufficient detail that the engineering plant manager can understand your approach. (*Hint:* Review the image model discussed in Section 2.3 and consider using one or more targets of known reflectivity.)

10.57 The speed of a bullet in flight is to be estimated by using high-speed imaging techniques. The method of choice involves the use of a CCD camera and flash that exposes the scene for K seconds. The bullet is 2.5 cm long, 1 cm wide, and its range of speed is 750 ± 250 m/s. The camera optics produce an image in which the bullet occupies 10% of the horizontal resolution of a 256×256 digital image.

(a)* Determine the maximum value of K that will guarantee that the blur from motion does not exceed 1 pixel.

(b) Determine the minimum number of frames

per second that would have to be acquired in order to guarantee that at least two complete images of the bullet are obtained during its path through the field of view of the camera.

(c)* Propose a segmentation procedure for automatically extracting the bullet from a sequence of frames.

(d) Propose a method for automatically determining the speed of the bullet.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

10.1 Edge models.

(a)* Write a function, **g = edgeModel4e(type,M,N,iLow, iHigh,width,angle)** for generating an image of size $M \times N$ containing an edge passing through the center of the image. Values of **type** define the edge as a **'step'**, **'ramp'**, or **'roof'** edge. Using a vertical edge (the default) as a reference, **iLow** denotes the intensity on the left side of the edge, and **iHigh** the intensity of its right side, with **iLow** < **iHigh**. Parameter **width** (in pixels) is the width of the transition of a ramp or roof edge; it is ignored for step edges. It is required that **width** be less than **N−2**. The default value for width is (**N/4**), but a value for **width** must be specified for all three edges if **angle** is specified (**width** is ignored for step edges). Parameter **angle** (in degrees) is the rotation of the edge about its center. The default is 0 for a vertical edge, and a positive value of angle results in counterclockwise rotation. The image generated by this function must fill the entire $M \times N$, rectangle. (*Hint:* Consider using project function **imageRotate4e** to perform the rotation.

(b)* Use **f = edgeModel4e('step',10,10,0,1)** to generate a small vertical binary step edge. Display the array. Apply both Sobel kernels (see Project 10.2) individually to the small image using function **twodConv4e** and display the two resulting numerical arrays. Explain why the results are different.

(c) Use function **edgeModel4e** to generate an edge as in (b), but oriented at 45°. Convolve this image with the north Kirsch kernel and display the resulting array. Repeat using the south Kirsch kernel. You will find that both kernels give nonzero results along the edge, but the results differ in the sign and location of the values. Explain what this means. Both results suggest the presence of a binary edge oriented at 45°, so what is the usefulness of having two kernels for 45° edge detection?

(d)* Test function **edgeModel4e** with more practical arrays by generating and displaying four 8-bit images of size 512×512 pixels with: a step edge oriented at 60°, a step edge oriented at −60°, a ramp edge with a ramp width of 128 pixels, oriented at −45°, and a roof edge of the same width oriented at 45°.

(e) Show how you would use function **edgeModel4e** to generate a black image containing eight white 1-pixel lines in the eight compass directions that intersect in the center of a 512×512 image.

10.2 Sobel, Prewitt, and Kirsch compass kernels.

(a) Write a function **w = edgeKernel4e(type,dir)** for generating the kernels in Figs. 10.14 and 10.15. Parameter **type** is one of three strings: **'prewitt'**, **'sobel'**, and **'kirsch'**. Parameter **dir** is **'v'** for the vertical edges in the Prewitt and Sobel kernels, and **'h'** for the horizontal edges in these two kernels. For the Kirsch kernels, **dir** is a detailed in Fig. 10.15.

(b)* Display all the kernels that your function is capable of generating.

10.3 Edge magnitude and angle.

(a)* Write a function **g = edgeMag4e(f,type,T)** for computing the magnitude of the gradient of image **f**. Parameter **'type'** designates the kernel used to compute the gradient: **'prewitt'**, **'sobel'**, and **'kirsch'**, corresponding to

11

Image Segmentation II Active Contours: Snakes and Level Sets

Divide each difficulty into as many parts as is feasible and necessary to resolve it.

Rene Descartes

Preview

In this chapter, we develop the foundation for image segmentation based on active contours, which are deformable models confined to the plane. We discuss two approaches: *snakes* and *level sets*. Snakes are active contours based on *explicit* (e.g., parametric) representation of segmentation curves; they derive their name from the way the curves appear to "slither" on the image plane in the process of seeking region boundaries. Level sets are based on *implicit* representation of curves, which are techniques for representing active contours as the intersection of a 3-D surface with a plane. The fundamental equations of both approaches are derived starting from basic principles. We give numerous examples designed to illustrate the strengths and limitations of both methods, and conclude the chapter with a brief discussion of a fast implementation approach for level sets.

Upon completion of this chapter, readers should:

- Have a command of the derivation of the snake and level set equations, starting from basic principles.
- Understand how to implement discrete, iterative formulations of both approaches.
- Be able to formulate explicit functions for use in snake segmentation.
- Understand how to formulate implicit functions for use in level set segmentation.
- Understand force fields and be able to apply them in active contour algorithms.
- Know how to generate level set functions, starting with a plane curve.
- Be aware of the central role played by the image gradient in both snake and level set force formulations.
- Be able to relate a variety of different level set approaches to the same fundamental iterative solution.
- Be familiar with the advantages and limitations of snakes and level sets as they apply to image processing.

11.1 BACKGROUND

With the exception of graph cuts and watershed segmentation, the material in the previous chapter dealt mostly with what we might call "traditional" segmentation methods, based primarily on detecting intensity discontinuities or similarities. In this chapter, we discuss techniques that approach segmentation from a "modeling" point of view. Specifically, we develop methods whose origin can be traced to work on deformable models conducted in the 1980s. *Deformable models* are physically based models of deformable curves, surfaces, and solids used traditionally in computer graphics. The topic of this chapter, *active contours* (also called *evolving fronts* or *evolving interfaces*), are deformable models confined to the plane. The term "active" indicates that the curves are *dynamic*, as opposed, for example, to segmentation curves resulting from a global thresholding operation. In segmentation, these active curves are attracted to region boundaries, acting under the influence of forces extracted typically from an image being segmented.

Work on active contours related specifically to image segmentation evolved along two different paths. One path was based on so-called *snakes*, introduced by Kass, Witkin, and Terzopoulus [1988]. Snakes are *parametric curves* that seek the boundary of a region by minimizing an energy functional, guided by internal forces, and influenced by image forces that pull it toward image features, such as lines and edges. The term "snake" is based on the appearance of a curve as it "slithers" on the image plane in the process of seeking its minimum energy.

A development parallel chronologically to snakes was based on *level sets*, introduced by Osher and Sethian [1988] as a tool in computational fluid dynamics for following fronts propagating through a medium. The key difference between the two approaches is that snakes are based on *explicit* representations of segmentation contours as parametric curves, while level sets rely on *implicit* representations of contours, expressed as the intersection of a 3-D surface with a plane. For example, the intersection of a sphere and plane is an implicit representation of a circle.

The body of work on active contours is impressive, both in breadth and depth, including countless articles and numerous books and monographs dealing with various aspects of the subject. Our focus in this chapter is on the foundation of both active contour approaches. In the next section, we will derive the fundamental snake equation, starting from basic principles. We then discuss various implementation details and give several application examples. We will follow a similar path in Section 11.3 by deriving the level set equation, and illustrating its implementation and use in image segmentation. These two equations are the foundation for most of the active-contour approaches you are likely to encounter in image processing.

11.2 IMAGE SEGMENTATION USING SNAKES

Snakes are parametric representations of active contours, so begin the discussion by reviewing some basic concepts of parametric curve representation.

EXPLICIT (PARAMETRIC) REPRESENTATION OF ACTIVE CONTOURS

A *parametric curve* in the xy-plane is defined by coordinates expressed as

$$(x, y) = \big(g(s), h(s)\big) \tag{11-1}$$

Solving this equation for $\mathbf{x}(t)$ and $\mathbf{y}(t)$ yields the iterative solution of the snake equation:

$$\begin{aligned}\mathbf{x}(t) &= \left[\mathbf{I} - \Delta t\mathbf{D}\right]^{-1}\left[\mathbf{x}(t-1) + \Delta t\mathbf{F}_x\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right] \\ \mathbf{y}(t) &= \left[\mathbf{I} - \Delta t\mathbf{D}\right]^{-1}\left[\mathbf{y}(t-1) + \Delta t\mathbf{F}_y\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right]\end{aligned} \tag{11-43}$$

where $\mathbf{I}$ is the $K \times K$ identity matrix. The Δt constant multiplies all derivative and force terms in this equation, so it has little selective influence on the internal versus the external forces. A more selective formulation is obtained by letting $\Delta t = 1$ and multiplying the external force components by a constant γ. This allows control of the internal forces by α and β, and the external force by γ. Equation (11-43) then becomes

$$\begin{aligned}\mathbf{x}(t) &= \left[\mathbf{I} - \mathbf{D}\right]^{-1}\left[\mathbf{x}(t-1) + \gamma\mathbf{F}_x\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right] \\ \mathbf{y}(t) &= \left[\mathbf{I} - \mathbf{D}\right]^{-1}\left[\mathbf{y}(t-1) + \gamma\mathbf{F}_y\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right]\end{aligned} \tag{11-44}$$

The term $\left[\mathbf{I} - \mathbf{D}\right]^{-1}$ does not depend on k nor t, so it is computed only once for fixed values of α and β. Letting this term be represented as a constant matrix,

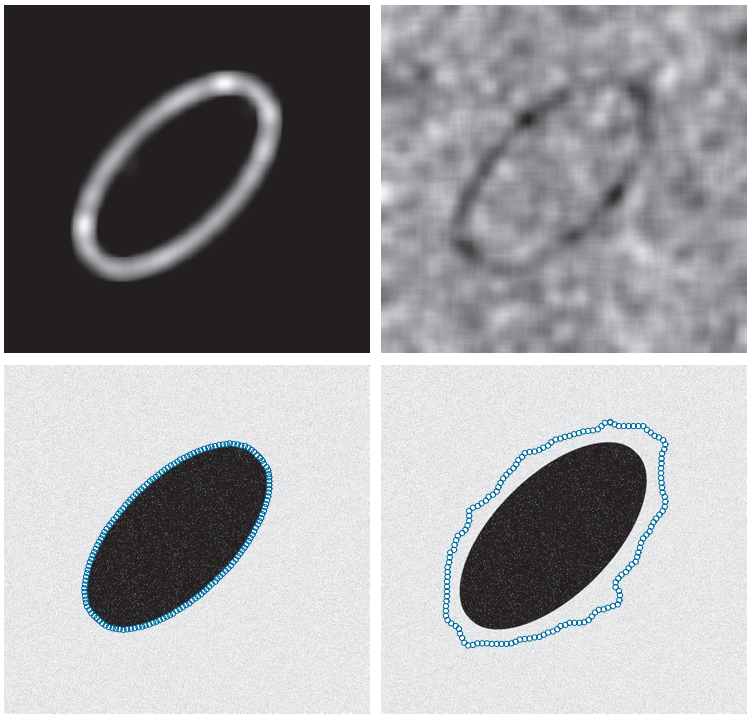
$$\mathbf{A} = \left[\mathbf{I} - \mathbf{D}\right]^{-1} \tag{11-45}$$

we can write Eq. (11-44) as

All vectors in this equation are K-dimensional. Matrix $\mathbf{A}$ is of size $K \times K$.

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{A}\left[\mathbf{x}(t-1) + \gamma\mathbf{F}_x\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right] \\ \mathbf{y}(t) &= \mathbf{A}\left[\mathbf{y}(t-1) + \gamma\mathbf{F}_y\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)\right]\end{aligned} \tag{11-46}$$

These two equations constitute the *iterative form* of the snake equation. As you can see, we have reduced the problem of finding a segmentation snake to solving two straightforward iterative equations—a trivial task, especially in a matrix-oriented language, such as MATLAB. The approach is to specify the coordinates $\mathbf{x}(0)$ and $\mathbf{y}(0)$ of an initial snake, then solve this equation iteratively for $t = 1, 2, \ldots$. At any value of t, $\mathbf{x}(t)$ and $\mathbf{y}(t)$ are vectors containing all the coordinates of the snake at that iterative step, while $\mathbf{x}(t-1)$ and $\mathbf{y}(t-1)$ are vectors containing the coordinates from the previous step. Similarly, $\mathbf{F}_x\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)$ and $\mathbf{F}_y\left(\mathbf{x}(t-1), \mathbf{y}(t-1)\right)$ are vectors containing the x and y components of the forces acting on all points of the snake at step $t-1$. In theory, the snake is said to have converged when $\mathbf{c}$ stops changing; that is, when $\mathbf{c}(t) = \mathbf{c}(t-1)$. In practice, we have to allow for variations due to factors such as noise, so comparing for equality is not feasible. One of the simplest ways to measure change is to compute the vector norm of the difference, $\|\mathbf{c}(t) - \mathbf{c}(t-1)\|$, and say that the snake has converged if the norm is less than a predefined threshold. All that



a b
c d

FIGURE 11.3
(a) Edge map used to generate the results in Fig. 11.2. (b) Edge map with only the MOG filtered and then thresholded. (c) Result after 200 iterations using the forces based on (a). (d) Result after 200 iterations using the forces based on (b). The initial snake is shown in Fig. 11.2(a). *(Continued)*

This fact is illustrated vividly by the force field in Fig. 11.5, which was obtained using the edge map in Fig. 11.3(b). Note the randomness of the force vectors throughout the entire force field; this explains the reason for the poor result in Fig. 11.3(d).

The points in the initial snake in Fig. 11.2(a) are equidistant (i.e., the arc distance between adjacent points is the same). However, because no provision is made in Eq. (11-46) for maintaining this spatial

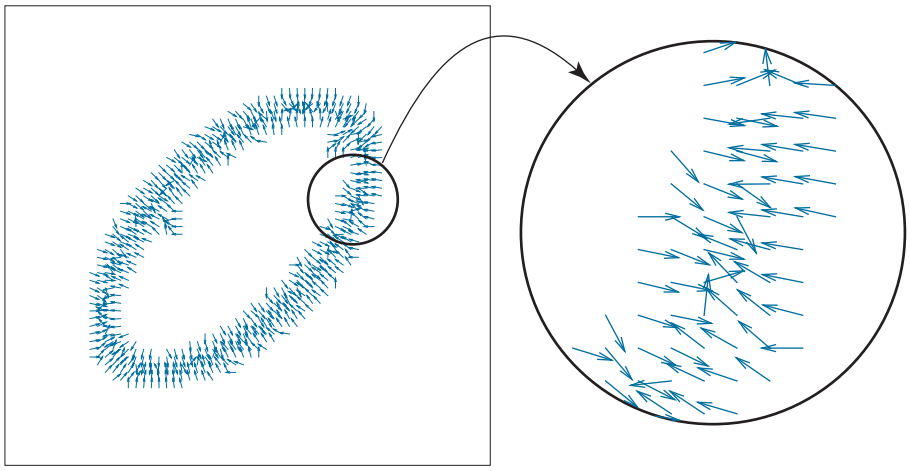


FIGURE 11.4
Force field obtained using the edge map in Fig. 11.3(a). All the arrows are of the same length because each element of the force field was normalized using Eqs. (11-49) and (11-50).

a b

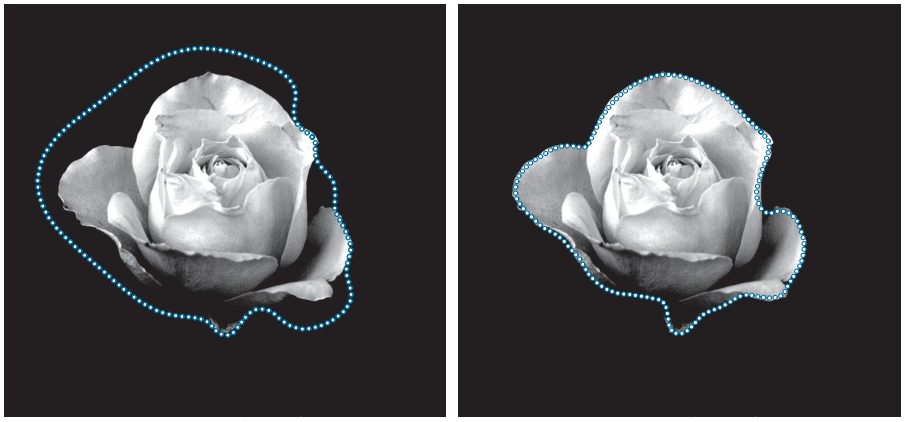
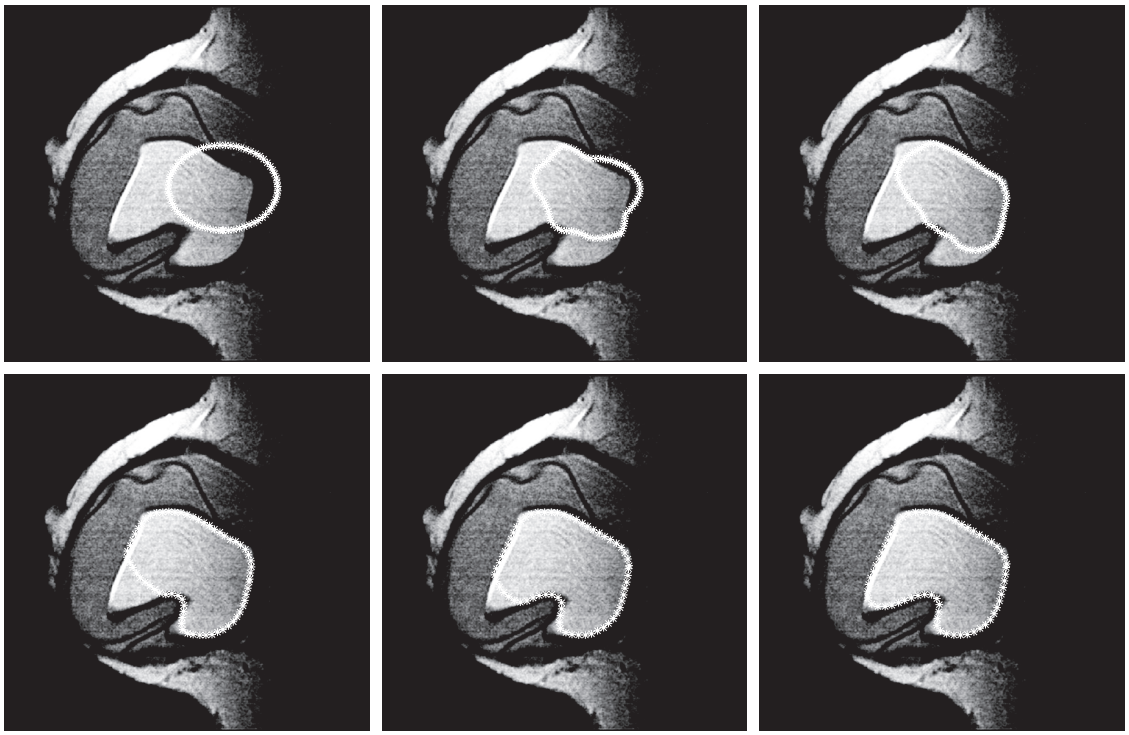


FIGURE 11.12
(a) MOG-based snake after 90 iterations. The snake is beginning to attach itself to the boundary, but it has a long way to go before it fully converges to the boundary. (b) GVF-based snake after the same number of iterations.



a b c
d e f

FIGURE 11.13 (a) 586×600 MRI image of a breast implant and initial snake. Results after: (b) 10, (c) 20, (d) 40, (e) 60, and (f) 80 iterations, respectively. The snake parameters used were $\alpha = 0.05$, $\beta = 0.5$, and $\gamma = 2.5$. (Original image courtesy of NIH/National Library of Medicine.)

implant (the ellipse shown is the initial snake). Our interest is in obtaining the boundary of the implant. As motivation for this type of processing, imagine you were conducting a study of a historical medical database containing thousands of images of breast implants. An important aspect of such a study might be to analyze the shape of the implants, in order to quantify abnormalities (e.g., collapsed implants) as a percentage of normal implants. Even if total automation is not acceptable (a typical constraint in medical image processing), a semi-automated technique, in which a human expert initiates the process by pointing to a starting location in the implants and letting a computer extract the boundary, often is acceptable. Such an approach can save many hours of effort and yield more accurate measurements than manual estimates.

To generate the results in Fig. 11.13, we used parameters similar to those in Example 11.3. The smoothing was the same, but the smoothed image was thresholded at 0.01. We used a GVF force field, with $\mu = 0.2$ and 160 iterations, which are the same settings as in Example 11.5. The results in Figs. 11.13(b) through (f) show how the snake evolved from an initial position straddling the boundary of the implant, to an almost perfect segmentation of that region. Observe how the snake contracted initially, and then expanded, finally snapping to the contour of the region of interest.

11.3 SEGMENTATION USING LEVEL SETS

As we mentioned in Section 11.1, level sets in our context are sets of points of a 2-D curve formed by the intersection of a plane and a 3-D surface. Unlike the parametric representation used for snakes, level sets are based on implicit representations. An important aspect of this approach is that it can adapt to changing topology (e.g., the discovery of "holes" within a region, and the emergence of new regions) during curve evolution. Inherently, parametric curves do not have this capability. However, as we will illustrate later in this section, each approach has strengths that make it an appropriate choice in a given application. As noted in Section 11.1, level sets were used initially to describe the propagation of interfaces between fluids. In the terminology of image segmentation, "fluids" represent image regions, and "interfaces" become segmentation contours separating one region from another.

IMPLICIT REPRESENTATION OF ACTIVE CONTOURS

The representation of snakes discussed in Section 11.2 is *explicit*, in the sense that an active contour is represented by an equation, written typically in Cartesian or (more frequently) parametric form. An alternate representation of a 2-D contour is to define it *implicitly* as the intersection of a plane and a 3-D surface. To illustrate, consider the explicit equation of a circle centered at point (x_0, y_0) in the xy-plane:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

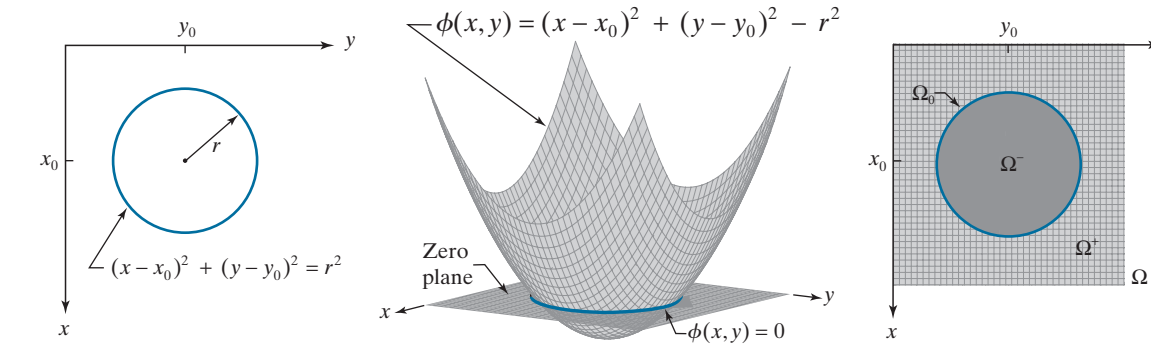
Figure 11.14(a) shows a generic plot of this function. We can write this equation equivalently as

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$$

Remember, a scalar function outputs a scalar value, regardless of the number of scalar variables on which it depends.

Suppose that we define the following scalar function of two variables:

$$\phi(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2$$



a b c

FIGURE 11.14 (a) Cartesian equation of a circle and its corresponding plot. (b) The same circle, obtained as the level set curve $\phi(x, y) = 0$ (i.e., the intersection of $\phi(x, y)$ and the zero plane). (c) Top view of (b); the dark area enclosed by the circle is the visible section of the zero-plane. The symbols Ω_0, Ω^-, and Ω^+, are the sets of points (*on the plane*) that are on, inside, and outside the boundary, respectively, while Ω represents the entire image plane.

Figure 11.14(b) is a plot of ϕ as a function of x and y. As you can see, ϕ is a surface in 3-D, while the equation of the circle is a curve in 2-D. However, we can extract the circle from the surface as the locus of points in the intersection of ϕ with the zero-plane. This set of points is given by values of x and y for which $\phi(x, y) = 0$, as Fig. 11.14(b) shows. Figure 11.14(c) is a top view, showing the intersection more clearly.

The set of points in the intersection just mentioned is called a *level set*, and ϕ is usually referred to as a *level set function*. When dealing with two variables (which is our focus in this chapter), the level set reduces to a *level set curve* C that, based on the preceding discussion, we define as

$$C = \{(x, y) | \phi(x, y) = 0\} \tag{11-57}$$

We use the term *level set* or *level set function* to refer to the 3-D function $\phi(x,y)$ in general, and *zero-level-set*, *zero-level-set function*, or *level-set curve* to refer to the 2-D curve defined by the equation $\phi(x,y) = 0$.

That is, a level curve is the set of points (x, y) such that $\phi(x, y) = 0$.[†] Viewed another way, we say that a 2-D curve C is *embedded* into a 3-D function ϕ by letting C be the zero-level set of this function. In the discussion that follows, level curves will become segmentation boundaries, and the power of this concept is that the level set approach does not require an explicit representation of these boundaries. In fact, you can easily visualize that if $\phi(x, y)$ in Fig. 11.14 were more complex, C in Eq. (11-57) could represent an arbitrarily complex curve that would be defined simply by the locus of points satisfying this equation.

Because the level set curves with which we work in this chapter are closed, it follows that $\phi(x, y)$ satisfies the following conditions for an arbitrary point (x, y):

[†] In reality this is the *zero-level-set* curve of $\phi(x, y)$. In general, a level set curve can be defined for any constant c, such that $\phi(x, y) = c$. In terms of Fig. 11.14, values of $c > 0$ would yield circles of a larger diameter than the one at the zero-plane, and $c < 0$ would yield circles of smaller diameter. Note that changing c changes the location (level) of the intersecting plane, hence the use of the term *level* in "level sets."

TABLE 11.1
Properties of signed distance functions. Regions Ω_1^- and Ω_2^- are regions enclosed by the zero level sets (i.e. boundaries) of signed distance functions $\phi_1(x,y)$ and $\phi_2(x,y)$, respectively.

Property	Description
1) Unit gradient magnitude.	$\|\nabla\phi(x,y)\| = 1$
2) Unit normal to the boundary at point (x,y).	$\mathbf{n} = \dfrac{\nabla\phi(x,y)}{\|\nabla\phi(x,y)\|} = \nabla\phi(x,y)$
3) Mean curvature (equal to the Laplacian).	$\kappa = \nabla\cdot\left(\dfrac{\nabla\phi(x,y)}{\|\nabla\phi(x,y)\|}\right) = \nabla\cdot\nabla\phi(x,y) = \nabla^2\phi(x,y) = \text{Laplacian}[\phi(x,y)]$
4) Point (x_B, y_B) on the boundary closest to an arbitrary point (x,y) on the plane (see Fig. 11.14).	$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \phi(x,y)\,\mathbf{n}$ where $\mathbf{n}$ is the unit normal from Property 2.
5) Convexity.	If Ω^- is convex, then its corresponding signed distance function, $\phi(x,y)$, is a convex function. (See Section 9.5 regarding convexity.)
6) Signed distance function of union.	The signed distance function of the union $\Omega_1^- \cup \Omega_2^-$ is given by $\phi(x,y) = \min\left(\phi_1(x,y), \phi_2(x,y)\right)$.
7) Signed distance function of intersection.	The signed distance function of the intersection $\Omega_1^- \cap \Omega_2^-$ is given by $\phi(x,y) = \max\left(\phi_1(x,y), \phi_2(x,y)\right)$.
8) Signed distance function of difference.	The signed distance function of the set difference $(\Omega_1^- - \Omega_2^-)$ is given by $\phi(x,y) = \max\left(\phi_1(x,y), -\phi_2(x,y)\right)$.
9) Inclusion [Eq. (11-89)]. The arrow is used to denote "implies."	For a given (x,y), $[\phi(x,y) > 0] \Rightarrow (x,y) \in \Omega^+$; $[\phi(x,y) < 0] \Rightarrow (x,y) \in \Omega^-$; and $[\phi(x,y) = 0] \Rightarrow (x,y) \in \Omega_0$.

$$\begin{aligned}\|\nabla\phi(x,y)\| &= \sqrt{(\partial\phi/\partial x)^2 + (\partial\phi/\partial y)^2} \\ &= \sqrt{\frac{x^2}{x^2+y^2} + \frac{y^2}{x^2+y^2}} \\ &= 1\end{aligned}$$

showing that, indeed, the given ϕ is a signed distance function.

Although circular functions are useful, their fixed shape limits their applicability. Fortunately, signed distance functions of arbitrary shape are not difficult to construct. We begin by specifying any closed interface curve that suits our initial purpose. This curve is on the plane, so the condition $\phi(x,y) = 0$ is automatically satisfied. Because the curve is closed, it follows that there is an infinite number of possible choices for the "rest" of ϕ such that any point inside the curve will give $\phi(x,y) < 0$, and any

Our final task in implementing a level set solution for image segmentation is to specify a force function, F, for use in the general iterative algorithm in Eq. (11-82). In the following sections, we will discuss two basic classes of force functions suitable for level set image segmentation. The first and simplest are forces based only on properties of the image to be segmented. The key advantage of these forces is that they can be precomputed because the input image obviously is available from the beginning. This reduces the computational load significantly. Their principal disadvantage is that properties of the level set function itself are ignored. Forces in the second category are based on properties of both the image and level set function. These forces are in general more powerful, but they have the disadvantage that they must be computed at every iterative step because the level set function itself changes during iterations of Eq. (11-82).

FORCE FUNCTIONS BASED ONLY ON IMAGE PROPERTIES

As an introduction to forces suitable for level set segmentation, we start with forces used for segmenting binary images. Because of their simplicity, these forces are an ideal way to introduce a number of important concepts that we will use in the remainder of this chapter. Because they depend only on pixel intensity values, the forces discussed in this section are calculated only once for a given image. During evolution, the force value acting on a point on the interface is determined completely by the (x, y) location of that point. Because the force is precomputed, we can find $\max|F|$ and use Eq. (11-83) to compute Δt. All examples in this section use this method for computing the time step.

In terms of segmentation, binary images are viewed as being composed of objects and background. The objects can be darker (0) than the background (1), or vice versa, but not both. For consistency in the following discussion, we assign 0's to dark pixels and 1's to light pixels. The simplest force we can consider is based only on the intensity of individual pixels. For example,

$$F(x,y) = \begin{cases} a & \text{if } f(x,y) = 1 \\ b & \text{if } f(x,y) = 0 \end{cases} \tag{11-93}$$

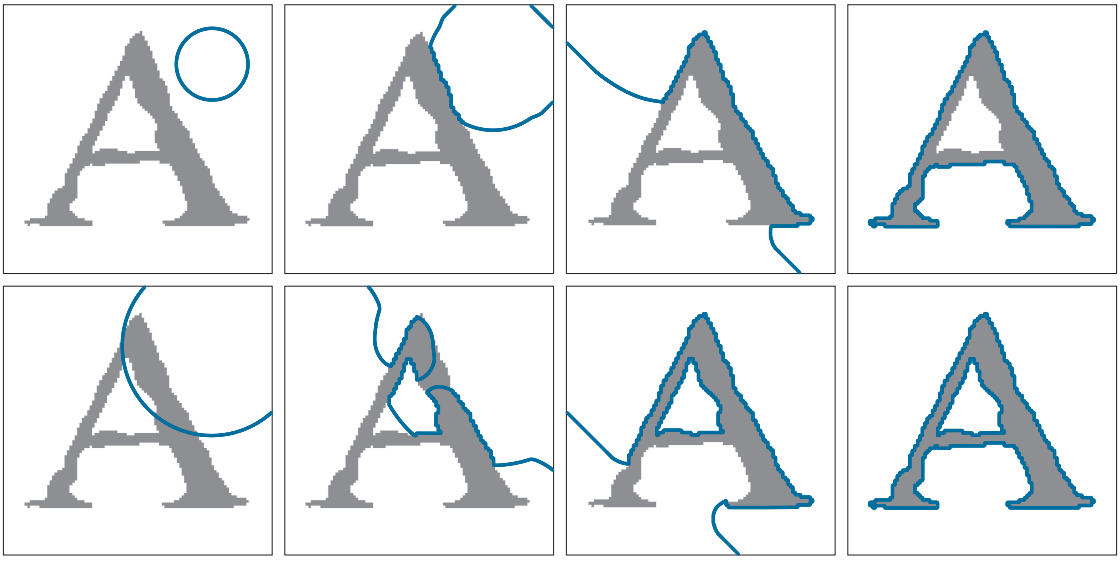
where a and b are constants whose values determine how ϕ behaves as a function of image intensity. We can write this equation equivalently as

$$F(x,y) = af(x,y) + b[1 - f(x,y)] \tag{11-94}$$

To illustrate how to use this force in Eq. (11-82), consider Fig. 11.18. Part (a) of this figure shows a dark object on a light background. The curve shown is ϕ^0, specified as a circle with an offset center (see Problem 11.17). Suppose that we let $a = 1$ and $b = 0$. The force in Eq. (11-94) then becomes

$$F(x,y) = f(x,y)$$

All points in the initial curve have value 1 because they are in the background. Thus, all values of F at that location are 1, and the force causes the circle to begin expanding (recall from Example 11.7 that a positive constant force expands a

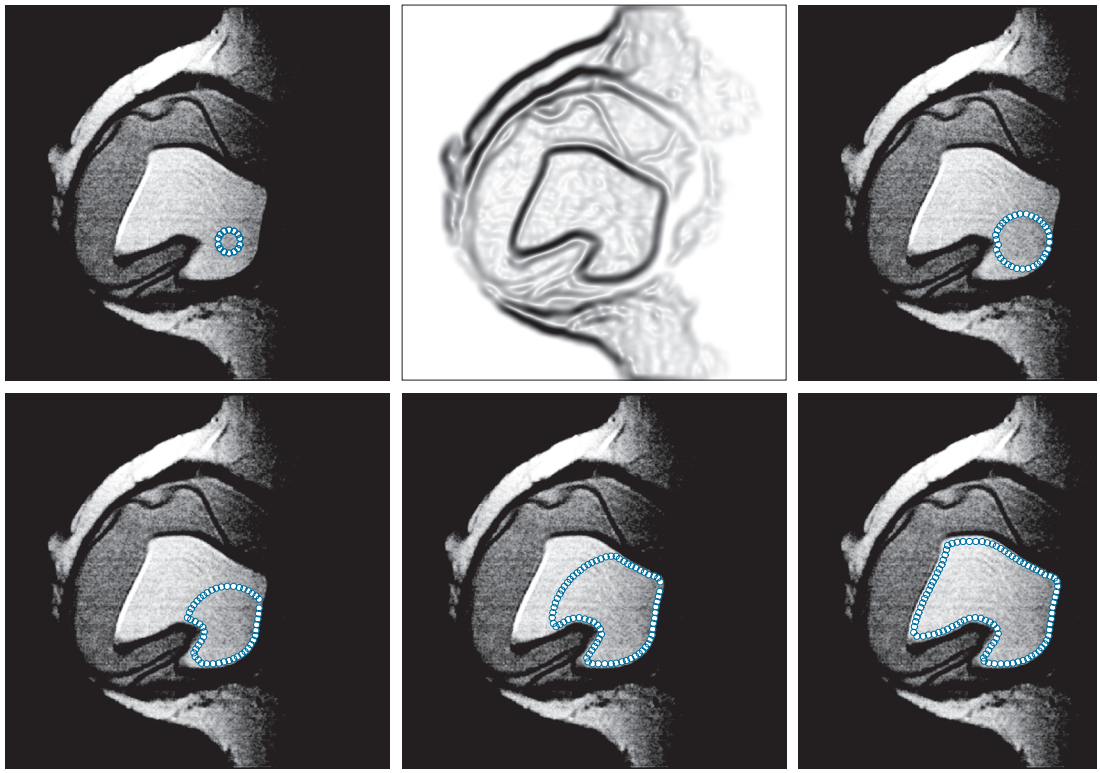


a b c d
e f g h

FIGURE 11.18 (a) Character image and initial zero level set boundary (gray = 0 and white = 1). (b)–(d) Results after 100, 400, and 900 iterations of Eq. (11-82) with $a = 1$ and $b = 0$ in the force definition. Only the outer boundary was detected. (e) A different initial level set function. (f)–(h) Results after 100, 400, and 900 iterations with $a = 1$ and $b = -1$ in the force definition. Both outer and inner boundaries were detected. (All curves are closed, but their values in (a), (d), and (h) are outside the confines of the image area.)

circle out uniformly). The expansion will continue until the evolving curve reaches the boundary of an object. Object points in this case have value 0 so, because $F(x,y) = f(x,y)$, the force acting on points on the object boundary will also be 0, meaning that points in the evolving curve will cease to move once they reach the boundary of an object. However, any curve points still residing on the background will continue to expand until they too reach the boundary. Eventually, the evolving curve will wrap itself completely around the object. Figures 11.18 (c) and (d) show various stages of this process.

Because the evolving curve was stopped at the outer boundary of the object, the inner white region was missed completely. In order to detect it, the initial curve has to contain at least one point in that region, as in Fig. 11.18(e). Then, because the force associated with white points is positive [we are still using $F(x,y) = f(x,y)$], that part of the curve will expand until it is stopped by the dark inner boundary, thus completely defining the inner region. However, part of the initial curve is now contained in the object (whose values are 0) and will not evolve because, again, the force we are using will be zero when $f(x,y) = 0$. Setting b to a positive value would not work because we do not want the curve to expand within the character. What we want is for the curve segments within the character to shrink, and this is accomplished by using a negative force, say –1. Then, letting $a = 1$ and $b = -1$, our force specification becomes



a b c
d e f

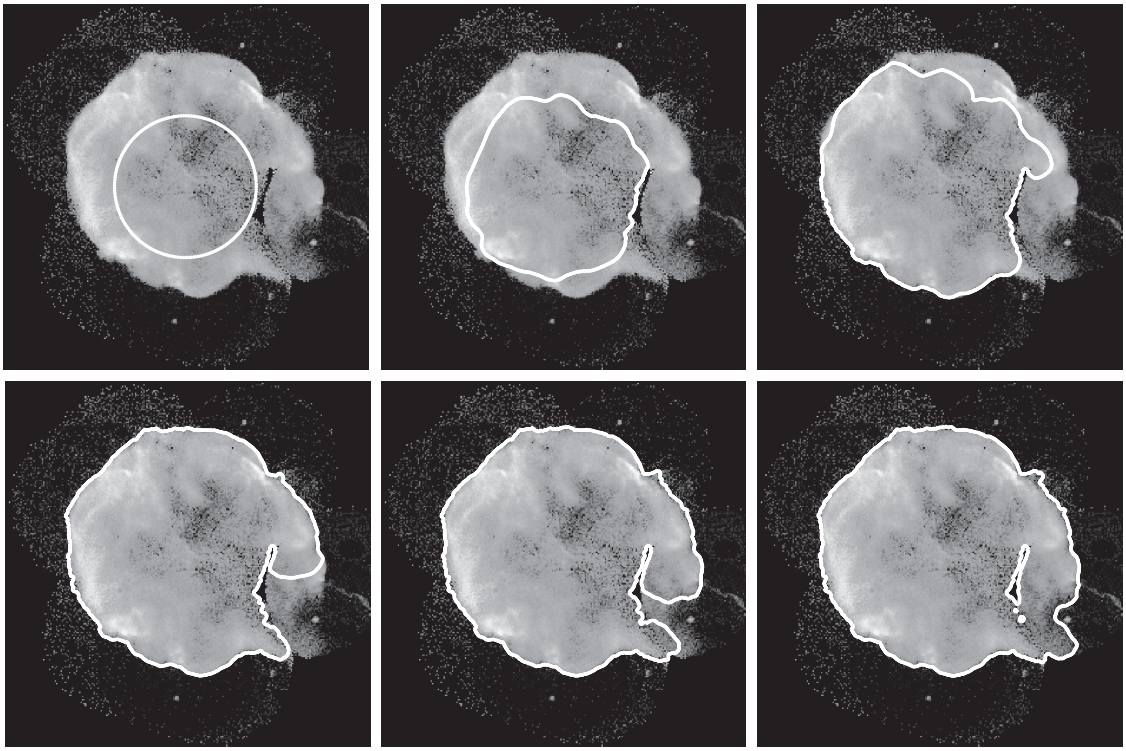
FIGURE 11.22 (a) 586×600 MRI image of a breast implant and initial level set curve. (b) Force field displayed as an image. Results after: (c) 50 iterations, (d) 100 iterations, (e) 200 iterations, and (f) 400 iterations. (Original image courtesy of NIH/National Library of Medicine.)

EDGE/CURVATURE-BASED FORCES

The forces discussed in the previous section are based only on image intensities. In this section we discuss forces based on image (edge), and level set (curvature) properties. Unlike the forces in the previous section, the forces discussed next have to be computed at every step because the level set function changes during iterations of Eq. (11-82).

In addition to being more general, the method discussed next is important from a historical perspective because it establishes that energy-minimizing solutions, like snakes, can be related to a level set solution based on *geodesic curves*.[†] The concept is based on minimizing a snake-like function of the form

[†] A *geodesic curve* is a local, length-minimizing curve. Equivalently, it can be interpreted as the path that a particle would follow if it is not accelerating. In a plane, geodesic curves are straight lines. In general, Euclidean geometry studies shapes on a plane. Riemann geometry is concerned with the way shapes work in spaces that curve back on themselves (e.g., how curves behave on the surface of a sphere, where geodesic lines are great circles, like the equator).



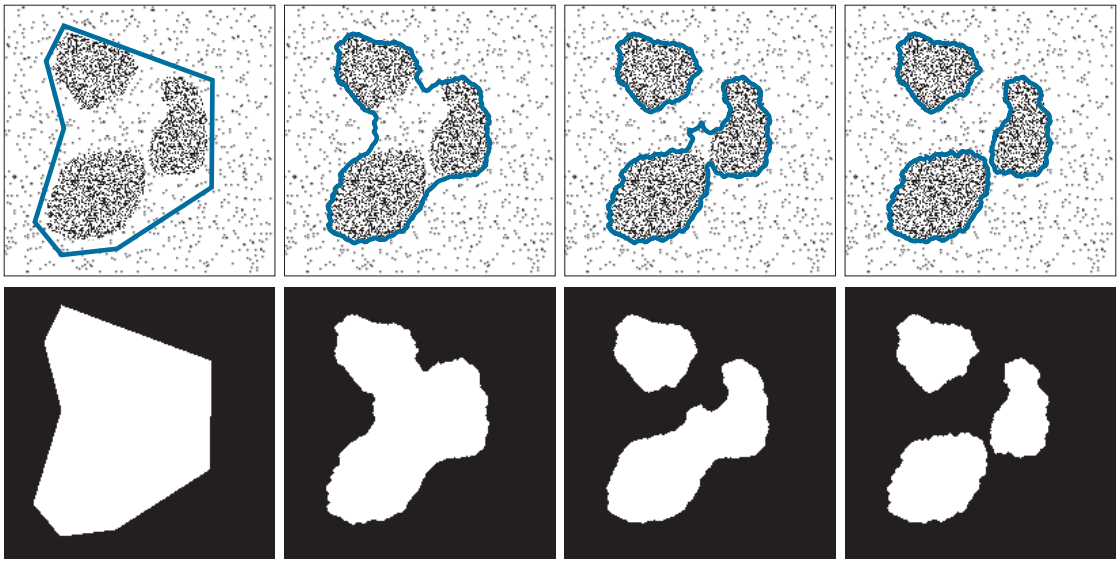
a b c
d e f

FIGURE 11.31 (a) Image and initial contour. (b) through (f) Results after 500, 1000, 1500, 2000, and 3500 iterations respectively. The following parameters were used: $\mu = 2$, $\nu = 0$, $\lambda_1 = 1$, and $\lambda_2 = 1$. Compare (f) with the results in Fig. 10.48(d). (Original image courtesy of NASA.)

EXAMPLE 11.14: Level set segmentation of multiple regions using region-based forces.

Figure 11.32(a) shows a noisy image containing three regions. The initial boundary, shown superimposed on the image, was specified interactively and then converted to a signed distance function using the approach discussed in Example 11.11. Figures 11.32(b) through (d) are the results of the number of iterations with the value of μ shown in the caption. Normalization and reinitialization was done as in the previous example. The resulting segmentation contours in Fig. 11.32(d) are an accurate representation of the content of the image in terms of the number of relevant regions. The second row shows the segmentation regions themselves, obtained as in Fig. 11.23.

As noted earlier, parameter μ controls the influence exercised by curvature on the segmentation process. If using $\mu = 0.5$ in the previous example resulted in three regions being detected, we would expect that increasing this parameter would result in fewer segmented regions. As Fig. 11.33 shows, this indeed was the case (using smaller values of μ would result is smaller noise points being picked up, and the region boundaries would become more ragged). The results in Fig. 11.33 also demonstrate that the region-based segmentation method has a rudimentary region-size "filtering" capability. None of the methods discussed earlier are able to do this.



a b c d
e f g h

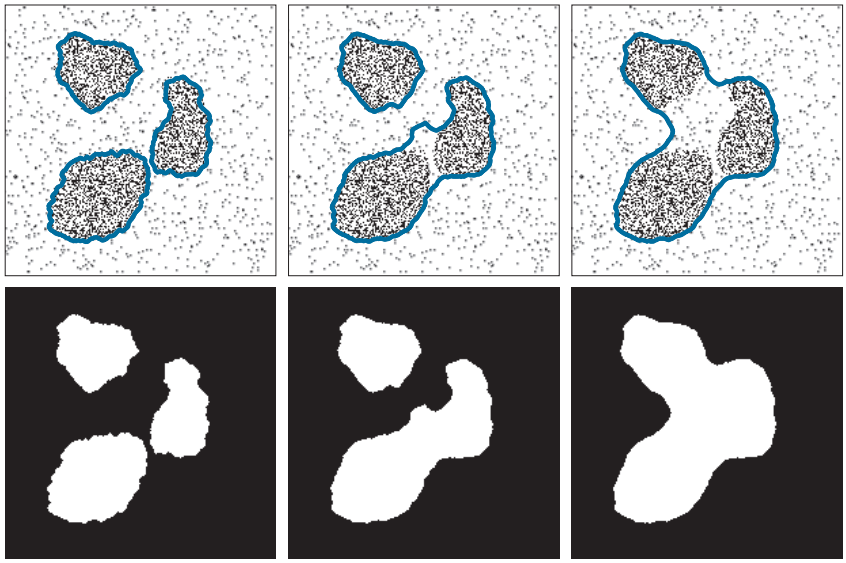
FIGURE 11.32 (a) 200×200 image containing three regions and an initial contour. (b)–(d) Results (using $\mu = 0.5$) after 500, 1000, and 1500 iterations, respectively. The second row shows the corresponding segmented regions (i.e., regions for which $\phi(x, y) \leq 0$).

EXAMPLE 11.15: Level set segmentation of the rose image using region-based forces.

Figure 11.34 shows segmentation of the rose image using region-based active contours. The number of iterations and parameters used are listed in the figure caption. The relatively low value $\mu = 0.5$ was selected to allow the contour to penetrate the troublesome deep concavity on the right side of the flower (see the comments in Example 11.12 regarding this issue). Normalization of the force and curvature were done as in the previous two examples, but the relatively low noise content and uniformity of the area where the curve evolved made reinitialization of the level set function unnecessary. For the most part, the contour evolved nicely to enclose the principal object of interest, but the segmentation is not as accurate as what we obtained with earlier methods. We discuss the reasons why in the following example.

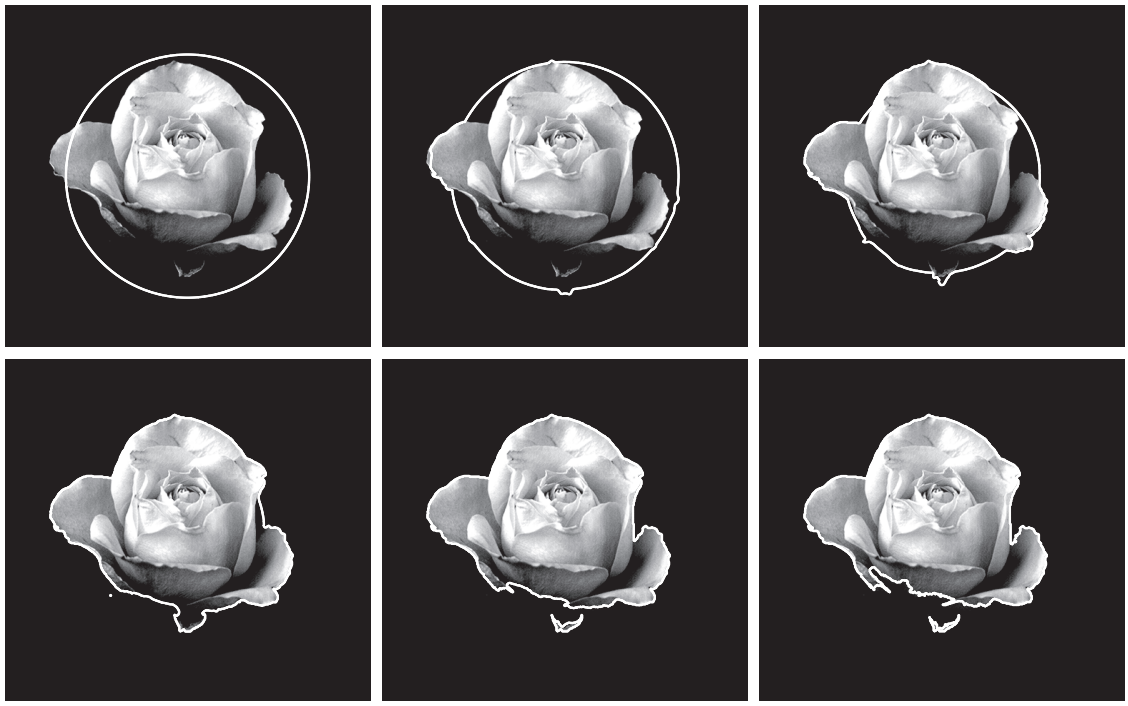
EXAMPLE 11.16: Some comparisons between snakes and level sets.

We conclude our list of examples of active contours with several comparisons between snakes and the two principal grayscale level set methods developed in this chapter. Figure 11.35 shows segmentation of the rose image using snakes, edge-based level sets, and region-based level sets. The boundaries are displayed with the same graphic symbols for consistency. The first obvious difference is in the smoothness of the contours. The snake result is the smoothest, a fact that can be attributed to the smoothing effect characteristic of the parametric representation of snakes. In contrast, the edge-based approach was powerful enough to take advantage of the clear definition of the original edges, without the need for blurring to extend the influence of the edges, as we had to do for the snake. This is important, because



a b c
d e f

FIGURE 11.33 Effect of parameter μ on the number of regions detected. (a) $\mu = 0.5$. (b) $\mu = 1.5$. (c) $\mu = 3.0$. (d)–(f) Corresponding segmented regions (i.e., regions for which $\phi(x, y) \leq 0$).



a b c
d e f

FIGURE 11.34 (a) Image and initial contour. Results after: (b) 100, (c) 300, (d) 500, (e) 700, and (f) 1100 iterations, respectively. We used $\mu = 0.5$ in all cases.

initial curve encloses all three regions. Let the object pixels be denoted by 0, and the background pixels be denoted by 1. Use $a = 1$ and $b = -1$ in Eq. (11-94).

(b) Repeat (a) but with the initial contour starting as a small circle near the center of the image (without touching any of the three regions).

11.32 If we let $a = -1$ and $b = 1$ in Eq. (11-94), and start with the configuration in the figure below, what would the segmentation contour look like at convergence?

11.33 Propose a segmentation solution based on morphology to obtain the same segmentation result as in Fig. 11.20(f).

11.34 It is possible to use a Gaussian function to accomplish the same objective as Eq. (11-96) for generating a force field that can be used for level set segmentation.

(a) Propose such a function using the same parameters, σ and λ, as in Eq. (11-96). [*Note:* Your function should have a Gaussian, form and not be a modification of Eq. (11-96).]

(b) Discuss how the parameters σ and λ would affect the shape of your function.

11.35* Demonstrate the validity of Eq. (11-100).

11.36 With reference to Fig. 11.41(d), assume that the force acting on point p_2 is negative. What would the boundary look like after p_2 is processed?

11.37 What would the figure below look like after the application of Procedure 3 in Table 11.4?

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

11.1 Snake input and display.

(a)* Read the image **rose512.tif** and use the utility function **snake_manual_input.m** to generate and display the coordinates of a 150-point snake enclosing the rose. The snake points should be displayed as small yellow circles.

(b) Write the MATLAB code necessary to generate a circular snake centered in the middle of image **rose512.tif** and large enough to enclose the rose without touching it or the image border. Use the utility function **snake_display.m** to display the image with the boundary superimposed on it using small red circles. (*Hint:* Use the parametric representation of a circle to generate the circular snake.)

11.2 Snake edge map.

(a)* Write a function, **emap = snakeMap4e(f,T,sig,n, order)**, to compute the edge map, **emap**, of input image **f**. If only **f** is provided in the input, **emap** will equal the magnitude of the gradient (MOG) of **f** without thresholding. If only **f** and **T** are provided, **emap** is thresholded such that **emap** > **T**, where threshold **T** is the range [0, 1]. No smoothing filter is applied. If all inputs are provided, the gradient is thresholded. Filtering with a Gaussian kernel of size **(n*sig)*(n*sig)** and standard deviation **sig** is determined by **order**, a character string with the following possible values: If **order = 'before'** then image **f** is filtered before the map (gradient) is computed. If **order = 'after'** filtering is performed on the edge map after it is computed. If **order = 'both'** filtering is performed before and after the edge map

12 Feature Extraction

Well, but reflect; have we not several times acknowledged that names rightly given are the likenesses and images of the things which they name?

Socrates

Preview

After an image has been segmented into regions or their boundaries using methods such as those in Chapters 10 and 11, the resulting sets of segmented pixels usually have to be converted into a form suitable for further computer processing. Typically, the step after segmentation is *feature extraction*, which consists of feature detection and feature description. *Feature detection* refers to finding the features in an image, region, or boundary. *Feature description* assigns quantitative attributes to the detected features. For example, we might *detect* corners in a region boundary, and *describe* those corners by their orientation and location, both of which are quantitative attributes. Feature processing methods discussed in this chapter are subdivided into three principal categories, depending on whether they are applicable to boundaries, regions, or whole images. Some features are applicable to more than one category. Feature descriptors should be as insensitive as possible to variations in parameters such as scale, translation, rotation, illumination, and viewpoint. The descriptors discussed in this chapter are either insensitive to, or can be normalized to compensate for, variations in one or more of these parameters.

Upon completion of this chapter, readers should:

- Understand the meaning and applicability of a broad class of features suitable for image processing.
- Understand the concepts of feature vectors and feature space, and how to relate them to the various descriptors developed in this chapter.
- Be skilled in the mathematical tools used in feature extraction algorithms.
- Be familiar with the limitations of the various feature extraction methods discussed.
- Understand the principal steps used in the solution of feature extraction problems.
- Be able to formulate feature extraction algorithms.
- Have a "feel" for the types of features that have a good chance of success in a given application.

12.1 BACKGROUND

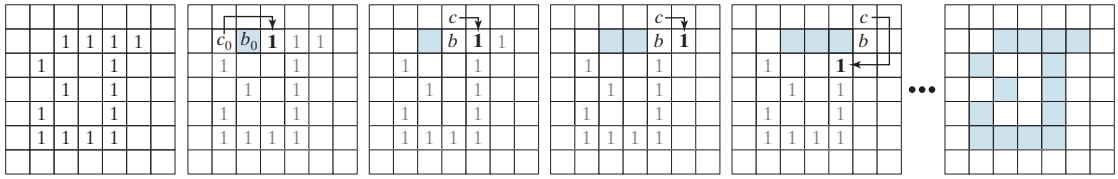
Although there is no universally accepted, formal definition of what constitutes an *image feature*, there is little argument that, intuitively, we generally think of a feature as a distinctive attribute or description of "something" we want to label or differentiate. For our purposes, the key words here are *label* and *differentiate*. The "something" of interest in this chapter refers either to individual image objects, or even to entire images or sets of images. Thus, we think of features as attributes that are going to help us assign unique labels to objects in an image or, more generally, are going to be of value in differentiating between entire images or families of images.

There are two principal aspects of image *feature extraction*: *feature detection*, and *feature description*. That is, when we refer to feature extraction, we are referring to both detecting the features and then describing them. To be useful, the extraction process must encompass both. The terminology you are likely to encounter in image processing and analysis to describe feature detection and description varies, but a simple example will help clarify our use of these term. Suppose that we use object corners as features for some image processing task. In this chapter, detection refers to *finding* the corners in a region or image. Description, on the other hand, refers to *assigning quantitative* (or sometimes *qualitative*) *attributes* to the detected features, such as corner orientation, and location with respect to other corners. In other words, knowing that there are corners in an image has limited use without additional information that can help us differentiate between objects in an image, or between images, based on corners and their attributes.

Given that we want to use features for purposes of differentiation, the next question is: What are the important characteristics that these features must possess in the realm of digital image processing? You are already familiar with some of these characteristics. In general, features should be independent of location, rotation, and scale. Other factors, such as independence of illumination levels and changes caused by the viewpoint between the imaging sensor(s) and the scene, also are important. Whenever possible, preprocessing should be used to normalize input images before feature extraction. For example, in situations where changes in illumination are severe enough to cause difficulties in feature detection, it would make sense to preprocess an image to compensate for those changes. Histogram equalization or specification come to mind as automatic techniques that we know are helpful in this regard. The idea is to use as much a priori information as possible to preprocess images in order to improve the chances of accurate feature extraction.

When used in the context of a feature, the word "independent" usually has one of two meanings: invariant or covariant. A feature descriptor is *invariant* with respect to a set of transformations if its value remains unchanged after the application (to the entity being described) of any transformation from the family. A feature descriptor is *covariant* with respect to a set of transformations if applying to the entity any transformation from the set produces the same result in the descriptor. For example, consider this set of affine transformations: {*translation*, *reflection*, *rotation*}, and suppose that we have an elliptical region to which we assign the feature descriptor *area*. Clearly, applying any of these transformations to the region does not change its area.

See Table 2.3 regarding affine transformations.



a b c d e f

FIGURE 12.1 Illustration of the first few steps in the boundary-following algorithm. The point to be processed next is labeled in bold, black; the points yet to be processed are gray; and the points found by the algorithm are shaded. Squares without labels are considered background (0) values.

The following algorithm traces the boundary of a 1-valued region, R, in a binary image.

See Section 2.5 for the definition of 4-neighbors, 8-neighbors, and m-neighbors of a point,

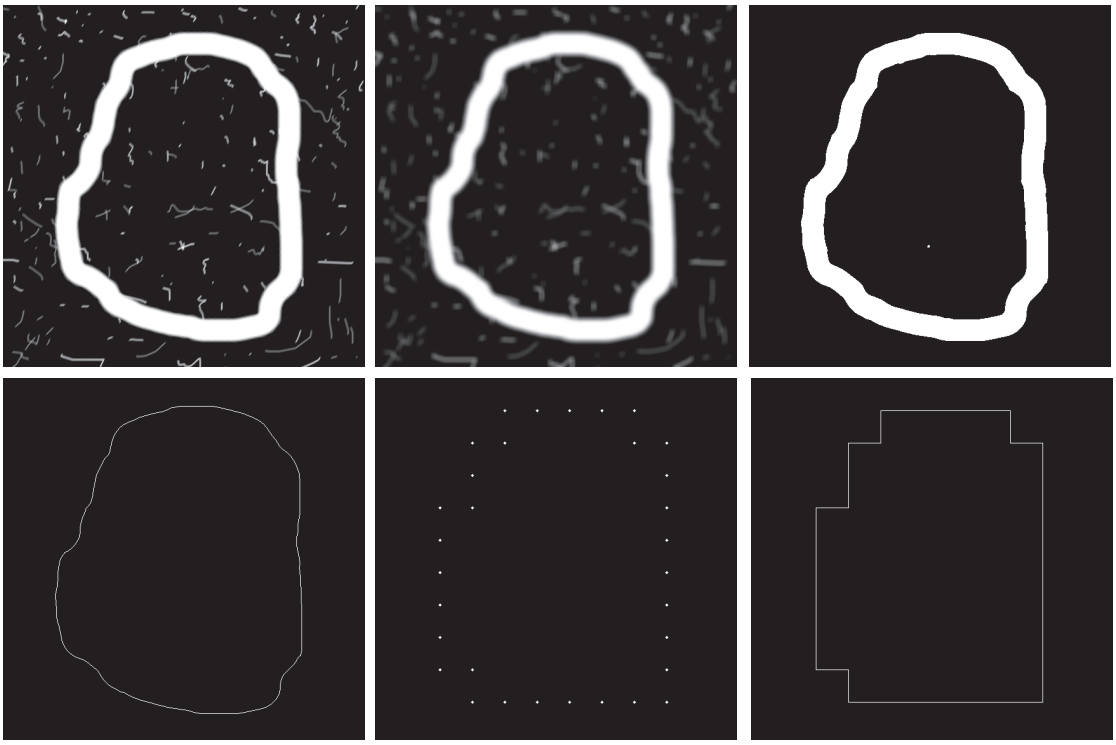
1. Let the starting point, b_0, be the *uppermost-leftmost* point† in the image that is labeled 1. Denote by c_0 the *west* neighbor of b_0 [see Fig. 12.1(b)]. Clearly, c_0 is always a background point. Examine the 8-neighbors of b_0, starting at c_0 and proceeding in a clockwise direction. Let b_1 denote the *first* neighbor encountered whose value is 1, and let c_1 be the (background) point immediately preceding b_1 in the sequence. Store the locations of b_0 for use in Step 5.
2. Let $b = b_0$ and $c = c_0$.
3. Let the 8-neighbors of b, starting at c and proceeding in a clockwise direction, be denoted by $n_1, n_2, \ldots, n_8$. Find the first neighbor labeled 1 and denote it by n_k.
4. Let $b = n_k$ and $c = n_{k-1}$.
5. Repeat Steps 3 and 4 until $b = b_0$. The sequence of b points found when the algorithm stops is the set of ordered boundary points.

Note that c in Step 4 is always a background point because n_k is the first 1-valued point found in the clockwise scan. This algorithm is referred to as the *Moore boundary tracing algorithm* after Edward F. Moore, a pioneer in cellular automata theory.

Figure 12.1 illustrates the first few steps of the algorithm. It is easily verified (see Problem 12.1) that continuing with this procedure will yield the correct boundary, shown in Fig. 12.1(f), whose points are ordered in a clockwise sequence. The algorithm works equally well with more complex boundaries, such as the boundary with an attached branch in Fig. 12.2(a) or the self-intersecting boundary in Fig. 12.2(b). Multiple boundaries [Fig. 12.2(c)] are handled by processing one boundary at a time (see Project 12.1).

If we start with a binary region instead of a boundary, the algorithm extracts the *outer boundary* of the region. Typically, the resulting boundary will be one pixel thick, but not always [see Problem 12.1(b)]. If the objective is to find the boundaries of holes in a region (these are called the *inner* or *interior boundaries* of the region),

†As you will see later in this chapter and in Problem 12.11, the uppermost-leftmost point in a 1-valued boundary has the important property that a polygonal approximation to the boundary has a convex vertex at that location. Also, the left and north neighbors of the point are guaranteed to be background points. These properties make it a good "standard" point at which to start boundary-following algorithms.



a b c
d e f

FIGURE 12.5 (a) Noisy image of size 570×570 pixels. (b) Image smoothed with a 9×9 box kernel. (c) Smoothed image, thresholded using Otsu's method. (d) Longest outer boundary of (c). (e) Subsampled boundary (the points are shown enlarged for clarity). (f) Connected points from (e).

Slope Chain Codes

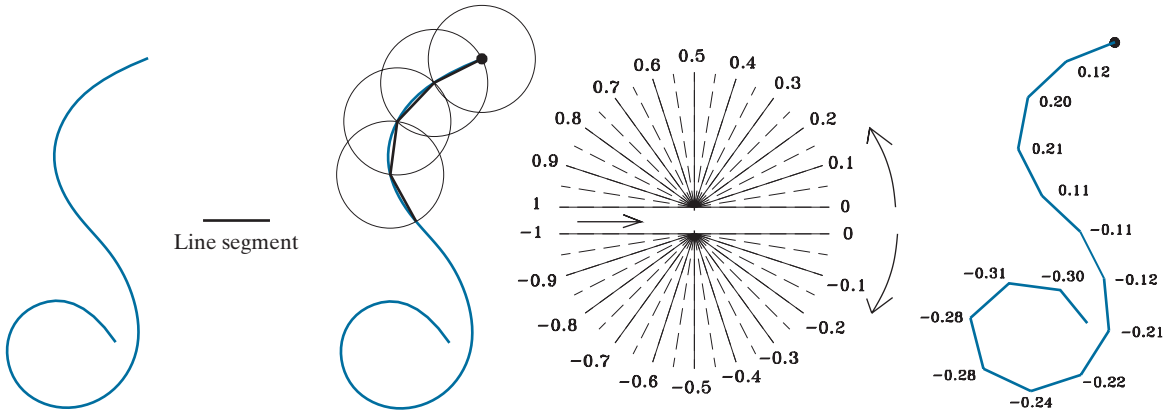
Using Freeman chain codes generally requires resampling a boundary to smooth small variations, a process that implies defining a grid and subsequently assigning all boundary points to their closest neighbors in the grid. An alternative to this approach is to use *slope chain codes* (SCCs) (Bribiesca [1992, 2013]). The SCC of a 2-D curve is obtained by placing straight-line segments of equal length around the curve, with the end points of the segments touching the curve.

Obtaining an SSC requires calculating the *slope changes* between contiguous line segments, and normalizing the changes to the *continuous* (open) interval $(-1, 1)$. This approach requires defining the length of the line segments, as opposed to Freeman codes, which require defining a grid and assigning curve points to it—a much more elaborate procedure. Like Freeman codes, SCCs are independent of rotation, but a larger range of possible slope changes provides a more accurate representation under rotation than the rotational independence of the Freeman codes, which is limited to the eight directions in Fig. 12.3(b). As with Freeman codes, SCCs are independent of translation, and can be normalized for scale changes (see Problem 12.8).

Figure 12.6 illustrates how an SCC is generated. The first step is to select the length of the line segment to use in generating the code [see Fig. 12.6(b)]. Next, a starting point (the origin) is specified (for an open curve, the logical starting point is one of its end points). As Fig. 12.6(c) shows, once the origin has been selected, one end of a line segment is placed at the origin and the other end of the segment is set to coincide with the curve. This point becomes the starting point of the next line segment, and we repeat this procedure until the starting point (or end point in the case of an open curve) is reached. As the figure illustrates, you can think of this process as a sequence of identical circles (with radius equal to the length of the line segment) traversing the curve. The intersections of the circles and the curve determine the nodes of the straight-line approximation to the curve.

Once the intersections of the circles are known, we determine the slope changes between contiguous line segments. Positive and zero slope changes are normalized to the open half interval $[0, 1)$, while negative slope changes are normalized to the open interval $(-1, 0)$. Not allowing slope changes of ± 1 eliminates the implementation issues that result from having to deal with the fact that such changes result in the same line segment with opposite directions.

The sequence of slope changes is the chain that defines the SCC approximation to the original curve. For example, the code for the curve in Fig. 12.6(e) is 0.12, 0.20, 0.21, 0.11, −0.11, −0.12, −0.21, −0.22, −0.24, −0.28, −0.28, −0.31, −0.30. The accuracy of the slope changes defined in Fig. 12.6(d) is 10^{-2}, resulting in an "alphabet" of 199 possible symbols (slope changes). The accuracy can be changed, of course. For instance, and accuracy of 10^{-1} produces an alphabet of 19 symbols (see Problem 12.9). Unlike a Freeman code, there is no guarantee that the last point of the coded curve will coincide with the last point of the curve itself. However, shortening the line



a b c d e

FIGURE 12.6 (a) An open curve. (b) A straight-line segment. (c) Traversing the curve using circumferences to determine slope changes; the dot is the origin (starting point). (d) Range of slope changes in the open interval $(-1, 1)$ (the arrow in the center of the chart indicates direction of travel). There can be ten subintervals between the slope numbers shown. (e) Resulting coded curve showing its corresponding numerical sequence of slope changes. (Courtesy of Professor Ernesto Bribiesca, IIMAS-UNAM, Mexico.)

length and/or increasing angle resolution often resolves the problem, because the results of computations are rounded to the nearest integer (remember we work with integer coordinates).

The *inverse* of an SCC is another chain of the same length, obtained by reversing the order of the symbols and their signs. The *mirror image* of a chain is obtained by starting at the origin and reversing the signs of the symbols. Finally, we point out that the preceding discussion is directly applicable to closed curves. Curve following would start at an arbitrary point (for example, the uppermost-leftmost point of the curve) and proceed in a clockwise or counterclockwise direction, stopping when the starting point is reached. We will illustrate an use of SSCs in Example 12.6.

BOUNDARY APPROXIMATIONS USING MINIMUM-PERIMETER POLYGONS

For an open curve, the number of segments of an exact polygonal approximation is equal to the number of points minus 1.

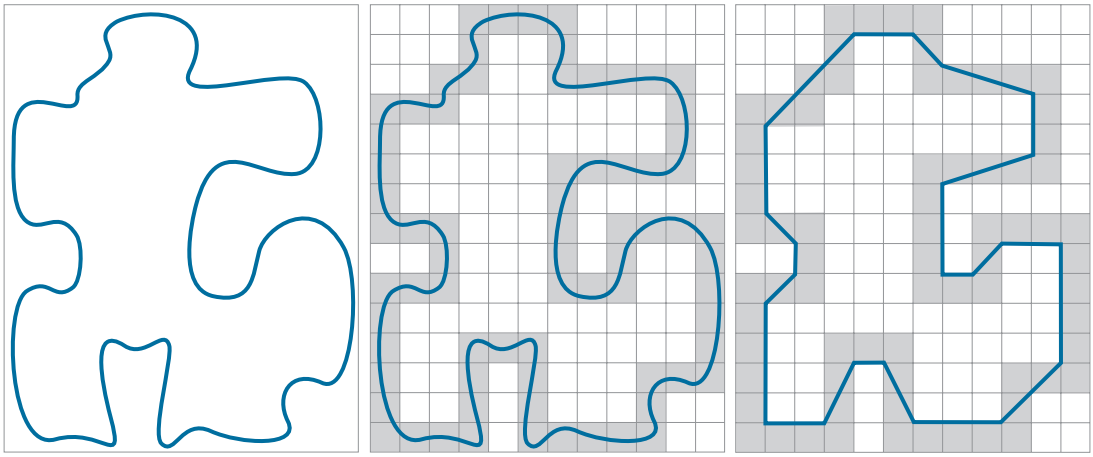
A digital boundary can be approximated with arbitrary accuracy by a polygon. For a closed curve, the approximation becomes exact when the number of segments of the polygon is equal to the number of points in the boundary, so each pair of adjacent points defines a segment of the polygon. The goal of a polygonal approximation is to capture the essence of the shape in a given boundary using the fewest possible number of segments. Generally, this problem is not trivial, and can turn into a time-consuming iterative search. However, approximation techniques of modest complexity are well suited for image-processing tasks. Among these, one of the most powerful is representing a boundary by a *minimum-perimeter polygon* (MPP), as defined in the following discussion.

Foundation

An intuitive approach for computing MPPs is to enclose a boundary [see Fig. 12.7(a)] by a set of concatenated cells, as in Fig. 12.7(b). Think of the boundary as a rubber band contained in the gray cells in Fig. 12.7(b). As it is allowed to shrink, the rubber band will be constrained by the vertices of the inner and outer walls of the region of the gray cells. Ultimately, this shrinking produces the shape of a polygon of minimum perimeter (with respect to this geometrical arrangement) that circumscribes the region enclosed by the cell strip, as in Fig. 12.7(c). Note in this figure that all the vertices of the MPP coincide with corners of either the inner or the outer wall.

The size of the cells determines the accuracy of the polygonal approximation. In the limit, if the size of each (square) cell corresponds to a pixel in the boundary, the maximum error in each cell between the boundary and the MPP approximation would be $\sqrt{2}d$, where d is the minimum possible distance between pixels (i.e., the distance between pixels established by the resolution of the original sampled boundary). This error can be reduced in half by forcing each cell in the polygonal approximation to be centered on its corresponding pixel in the original boundary. The objective is to use the largest possible cell size acceptable in a given application, thus producing MPPs with the fewest number of vertices. Our objective in this section is to formulate a procedure for finding these MPP vertices.

The cellular approach just described reduces the shape of the object enclosed by the original boundary, to the area circumscribed by the gray walls in Fig. 12.7(b).



a b c

FIGURE 12.7 (a) An object boundary. (b) Boundary enclosed by cells (shaded). (c) Minimum-perimeter polygon obtained by allowing the boundary to shrink. The vertices of the polygon are created by the corners of the inner and outer walls of the gray region.

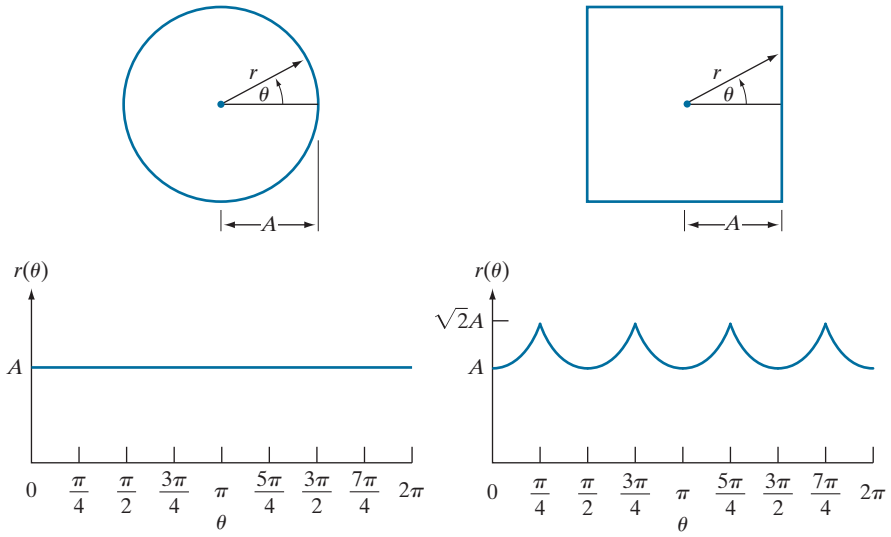
A convex vertex is the center point of a triplet of points that define an angle in the range $0° < \theta < 180°$. Similarly, angles of a concave vertex are in the range $180° < \theta < 360°$. An angle of 180° defines a *degenerate vertex* (i.e., segment of a straight line), which cannot be an MPP-vertex.

Figure 12.8(a) shows this shape in dark gray. Suppose that we traverse the boundary of the dark gray region in a *counterclockwise* direction. Every turn encountered in the traversal will be either a convex or a concave vertex (the angle of a vertex is defined as an *interior* angle of the boundary at that vertex). Convex and concave vertices are shown, respectively, as white and blue dots in Fig. 12.8(b). Note that these vertices are the vertices of the inner wall of the light-gray bounding region in Fig. 12.8(b), and that every concave (blue) vertex in the dark gray region has a corresponding concave "mirror" vertex in the light gray wall, located diagonally opposite the vertex. Figure 12.8(c) shows the mirrors of all the concave vertices, with the MPP from Fig. 12.7(c) superimposed for reference. We see that the vertices of the MPP coincide either with convex vertices in the inner wall (white dots) or with the mirrors of the concave vertices (blue dots) in the outer wall. Only convex vertices of the inner wall and concave vertices of the outer wall can be vertices of the MPP. Thus, our algorithm needs to focus attention only on those vertices.

MPP Algorithm

The set of cells enclosing a digital boundary [e.g., the gray cells in Fig. 12.7(b)] is called a *cellular complex*. We assume the cellular complexes to be *simply connected*, in the sense the boundaries they enclose are not self-intersecting. Based on this assumption, and letting *white* (W) denote convex vertices, and *blue* (B) denote mirrored concave vertices, we state the following observations:

1. The MPP bounded by a simply connected cellular complex is not self-intersecting.
2. Every *convex* vertex of the MPP is a W vertex, but not every W vertex of a boundary is a vertex of the MPP.



a b

FIGURE 12.10 Distance-versus-angle signatures. In (a), $r(\theta)$ is constant. In (b), the signature consists of repetitions of the pattern $r(\theta) = A \sec\theta$ for $0 \le \theta \le \pi/4$, and $r(\theta) = A \csc\theta$ for $\pi/4 < \theta \le \pi/2$.

tangent-angle values. Because a histogram is a measure of the concentration of values, the slope density function responds strongly to sections of the boundary with constant tangent angles (straight or nearly straight segments) and has deep valleys in sections producing rapidly varying angles (corners or other sharp inflections).

EXAMPLE 12.4: Signatures of two regions.

Figures 12.11(a) and (d) show two binary objects, and Figs. 12.11(b) and (e) are their boundaries. The corresponding $r(\theta)$ signatures in Figs. 12.11(c) and (f) range from 0° to 360° in increments of 1°. The number of prominent peaks in the signatures is sufficient to differentiate between the shapes of the two objects.

SKELETONS, MEDIAL AXES, AND DISTANCE TRANSFORMS

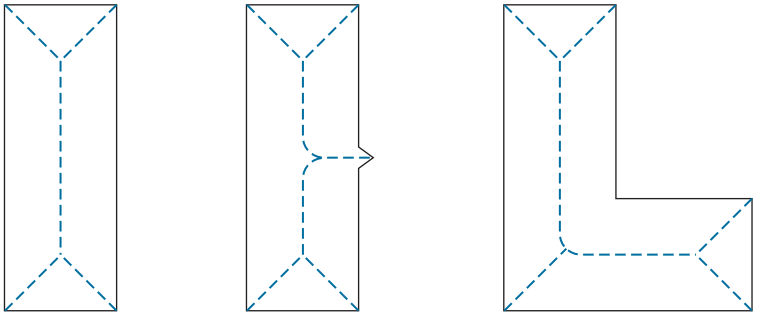
Like boundaries, skeletons are related to the shape of a region. Skeletons can be computed from a boundary by filling the area enclosed by the boundary with foreground values, and treating the result as a binary region. In other words, a skeleton is computed using the coordinates of points in the entire region, including its boundary. The idea is to reduce a region to a tree or graph by computing its skeleton. As we explained in Section 9.5 (see Fig. 9.25), the *skeleton* of a region is the set of points in the region that are equidistant from the border of the region.

As is true of thinning, the MAT is highly susceptible to boundary and internal region irregularities, so smoothing and other preprocessing steps generally are required to obtain a clean a binary image.

The skeleton is obtained using one of two principal approaches: (1) by successively thinning the region (e.g., using morphological erosion) while preserving end points and line connectivity (this is called *topology-preserving* thinning); or (2) by computing the *medial axis* of the region via an efficient implementation of the *medial axis transform* (MAT) proposed by Blum [1967]. We discussed thinning in Section 9.5. The MAT of a region R with border B is as follows: For each point p in R, we find its closest neighbor in B. If p has more than one such neighbor, it is said

a b c

FIGURE 12.12 Medial axes (dashed) of three simple regions.



pixels to their nearest background (zero) pixels, which constitute the region boundary. Thus, we compute the distance transform of the *complement* of the image, as Figs. 12.13(c) and (d) illustrate. By comparing Figs. 12.13(d) and 12.12(a), we see in the former that the MAT (skeleton) is equivalent to the *ridge* of the distance transform [i.e., the ridge in the image in Fig. 12.13(d)]. This ridge is the set of *local maxima* [shown bold in Fig. 12.13(d)]. Figures 12.13(e) and (f) show the same effect on a larger (414×708) binary image.

Finding approaches for computing the distance transform efficiently has been a topic of research for many years. Numerous approaches exist that can compute the distance transform with linear time complexity, $O(K)$, for a binary image with K pixels. For example, the algorithm by Maurer et al. [2003] not only can compute the distance transform in $O(K)$, it can compute it in $O(K/P)$ using P processors.

a b
c d
e f

FIGURE 12.13 (a) A small image and (b) its distance transform. Note that all 1-valued pixels in (a) have corresponding 0's in (b). (c) A small image, and (d) the distance transform of its *complement*. (e) A larger image, and (f) the distance transform of its complement. The Euclidian distance was used throughout.

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

1.41	1	1	1	1.41
1	0	0	0	1
1	0	0	0	1
1.41	1	1	1	1.41

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0
0	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	**1**	1	1	1	1	1	**1**	0
0	1	**2**	2	2	2	**2**	1	0
0	1	2	**3**	**3**	**3**	2	1	0
0	1	**2**	2	2	2	**2**	1	0
0	**1**	1	1	1	1	1	**1**	0
0	0	0	0	0	0	0	0	0

The value of this descriptor is 1 for a circle (its maximum value) and $\pi/4$ for a square. Note that these two measures are independent of size, orientation, and translation. Another measure based on a circle is the *effective diameter*:

$$d_e = 2\sqrt{\frac{A}{\pi}} \tag{12-20}$$

This is the diameter of a circle having the same area, A, as the region being processed. This measure is neither dimensionless nor independent of region size, but it is independent of orientation and translation. It can be normalized for size and made dimensionless by dividing it by the largest diameter expected in a given application.

In a manner analogous to the way we defined compactness and circularity relative to a circle, we define the *eccentricity* of a region relative to an ellipse as the eccentricity of an ellipse that has the same second central moments as the region. For 1-D, the second central moment is the variance which, for discrete variables, we estimate using Eq. (2-114). For 2-D discrete data, we have to consider the variance of each variable as well as the covariance between them. These are the components of the covariance matrix, which is estimated from samples using Eq. (2-130), with the samples in this case being 2-D vectors representing the coordinates of the data.

Figure 12.21(a) shows an ellipse in *standard form* (i.e., an ellipse whose major and minor axes are aligned with the coordinate axes). The eccentricity of such an ellipse is defined as the ratio of the distance between foci ($2c$ in Fig. 12.21), and the length of its major axis ($2a$), which gives the ratio $2c/2a = c/a$. That is,

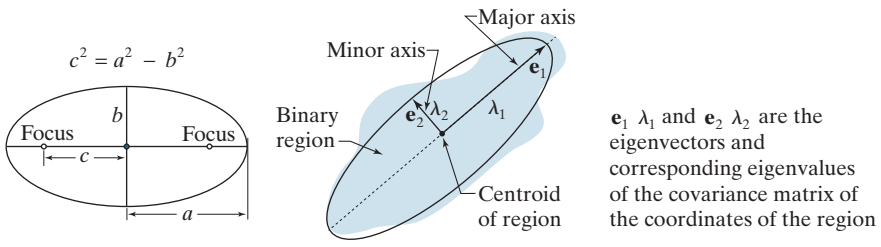
$$\text{eccentricity} = \frac{c}{a} = \frac{\sqrt{a^2 - b^2}}{a} = \sqrt{1 - (b/a)^2} \quad a \geq b$$

However, we are interested in the eccentricity of an ellipse that has the same second central moments as a given 2-D region, which means that our ellipses can have arbitrary orientations. Intuitively, what we are trying to do is approximate our 2-D data by an elliptical region whose axes are aligned with the principal axes of the data, as Fig. 12.21(b) illustrates. As you will learn in Section 12.5 (see Example 12.17), the principal axes are the eigenvectors of the covariance matrix, $\mathbf{C}$, of the data, which is given by:

$$\mathbf{C} = \frac{1}{K-1}\sum_{k=1}^{K}(\mathbf{z}_k - \overline{\mathbf{z}})(\mathbf{z}_k - \overline{\mathbf{z}})^T \tag{12-21}$$

a b

FIGURE 12.21
(a) An ellipse in standard form. (b) An ellipse approximating a region in arbitrary orientation.



$\mathbf{e}_1$ λ_1 and $\mathbf{e}_2$ λ_2 are the eigenvectors and corresponding eigenvalues of the covariance matrix of the coordinates of the region

where $\mathbf{z}_k$ is a 2-D vector whose elements are the two spatial coordinates of a point in the region, K is the total number of points, and $\overline{\mathbf{z}}$ is the mean vector:

$$\overline{\mathbf{z}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_k \tag{12-22}$$

The main diagonal elements of $\mathbf{C}$ are the variances of the coordinate values of the points in the region, and the off-diagonal elements are their covariances (see the discussion on the multivariate Gaussian density in Section 2.6 and Example 2.22).

An ellipse oriented in the same direction as the principal axes of the region can be interpreted as the intersection of a 2-D Gaussian function with the xy-plane. The orientation of the axes of the ellipse are also in the direction of the eigenvectors of the covariance matrix, and the distances from the center of the ellipse to its intersection with its major and minor axes is equal to the largest and smallest eigenvalues of the covariance matrix, respectively, as Fig. 12.21(b) shows. With reference to Fig. 12.21, and the equation of its eccentricity given above, we see by analogy that the eccentricity of an ellipse with the same second moments as the region is given by

$$\begin{aligned}\text{eccentricity} &= \frac{\sqrt{\lambda_2^2 - \lambda_1^2}}{\lambda_2} \\ &= \sqrt{1 - (\lambda_1/\lambda_2)^2} \qquad \lambda_2 \geq \lambda_1\end{aligned} \tag{12-23}$$

For circular regions, $\lambda_1 = \lambda_2$ and the eccentricity is 0. For a line, $\lambda_1 = 0$ and the eccentricity is 1. Thus, values of this descriptor are in the range $[0,1]$.

EXAMPLE 12.9: Comparison of feature descriptors.

Figure 12.22 shows values of the preceding descriptors for several region shapes. None of the descriptors for the circle was exactly equal to its theoretical value because digitizing a circle introduces error into the computation, and because we approximated the length of a boundary as its number of elements. The eccentricity of the square did have an exact value of 0, because a square with no rotation aligns perfectly with the sampling grid. The other two descriptors for the square were close to their theoretical values also.

The values listed in the first two rows of Fig. 12.22 carry the same information. For example, we can tell that the star is less compact and less circular than the other shapes. Similarly, it is easy to tell from the numbers listed that the teardrop region has by far the largest eccentricity, but it is harder to differentiate from the other shapes using compactness or circularity.

As we discussed in Section 12.1, feature descriptors typically are arranged in the form of feature vectors for subsequent processing. Figure 12.23 shows the feature space for the descriptors in Fig. 12.22.

a b c d

FIGURE 12.22 Compactness, circularity, and eccentricity of some simple binary regions.

Descriptor				
Compactness	10.1701	42.2442	15.9836	13.2308
Circularity	1.2356	0.2975	0.7862	0.9478
Eccentricity	0.0411	0.0636	0	0.8117

MOMENT INVARIANTS

The 2-D *moment* of order $(p+q)$ of an $M \times N$ digital image, $f(x,y)$, is defined as

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y) \tag{12-34}$$

where $p = 0,1,2,\ldots$ and $q = 0,1,2,\ldots$ are integers. The corresponding *central moment* of order $(p+q)$ is defined as

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x-\bar{x})^p (y-\bar{y})^q f(x,y) \tag{12-35}$$

for $p = 0,1,2,\ldots$ and $q = 0,1,2,\ldots$, where

$$\bar{x} = \frac{m_{10}}{m_{00}} \text{ and } \bar{y} = \frac{m_{01}}{m_{00}} \tag{12-36}$$

The *normalized central moment* of order $(p+q)$, denoted η_{pq}, is defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{12-37}$$

where

$$\gamma = \frac{p+q}{2} + 1 \tag{12-38}$$

for $p+q = 2,3,\ldots$. A set of seven, 2-D *moment invariants* can be derived from the second and third normalized central moments:[†]

$$\phi_1 = \eta_{20} + \eta_{02} \tag{12-39}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{12-40}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{12-41}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{12-42}$$

[†]Derivation of these results requires concepts that are beyond the scope of this discussion. The book by Bell [1965] and the paper by Hu [1962] contain detailed discussions of these concepts. For generating moment invariants of an order higher than seven, see Flusser [2000]. Moment invariants can be generalized to n dimensions (see Mamistvalov [1998]).

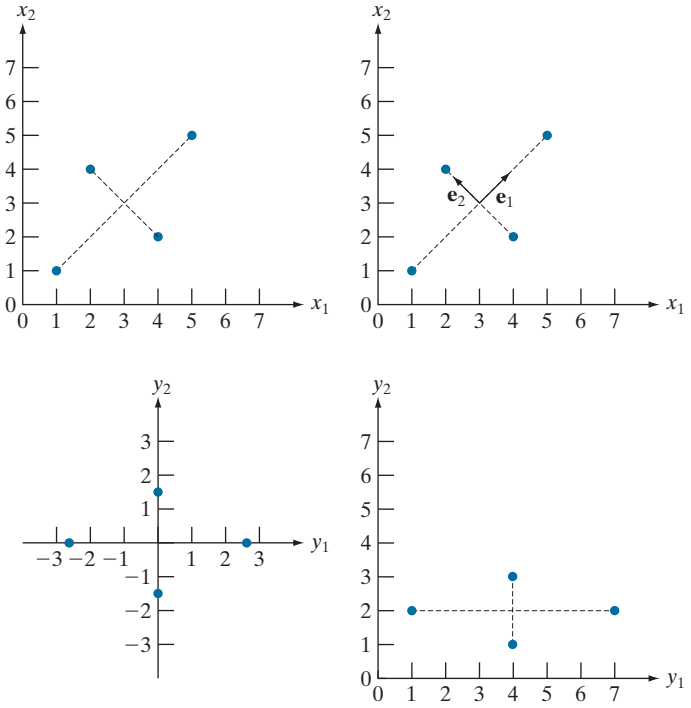


FIGURE 12.44 A manual example. (a) Original points. (b) Eigenvectors of the covariance matrix of the points in (a). (c) Transformed points obtained using Eq. (12-49). (d) Points from (c), rounded and translated so that all coordinate values are integers greater than 0. The dashed lines are included to facilitate viewing. They are not part of the data.

The state of the art in image processing is such that as the complexity of the task increases, the number of techniques suitable for addressing those tasks decreases. This is particularly true when dealing with feature descriptors applicable to entire images that are members of a large family of images. In this section, we discuss two of the principal feature detection methods currently being used for this purpose. One is based on detecting corners, and the other works with entire regions in an image. Then, in Section 12.7 we present a feature detection and description approach designed specifically to work with these types of features.

The discussion in Sections 13.5 through 13.7 dealing with neural networks is also important in terms of processing large numbers of entire images for the purpose of characterizing their content.

THE HARRIS-STEPHENS CORNER DETECTOR

Our use the term "corner" is broader than just 90° corners; it refers to features that are "corner-like."

Intuitively, we think of a corner as a rapid change of direction in a curve. Corners are highly effective features because they are distinctive and reasonably invariant to viewpoint. Because of these characteristics, corners are used routinely for matching image features in applications such as tracking for autonomous navigation, stereo machine vision algorithms, and image database queries.

In this section, we discuss an algorithm for corner detection formulated by Harris and Stephens [1988]. The idea behind the *Harris-Stephens (HS) corner detector* is illustrated in Fig. 12.45. The basic approach is this: Corners are detected by running a small window over an image, as we did in Chapter 3 for spatial filtering. The detector window is designed to compute intensity changes. We are interested in three scenarios: (1) Areas of zero (or small) intensity changes in all directions, which

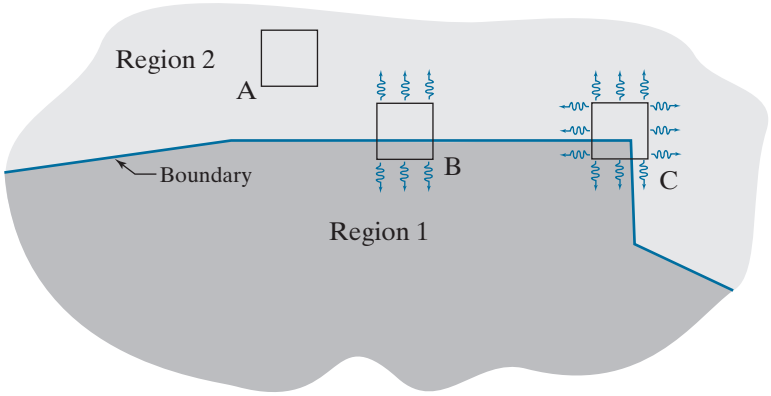


FIGURE 12.45 Illustration of how the Harris-Stephens corner detector operates in the three types of sub-regions indicated by A (flat), B (edge), and C (corner). The wiggly arrows indicate graphically a directional response in the detector as it moves in the three areas shown.

happens when the window is located in a constant (or nearly constant) region, as in location A in Fig. 12.45; (2) areas of changes in one direction but no (or small) changes in the orthogonal direction, which this happens when the window spans a boundary between two regions, as in location B; and (3) areas of significant changes in all directions, a condition that happens when the window contains a corner (or isolated points), as in location C. The HS corner detector is a mathematical formulation that attempts to differentiate between these three conditions.

A *patch* is the image area spanned by the detector window at any given time.

Let f denote an image, and let $f(s,t)$ denote a patch of the image defined by the values of (s,t). A patch of the same size, but shifted by (x,y), is given by $f(s+x,t+y)$. Then, the weighted sum of squared differences between the two patches is given by

$$C(x,y) = \sum_s \sum_t w(s,t)\left[f(s+x,t+y) - f(s,t)\right]^2 \tag{12-56}$$

where $w(s,t)$ is a weighting function to be discussed shortly. The shifted patch can be approximated by the linear terms of a Taylor expansion

$$f(s+x,t+y) \approx f(s,t) + xf_x(s,t) + yf_y(s,t) \tag{12-57}$$

where $f_x(s,t) = \partial f/\partial x$ and $f_y(s,t) = \partial f/\partial y$, both evaluated at (s,t). We can then write Eq. (12-56) as

$$C(x,y) = \sum_s \sum_t w(s,t)\left[xf_x(s,t) + yf_y(s,t)\right]^2 \tag{12-58}$$

This equation can written in matrix form as

$$C(x,y) = \begin{bmatrix} x & y \end{bmatrix} \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \tag{12-59}$$

a b c
d e f

FIGURE 12.49 600×600 image of a building. (b) Result of applying the HS corner detector with $k = 0.04$ and $T = 0.01$ (the default values in our implementation). Numerous irrelevant corners were detected. (c) Result using $k = 0.249$ and the default value for T. (d) Result using $k = 0.17$ and $T = 0.05$. (e) Result using the default value for k and $T = 0.05$. (f) Result using the default value of k and $T = 0.07$.

As you can see, numerous detection errors occurred (see, for example, the large number of wrong corner detections in the right edge of the building). Increasing k alone had little effect on the over-detection of corners until k was near its maximum value. Using the same values as in Fig. 12.48(c) resulted in the image in 12.49(c), which shows a reduced number of erroneous corners, at the expense of missing numerous important ones in the front of the building. Reducing k to 0.17 and increasing T to 0.05 did a much better job, as Fig. 12.49(d) show. Parameter k did not play a major role in corner detection for the building image. In fact, Figs. 12.49(e) and (f) show essentially the same level of performance obtained by reducing k to its default value of 0.04, and using $T = 0.05$ and $T = 0.07$, respectively.

Finally, Fig. 12.50 shows corner detection on a rotated image. The result in Fig. 12.50(b) was obtained using the same parameters we used in Fig. 12.49(f), showing the relative insensitivity of the method to rotation. Figures 12.49(f) and 12.50(b) show detection of at least one corner in every major structural feature of the image, such as the front door, all the windows, and the corners that define the apex of the facade. For matching purposes, these are excellent results.

a b

FIGURE 12.50
(a) Image rotated 5°. (b) Corners detected using the parameters used to obtain Fig. 12.49(f).

MAXIMALLY STABLE EXTREMAL REGIONS (MSERs)

The Harris-Stephens corner detector discussed in the previous section is useful in applications characterized by sharp transitions of intensities, such as the intersection of straight edges, that result in corner-like features in an image. Conversely, the maximally stable extremal regions (MSERs) introduced by Matas et al. [2002] are more "blob" oriented. As with the HS corner detector, MSERs are intended to yield whole image features for the purpose of establishing correspondence between two or more images.

We know from Fig. 2.18 that a grayscale image can be viewed as a topographic map, with the xy-axes representing spatial coordinates, and the z-axis representing intensities. Imagine that we start thresholding an 8-bit grayscale image one intensity level at a time. The result of each thresholding is a binary image in which we show the pixels at or above the threshold in white, and the pixels below the threshold as black. When the threshold, T, is 0, the result is a white image (all pixel values are at or above 0). As we start increasing T in increments of one intensity level, we will begin to see black components in the resulting binary images. These correspond to local minima in the topographic map view of the image. These black regions may begin to grow and merge, but they never get smaller from image to image. Finally, when we reach $T = 255$, the resulting image will be black (there are no pixel values above this level). Because each stage of thresholding results in a binary image, there will be one or more connected components of white pixels in each image. The set of all such components resulting from all thresholdings is the set of *extremal regions*. Extremal regions that do not change size (number of pixels) appreciably over a range of threshold values are called *maximally stable extremal regions*.

As you will see shortly, the procedure just discussed can be cast in the form of a rooted, connected tree called a *component tree*, where each level of the tree corresponds to a value of the threshold discussed in the previous paragraph. Each node of this tree represents an extremal region, R, defined as

Remember, $\forall$ means "for any," $\in$ means "belonging to," and a colon, :, is used to mean "it is true that."

$$\forall p \in R \text{ and } \forall q \in \text{boundary}(R) : I(p) > I(q) \tag{12-64}$$

a b
c d e

FIGURE 12.53 (a) Building image of size 600×600 pixels. (b) Image smoothed using a 5×5 box kernel. (c) and (d) MSERs detected using $T = 0$, $\Delta T = 10$, and MSER size range between 10,000 and 30,000 pixels, corresponding approximately to 3% and 8% of the area of the image. (e) Composite image.

of the original area, we reduced the valid MSER range by one-fourth to 2500–7500 pixels. Other than these changes, we used the same parameters as in Fig. 12.53. Figure 12.55(c) shows the resulting MSER. As you can see, this figure is quite close to the full-size result in Fig. 12.53(e).

12.7 SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

SIFT is an algorithm developed by Lowe [2004] for extracting invariant features from an image. It is called a *transform* because it transforms image data into scale-invariant coordinates relative to local image features. SIFT is by far the most complex feature detection and description approach we discuss in this chapter.

As you progress though this section, you will notice the use of a significant number of experimentally determined parameters. Thus, unlike most of the formulations of individual approaches we have discussed thus far, SIFT is strongly heuristic. This is a consequence of the fact that our current knowledge is insufficient to tell us how

SCALE SPACE

The first stage of the SIFT algorithm is to find image locations that are invariant to scale change. This is achieved by searching for stable features across all possible scales, using a function of scale known as *scale space*, which is a multi-scale representation suitable for handling image structures at different scales in a consistent manner. The idea is to have a formalism for handling the fact that objects in unconstrained scenes will appear in different ways, depending on the scale at which images are captured. Because these scales may not be known beforehand, a reasonable approach is to work with all relevant scales simultaneously. Scale space represents an image as a one-parameter *family* of smoothed images, with the objective of simulating the loss of detail that would occur as the scale of an image decreases. The parameter controlling the smoothing is referred to as the *scale parameter*.

In SIFT, Gaussian kernels are used to implement smoothing, so the scale parameter is the standard deviation. The reason for using Gaussian kernels in based on work performed by Lindberg [1994], who showed that the only smoothing kernel that meets a set of important constraints, such as linearity and shift-invariance, is the Gaussian lowpass kernel. Based on this, the scale space, $L(x, y, \sigma)$, of a grayscale image, $f(x, y)$,[†] is produced by convolving f with a variable-scale Gaussian kernel, $G(x, y, \sigma)$:

As in Chapter 3, "$\star$" indicates spatial convolution.

$$L(x, y, \sigma) = G(x, y, \sigma) \star f(x, y) \tag{12-66}$$

where the scale is controlled by parameter σ, and G is of the form

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2 + y^2)/2\sigma^2} \tag{12-67}$$

The input image $f(x, y)$ is successively convolved with Gaussian kernels having standard deviations $\sigma, k\sigma, k^2\sigma, k^3\sigma, \ldots$ to generate a "stack" of Gaussian-filtered (smoothed) images that are separated by a constant factor k, as shown in the lower left of Fig. 12.56.

SIFT subdivides scale space into *octaves*, with each octave corresponding to a doubling of σ, just as an octave in music theory corresponds to doubling the frequency of a sound signal. SIFT further subdivides each octave into an integer number, s, of intervals, so that an interval of 1 consists of two images, an interval of 2 consists of three images, and so forth. It then follows that the value used in the Gaussian kernel that generates the image corresponding to an octave is $k^s\sigma = 2\sigma$ which means that $k = 2^{1/s}$. For example, for $s = 2$, $k = \sqrt{2}$, and the input image is successively smoothed using standard deviations of σ, $(\sqrt{2})\sigma$, and $(\sqrt{2})^2\sigma$, so that the third image (i.e., the octave image for $s = 2$) in the sequence is filtered using a Gaussian kernel with standard deviation $(\sqrt{2})^2\sigma = 2\sigma$.

[†] Experimental results reported by Lowe [2004] suggest that smoothing the original image using a Gaussian kernel with $\sigma = 0.5$ and then doubling its size by linear (nearest-neighbor) interpolation improves the number of stable features detected by SIFT. This preprocessing step is an integral part of the algorithm. Images are assumed to have values in the range [0, 1].

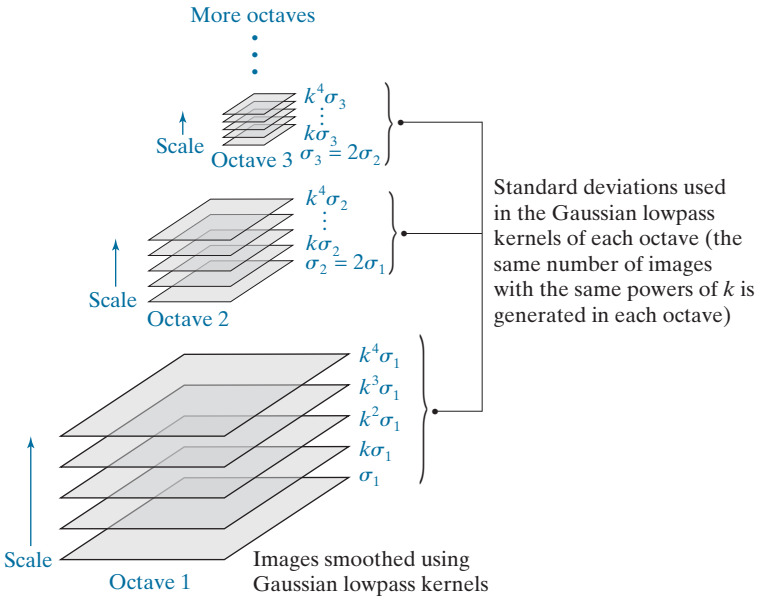


FIGURE 12.56 Scale space, showing three octaves. Because $s = 2$ in this case, each octave has five smoothed images. A Gaussian kernel was used for smoothing, so the space parameter is σ.

The preceding discussion indicates that the number of smoothed images generated in an octave is $s + 1$. However, as you will see in the next section, the smoothed images in scale space are used to compute differences of Gaussians [see Eq. (10-32)] which, in order to cover a full octave, implies that an additional two images past the octave image are required, giving a total of $s + 3$ images. Because the octave image is always the $(s + 1)$th image in the stack (counting from the bottom), it follows that this image is the third image from the top in the expanded sequence of $s + 3$ images. Each octave in Fig. 12.56 contains five images, indicating that $s = 2$ was used in this case.

The *first* image in the *second* octave is formed by downsampling the original image (by skipping every other row and column), and then smoothing it using a kernel with twice the standard deviation used in the first octave (i.e., $\sigma_2 = 2\sigma_1$). Subsequent images in that octave are smoothed using σ_2, with the same sequence of values of k as in the first octave (this is denoted by dots in Fig. 12.56). The same basic procedure is then repeated for subsequent octaves. That is, the *first* image of the *new* octave is formed by: (1) downsampling the original image enough times to achieve half the size of the image in the previous octave, and (2) smoothing the downsampled image with a new standard deviation that is twice the standard deviation of the previous octave. The rest of the images in the new octave are obtained by smoothing the downsampled image with the new standard deviation multiplied by the same sequence of values of k as before.

Instead of repeatedly downsampling the original image, we can carry the previously downsampled image, and downsample it by 2 to obtain the image required for the next octave.

When $k = \sqrt{2}$, we can obtain the first image of a new octave without having to smooth the downsampled image. This is because, for this value of k, the kernel used to smooth the first image of every octave is the same as the kernel used to smooth

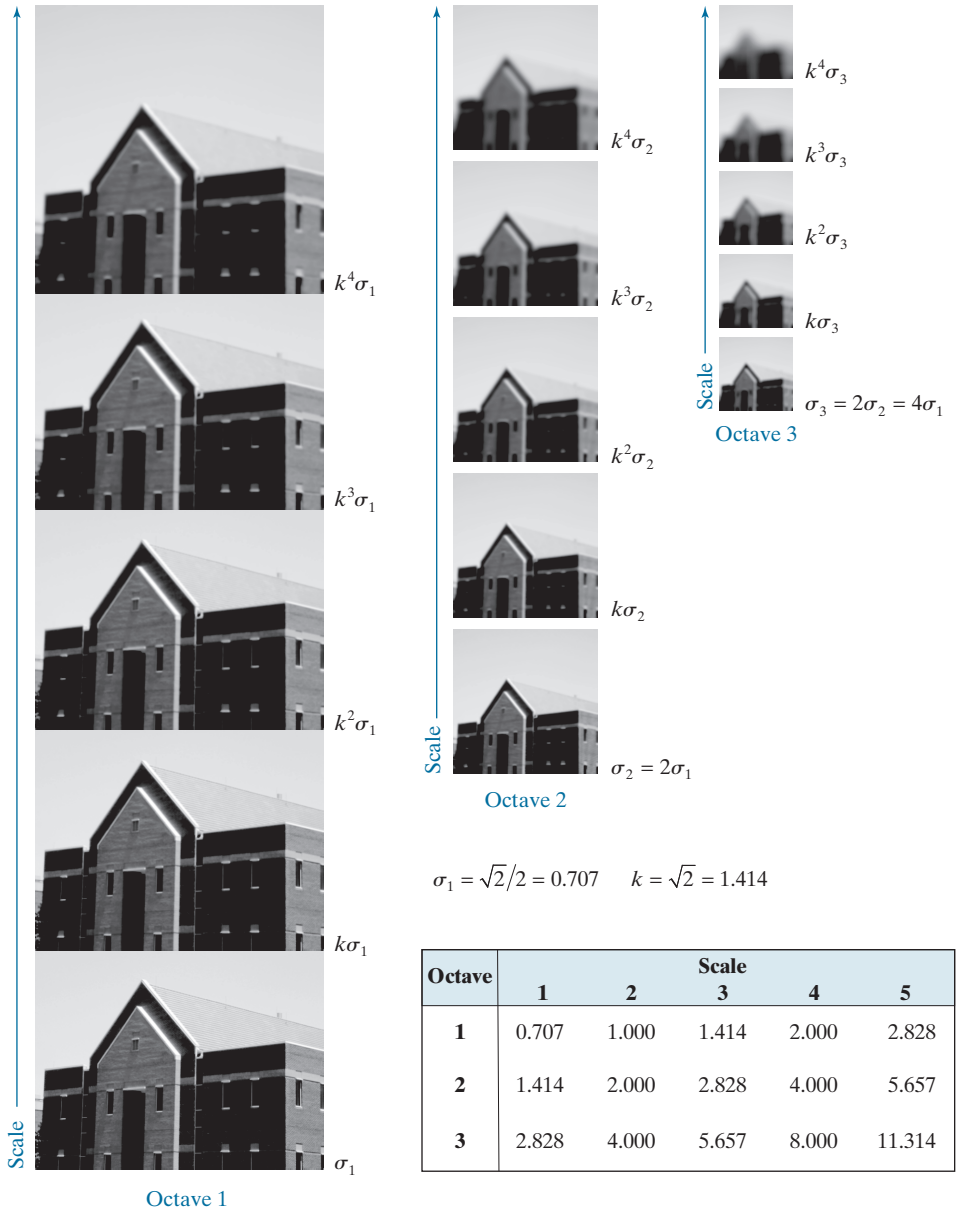


FIGURE 12.57 Illustration using images of the first three octaves of scale space in SIFT. The entries in the table are values of standard deviation used at each scale of each octave. For example the standard deviation used in scale 2 of octave 1 is $k\sigma_1$, which is equal to 1.0. (The images of octave 1 are shown slightly overlapped to fit in the figure space.)

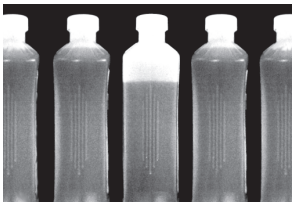
$\sigma_1 = \sqrt{2}/2 = 0.707 \quad k = \sqrt{2} = 1.414$

Octave	Scale				
	1	2	3	4	5
1	0.707	1.000	1.414	2.000	2.828
2	1.414	2.000	2.828	4.000	5.657
3	2.828	4.000	5.657	8.000	11.314

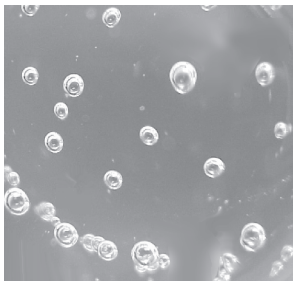
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G \tag{12-70}$$

Therefore, DoGs already have the necessary scaling "built in." The factor $(k - 1)$ is constant over all scales, so it does not influence the process of locating extrema in scale space. Although Eqs. (12-68) and (12-69) are applicable to the first two images

full. As they move along a conveyor line past an automatic filling and capping station, the bottles appear as shown in the following image. A bottle is considered imperfectly filled when the level of the liquid is below the midway point between the bottom of the neck and the shoulder of the bottle. The shoulder is defined as the intersection of the sides and slanted portions of the bottle. The bottles move at a high rate of speed, but the company has an imaging system equipped with an illumination flash front end that effectively stops motion, so you will be given images that look very close to the sample shown here. Based on the material you have learned up to this point, propose a solution for detecting bottles that are not filled properly. State clearly all assumptions that you make and that are likely to impact the solution you propose.



12.42 Having heard about your success with the bottle inspection problem, you are contacted by a fluids company that wishes to automate bubble-counting in certain processes for quality control. The company has solved the imaging problem and can obtain 8-bit images of size 700×700 pixels, such as the one shown in the figure below.



Each image represents an area of 7cm^2. The company wishes to do two things with each image: (1) Determine the ratio of the area occupied by bubbles to the total area of the image; and (2) count the number of distinct bubbles. Based on the material you have learned up to this point, propose a solution to this problem. In your report, state the physical dimensions of the smallest bubble your solution can detect. State clearly all assumptions that you make and that are likely to impact the solution you propose.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

12.1 Boundary tracing.

(a)* Write a function **B = boundaryTracer4e(I,dir)** that traces the boundaries of multiple objects in binary image **I** using 8-connectivity. The background pixels in **I** must be 0. If **dir = 'cw'** the boundary is traced in the clockwise direction (this is the default). If **dir = 'ccw'** the boundary is traced in the counterclockwise direction. In either case, tracing starts at the uppermost-leftmost point in each object [see part (b)]. **B** is a cell array with **P** cells, where **P** is the number of objects in **I**. Each cell of **B** is an $np \times 2$ matrix, each row of which contains the coordinates of one boundary point, and np is the total number of points. The key usefulness of this algorithm is that it returns boundary points as an ordered sequence in a specified direction.

(b) Write a function **ulp = uppermostLeftmost4e(b)** that finds the uppermost-leftmost point in a boundary whose 2-D coordinates are given in $np \times 2$ array **b**. In the output, **ulp** is a 1×2 vector containing the (x, y) coordinates of the uppermost-leftmost point in **b**. As discussed in Section 12.3, the uppermost-leftmost point has some very important properties that are useful for setting the starting point of boundary processing algorithms.

13 Image Pattern Classification

One of the most interesting aspects of the world is that it can be considered to be made up of patterns.

A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements.

Norbert Wiener

Preview

We conclude our coverage of digital image processing with an introduction to techniques for image pattern classification. The approaches developed in this chapter are divided into three principal categories: classification by *prototype matching*, classification based on an *optimal statistical* formulation, and classification based on *neural networks*. The first two approaches are used extensively in applications in which the nature of the data is well understood, leading to an effective pairing of features and classifier design. These approaches often rely on a great deal of engineering to define features and elements of a classifier. Approaches based on neural networks rely less on such knowledge, and lend themselves well to applications in which pattern class characteristics (e.g., features) are learned by the system, rather than being specified a priori by a human designer. The focus of the material in this chapter is on principles, and on how they apply specifically in image pattern classification.

Upon completion of this chapter, readers should:

- Understand the meaning of patterns and pattern classes, and how they relate to digital image processing.
- Be familiar with the basics of minimum-distance classification.
- Know how to apply image correlation techniques for template matching.
- Understand the concept of string matching.
- Be familiar with Bayes classifiers.
- Understand perceptrons and their history.
- Be familiar with the concept of learning from training samples.
- Understand neural network architectures.
- Be familiar with the concept of deep learning in fully connected and deep convolutional neural networks. In particular, be familiar with the importance of the latter in digital image processing.

13.1 BACKGROUND

Humans possess the most sophisticated pattern recognition capabilities in the known biological world. By contrast, the capabilities of current recognition machines pale in comparison with tasks humans perform routinely, from being able to interpret the meaning of complex images, to our ability for generalizing knowledge stored in our brains. But recognition machines play an important, sometimes even crucial role in everyday life. Imagine what modern life would be like without machines that read barcodes, process bank checks, inspect the quality of manufactured products, read fingerprints, sort mail, and recognize speech.

In image pattern recognition, we think of a *pattern* as a spatial arrangement of features. A *pattern class* is a set of patterns that share some common properties. Pattern recognition by machine encompasses techniques for automatically assigning patterns to their respective classes. That is, given a pattern or sets of patterns whose class is unknown, the job of a pattern recognition system is to assign a class label to each of its input patterns.

There are four main stages involved in recognition: (1) sensing, (2) preprocessing, (3) feature extraction, and (4) classification. In terms of image processing, sensing is concerned with generating signals in a spatial (2-D) or higher-dimensional format. We covered numerous aspects of image sensing in Chapter 1. Preprocessing deals with techniques for tasks such as noise reduction, enhancement, restoration, and segmentation, as discussed in earlier chapters. You learned about feature extraction in Chapters 12. Classification, the focus of this chapter, deals with using a set of features as the basis for assigning class labels to unknown input image patterns.

In the following section, we will discuss three basic approaches used for image pattern classification: (1) classification based on matching unknown patterns against specified prototypes, (2) optimum statistical classifiers, and (3) neural networks. One way to characterize the differences between these approaches is in the level of "engineering" required to transform raw data into formats suitable for computer processing. Ultimately, recognition performance is determined by the discriminative power of the features used.

In classification based on prototypes, the objective is to make the features so unique and easily detectable that classification itself becomes a simple task. A good example of this are bank-check processors, which use stylized font styles to simplify machine processing (we will discuss this application in Section 13.3).

In the second category, classification is cast in decision-theoretic, statistical terms, and the classification approach is based on selecting parameters that can be shown to yield optimum classification performance in a statistical sense. Here, emphasis is placed on both the features used, and the design of the classifier. We will illustrate this approach in Section 13.4 by deriving the Bayes pattern classifier, starting from basic principles.

In the third category, classification is performed using neural networks. As you will learn in Sections 13.5 and 13.6, neural networks can operate using engineered features too, but they have the unique ability of being able to generate, on their own, representations (features) suitable for recognition. These systems can accomplish this using raw data, without the need for engineered features.

One characteristic shared by the preceding three approaches is that they are based on parameters that must be either specified or learned from patterns that represent the recognition problem we want to solve. The patterns can be *labeled*, meaning that we know the class of each pattern, or *unlabeled*, meaning that the data are known to be patterns, but the class of each pattern is unknown. A classic example of labeled data is the character recognition problem, in which a set of character samples is collected and the identity of each character is recorded as a label from the group 0 through 9 and *a* through *z*. An example of unlabeled data is when we are seeking clusters in a data set, with the aim of utilizing the resulting cluster centers as being prototypes of the pattern classes contained in the data.

Because the examples in this chapter are intended to demonstrate basic principles and are not large scale, we dispense with validation and subdivide the pattern data into training and test sets.

When working with a labeled data, a given data set generally is subdivided into three subsets: a *training set*, a v*alidation set*, and a *test set* (a typical subdivision might be 50% training, and 25% each for the validation and test sets). The process by which a training set is used to generate classifier parameters is called *training.* In this mode, a classifier is given the class label of each pattern, the objective being to make adjustments in the parameters if the classifier makes a mistake in identifying the class of the given pattern. At this point, we might be working with several candidate designs. At the end of training, we use the validation set to compare the various designs against a performance objective. Typically, several iterations of training/validation are required to establish the design that comes closest to meeting the desired objective. Once a design has been selected, the final step is to determine how it will perform "in the field." For this, we use the test set, which consists of patterns that the system has never "seen" before. If the training and validation sets are truly representative of the data the system will encounter in practice, the results of training/validation should be close to the performance using the test set. If training/validation results are acceptable, but test results are not, we say that training/validation "over fit" the system parameters to the available data, in which case further work on the system architecture is required. Of course all this assumes that the given data are truly representative of the problem we want to solve, and that the problem in fact can be solved by available technology.

A system that is designed using training data is said to undergo *supervised learning.* If we are working with unlabeled data, the system learns the pattern classes themselves while in an *unsupervised* learning mode. In this chapter, we deal only with supervised learning. As you will see in this and the next chapter, supervised learning covers a broad range of approaches, from applications in which a system learns parameters of features whose form is fixed by a designer, to systems that utilize *deep learning* and large sets of raw data sets to learn, *on their own*, the features required for classification. These systems accomplish this task without a human designer having to specify the features, a priori.

Generally, we associate the concept of deep learning with large sets of data. These ideas are discussed in more detail later in this section and next.

After a brief discussion in the next section of how patterns are formed, and on the nature of patterns classes, we will discuss in Section 13.3 various approaches for prototype-based classification. In Section 13.4, we will start from basic principles and derive the equations of the Bayes classifier, an approach characterized by optimum classification performance on an average basis. We will also discuss supervised training of a Bayes classifier based on the assumption of multivariate Gaussian

distributions. Starting with Section 13.5, we will spend the rest of the chapter discussing neural networks. We will begin Section 13.5 with a brief introduction to perceptrons and some historical facts about machine learning. Then, we will introduce the concept of deep neural networks and derive the equations of backpropagation, the method of choice for training deep neural nets. These networks are well-suited for applications in which input patterns are vectors. In Section 13.6, we will introduce deep convolutional neural networks, which currently are the preferred approach when the system inputs are digital images. After deriving the backpropagation equations used for training convolutional nets, we will give several examples of applications involving classes of images of various complexities. In addition to working directly with image inputs, deep convolutional nets are capable of learning, on their own, image features suitable for classification. This is accomplished starting with raw image data, as opposed to the other classification methods discussed in Sections 13.3 and 13.4, which rely on "engineered" features whose form, as noted earlier, is specified a priori by a human designer.

13.2 PATTERNS AND PATTERN CLASSES

In image pattern classification, the two principal pattern arrangements are quantitative and structural. *Quantitative patterns* are arranged in the form of *pattern vectors*. *Structural patterns* typically are composed of symbols, arranged in the form of *strings*, *trees*, or, less frequently, as *graphs*. Most of the work in this chapter is based on pattern vectors, but we will discuss structural patterns briefly at the end of this section, and give an example at the end of Section 13.3.

PATTERN VECTORS

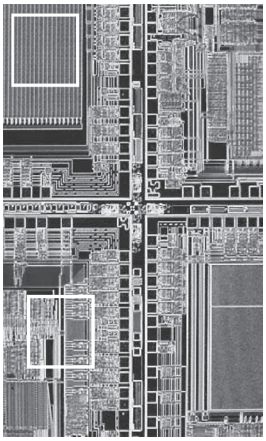
Pattern vectors are represented by lowercase letters, such as $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, and have the form

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{13-1}$$

where each component, x_i, represents the ith feature descriptor, and n is the total number of such descriptors. We can express a vector in the form of a column, as in Eq. (13-1), or in the equivalent row form $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, where T indicates transposition. A pattern vector may be "viewed" as a point in n-dimensional Euclidean space, and a pattern class may be interpreted as a "hypercloud" of points in this *pattern space*. For the purpose of recognition, we like for our pattern classes to be grouped tightly, and as far away from each other as possible.

We discussed linear indexing in Section 2.4 (see Fig. 2.22).

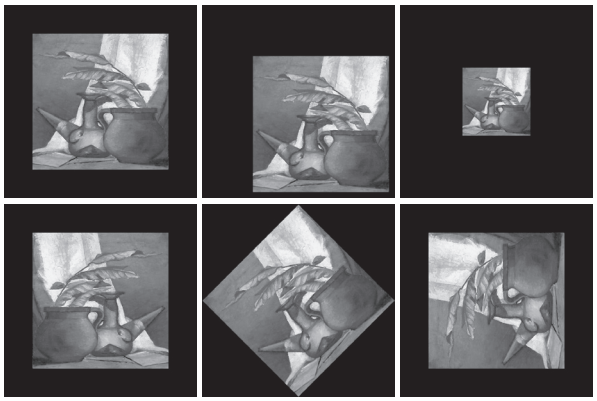
Pattern vectors can be formed directly from image pixel intensities by vectorizing the image using, for example, linear indexing, as in Fig. 13.1. A more common approach is for pattern elements to be features. An early example is the work of Fisher [1936] who, close to a century ago, reported the use of what then was a new



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

x_1 = max probability
x_2 = correlation
x_3 = contrast
x_4 = uniformity
x_5 = homogeneity
x_6 = entropy

FIGURE 13.5 An example of pattern vectors based on properties of subimages. See Table 12.3 for an explanation of the components of **x**.



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \\ \phi_6 \\ \phi_7 \end{bmatrix}$$

The ϕ's are moment invariants

FIGURE 13.6 Feature vectors with components that are invariant to transformations such as rotation, scaling, and translation. The vector components are moment invariants.

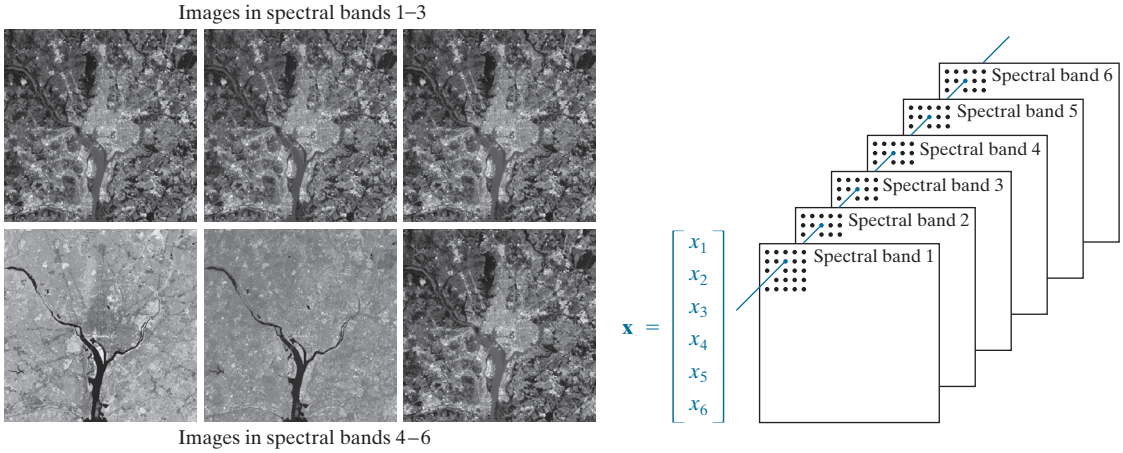
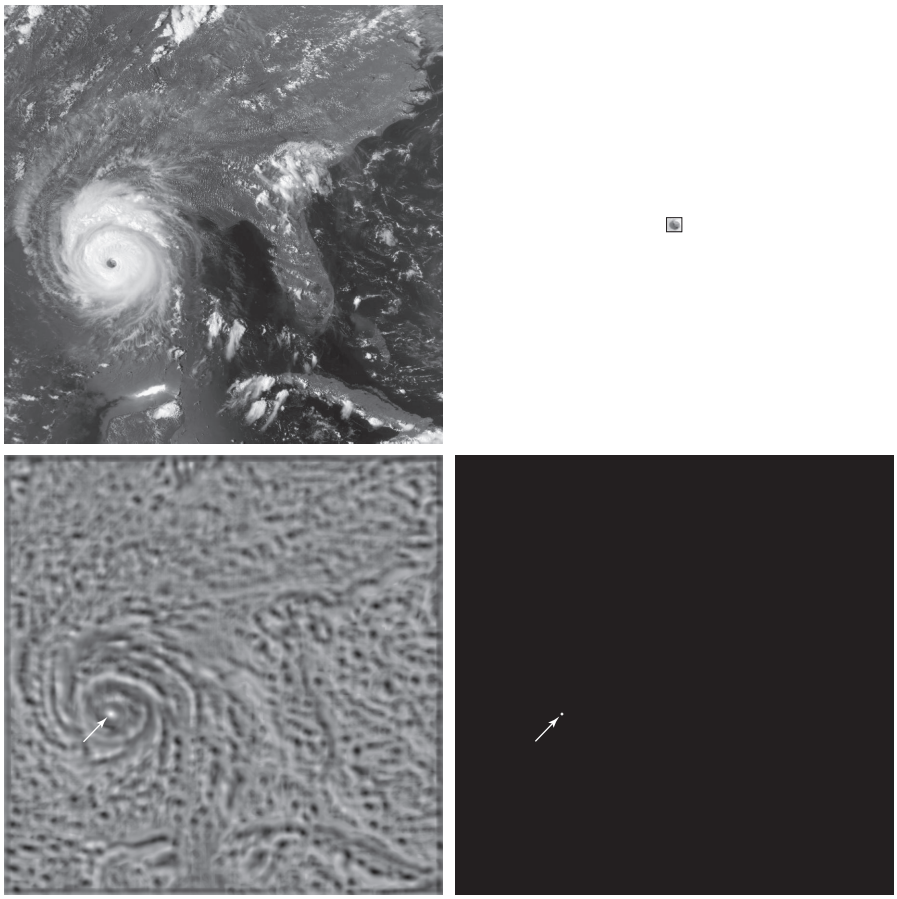


FIGURE 13.7 Pattern (feature) vectors formed by concatenating corresponding pixels from a set of registered images. (Original images courtesy of NASA.)



a b
c d

FIGURE 13.13
(a) 913×913 satellite image of Hurricane Andrew. (b) 31×31 template of the eye of the storm. (c) Correlation coefficient shown as an image (note the brightest point, indicated by an arrow). (d) Location of the best match (identified by the arrow). This point is a single pixel, but its size was enlarged to make it easier to see. (Original image courtesy of NOAA.)

white dot the location of this maximum correlation value (in this case there was a unique match whose maximum value was 1), which we see corresponds closely with the location of the eye in Fig. 13.13(a).

MATCHING SIFT FEATURES

We discussed the scale-invariant feature transform (SIFT) in Section 12.7. SIFT computes a set of invariant features that can be used for matching between known (prototype) and unknown images. The SIFT implementation in Section 12.7 yields 128-dimensional feature vectors for each local region in an image. SIFT performs matching by looking for correspondences between sets of stored feature vector prototypes and feature vectors computed for an unknown image. Because of the large number of features involved, searching for exact matches is computationally intensive. Instead, the approach is to use a best-bin-first method that can identify the nearest neighbors with high probability using only a limited amount of computation (see Lowe [1999], [2004]). The search is further simplified by looking for clusters of potential solutions using the generalized Hough transform proposed by Ballard [1981]. We

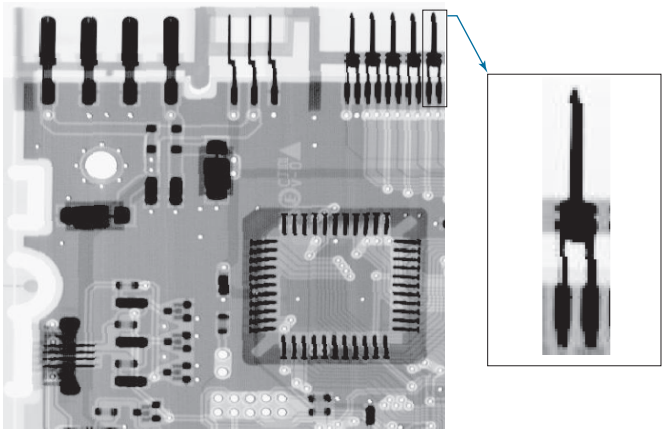


FIGURE 13.14 Circuit board image of size 948×915 pixels, and a subimage of one of the connectors. The subimage is of size 212×128 pixels, shown zoomed on the right for clarity. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

know from the discussion in Section 10.2 that the Hough transform simplifies looking for data patterns by utilizing bins that reduce the level of detail with which we look at a data set. We already discussed the SIFT algorithm in Section 12.7. The focus in this section is to further illustrate the capabilities of SIFT for prototype matching.

Figure 13.14 shows the circuit board image we have used several times before. The small rectangle enclosing the rightmost connector on the top of the large image identifies an area from which an image of the connector was extracted. The small image is shown zoomed for clarity. The sizes of the large and small images are shown in the figure caption. Figure 13.15 shows the keypoints found by SIFT, as explained in Section 12.7. They are visible as faint lines on both images. The zoomed view of the subimage shows them a little clearer. It is important to note that the keypoints for the image and subimage were found independently by SIFT. The large image had 2714 keypoints, and the small image had 35.

Figure 13.16 shows the matches between keypoints found by SIFT. A total of 41 matches were found between the two images. Because there are only 35 keypoints

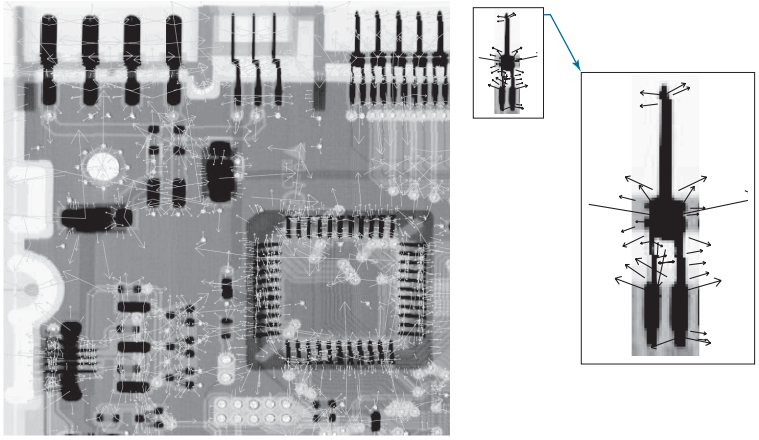


FIGURE 13.15 Keypoints found by SIFT. The large image has 2714 keypoints (visible as faint gray lines). The subimage has 35 keypoints. This is a separate image, and SIFT found its keypoints independently of the large image. The zoomed section is shown for clarity.

13.4 OPTIMUM (BAYES) STATISTICAL CLASSIFIERS

In this section, we develop a probabilistic approach to pattern classification. As is true in most fields that deal with measuring and interpreting physical events, probability considerations become important in pattern recognition because of the randomness under which pattern classes normally are generated. As shown in the following discussion, it is possible to derive a classification approach that is optimal in the sense that, on average, it yields the lowest probability of committing classification errors (see Problem 13.12).

DERIVATION OF THE BAYES CLASSIFIER

The probability that a pattern vector $\mathbf{x}$ comes from class c_i is denoted by $p(c_i/\mathbf{x})$. If the pattern classifier decides that $\mathbf{x}$ came from class c_j when it actually came from c_i it incurs a *loss* (to be defined shortly), denoted by L_{ij}. Because pattern $\mathbf{x}$ may belong to any one of N_c possible classes, the average loss incurred in assigning $\mathbf{x}$ to class c_j is

$$r_j(\mathbf{x}) = \sum_{k=1}^{N_c} L_{kj} p(c_k/\mathbf{x}) \tag{13-16}$$

Quantity $r_j(\mathbf{x})$ is called the *conditional average risk* or *loss* in decision-theory terminology.

We know from Bayes' rule (see Section 2.6) that $p(a/b) = [p(a)p(b/a)]/p(b)$, so we can write Eq. (13-16) as

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{N_c} L_{kj} p(\mathbf{x}/c_k) P(c_k) \tag{13-17}$$

where $p(\mathbf{x}/c_k)$ is the probability density function (PDF) of the patterns from class c_k, and $P(c_k)$ is the probability of occurrence of class c_k (sometimes $P(c_k)$ is referred to as the *a priori*, or simply the *prior*, *probability*). Because $1/p(\mathbf{x})$ is positive and common to all the $r_j(\mathbf{x})$, $j = 1, 2, \ldots, N_c$, it can be dropped from Eq. (13-17) without affecting the relative order of these functions from the smallest to the largest value. The expression for the average loss then reduces to

$$r_j(\mathbf{x}) = \sum_{k=1}^{N_c} L_{kj} p(\mathbf{x}/c_k) P(c_k) \tag{13-18}$$

Given an unknown pattern, the classifier has N_c possible classes from which to choose. If the classifier computes $r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_{N_c}(\mathbf{x})$ for each pattern $\mathbf{x}$ and assigns the pattern to the class with the smallest loss, the total average loss with respect to all decisions will be minimum. The classifier that minimizes the total average loss is called the *Bayes classifier*. This classifier assigns an unknown pattern $\mathbf{x}$ to class c_i if $r_i(\mathbf{x}) < r_j(\mathbf{x})$ for $j = 1, 2, \ldots, N_c$; $j \neq i$. In other words, $\mathbf{x}$ is assigned to class c_i if

$$\sum_{k=1}^{N_c} L_{ki} p(\mathbf{x}/c_k) P(c_k) < \sum_{q=1}^{N_c} L_{qj} p(\mathbf{x}/c_q) P(c_q) \tag{13-19}$$

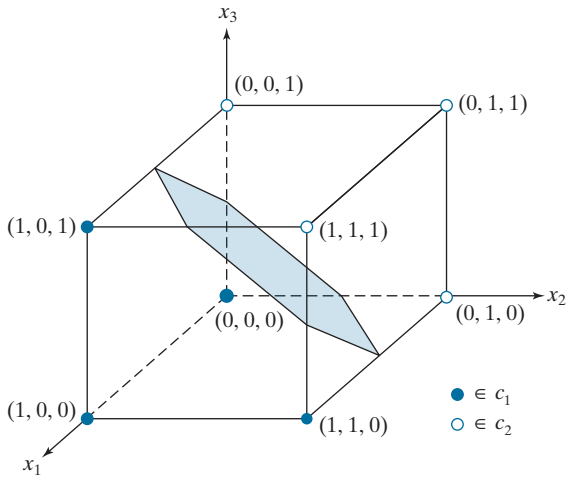


FIGURE 13.20 Two simple pattern classes and the portion of their Bayes decision boundary (shaded) that intersects the cube.

$$\mathbf{C}_1 = \mathbf{C}_2 = \frac{1}{16}\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}$$

The inverse of this matrix is

$$\mathbf{C}_1^{-1} = \mathbf{C}_2^{-1} = \begin{bmatrix} 8 & -4 & -4 \\ -4 & 8 & 4 \\ -4 & 4 & 8 \end{bmatrix}$$

Next, we obtain the decision functions. Equation (13-34) applies because the covariance matrices are equal, and we are assuming that the classes are equally likely:

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{C}^{-1}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{C}^{-1}\mathbf{m}_j$$

Carrying out the vector-matrix expansion, we obtain the two decision functions:

$$d_1(\mathbf{x}) = 4x_1 - 1.5 \quad \text{and} \quad d_2(\mathbf{x}) = -4x_1 + 8x_2 + 8x_3 - 5.5$$

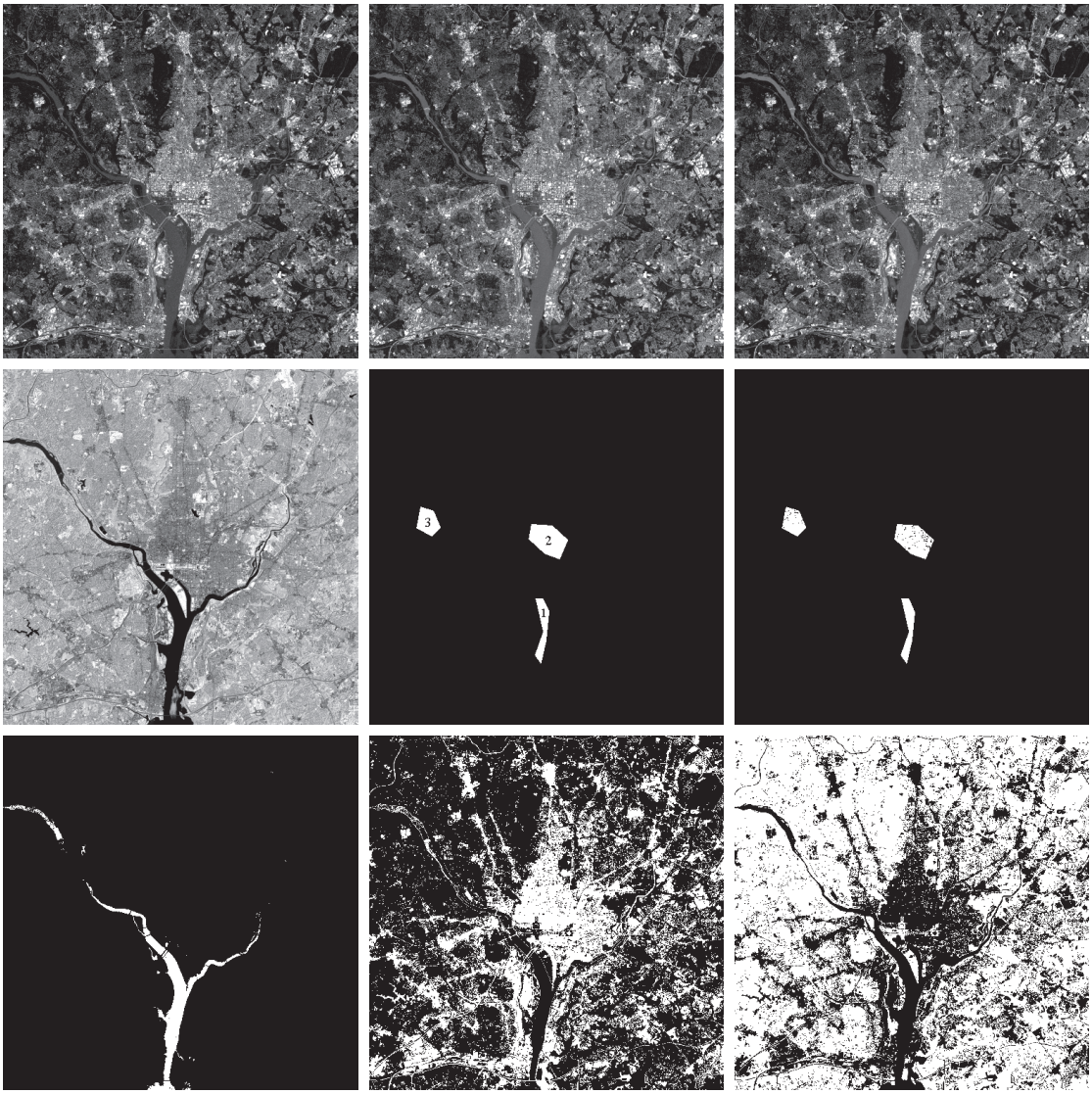
The decision boundary separating the two classes is then

$$d_1(\mathbf{x}) - d_2(\mathbf{x}) = 8x_1 - 8x_2 - 8x_3 + 4 = 0$$

Figure 13.20 shows a section of this planar surface. Note that the classes were separated effectively.

EXAMPLE 13.6: Classification of multispectral data using a Bayes classifier.

As discussed in Sections 1.3 and 12.5, a multispectral scanner responds to selected bands of the electromagnetic energy spectrum, such as the bands: 0.45–0.52, 0.53–0.61, 0.63–0.69, and 0.78–0.90 microns. These ranges are in the visible blue, visible green, visible red, and near infrared bands, respectively. A region on the ground scanned using these multispectral bands produces four digital images of the region,



a b c
d e f
g h i

FIGURE 13.21 Bayes classification of multispectral data. (a)–(d) Images in the visible blue, visible green, visible red, and near infrared wavelength bands. (e) Masks for regions of water (labeled 1), urban development (labeled 2), and vegetation (labeled 3). (f) Results of classification; the black dots denote points classified incorrectly. The other (white) points were classified correctly. (g) All image pixels classified as water (in white). (h) All image pixels classified as urban development (in white). (i) All image pixels classified as vegetation (in white).

TABLE 13.1
Bayes classification of multispectral image data. Classes 1, 2, and 3 are water, urban, and vegetation, respectively.

Training Patterns						Test Patterns					
		Classified into Class						Classified into Class			
Class	No. of Samples	1	2	3	% Correct	Class	No. of Samples	1	2	3	% Correct
1	484	482	2	0	99.6	1	483	478	3	2	98.9
2	933	0	885	48	94.9	2	932	0	880	52	94.4
3	483	0	19	464	96.1	3	482	0	16	466	96.7

have mentioned previously, estimating these densities is not a trivial task. If assumptions have to be made (e.g., as in assuming Gaussian densities), then the degree of optimality achieved in classification depends on how close the assumptions are to reality.

13.5 NEURAL NETWORKS AND DEEP LEARNING

The principal objectives of the material in this section and in Section 13.6 are to present an introduction to deep neural networks, and to derive the equations that are the foundation of deep learning. We will discuss two types of networks. In this section, we focus attention on multilayer, fully connected neural networks, whose inputs are pattern vectors of the form introduced in Section 13.2. In Section 13.6, we will discuss convolutional neural networks, which are capable of accepting images as inputs. We follow the same basic approach in presenting the material in these two sections. That is, we begin by developing the equations that describe how an input is mapped through the networks to generate the outputs that are used to classify that input. Then, we derive the equations of backpropagation, which are the tools used to train both types of networks. We give examples in both sections that illustrate the power of deep neural networks and deep learning for solving complex pattern classification problems.

BACKGROUND

The essence of the material that follows is the use of a multitude of elemental nonlinear computing elements (called *artificial neurons*), organized as networks whose interconnections are similar in some respects to the way in which neurons are interconnected in the visual cortex of mammals. The resulting models are referred to by various names, including *neural networks*, *neurocomputers*, *parallel distributed processing models*, *neuromorphic systems*, *layered self-adaptive networks*, and *connectionist models*. Here, we use the name *neural networks*, or *neural nets* for short. We use these networks as vehicles for adaptively learning the parameters of decision functions via successive presentations of training patterns.

Interest in neural networks dates back to the early 1940s, as exemplified by the work of McCulloch and Pitts [1943], who proposed neuron models in the form of

binary thresholding devices, and stochastic algorithms involving sudden 0–1 and 1–0 changes of states, as the basis for modeling neural systems. Subsequent work by Hebb [1949] was based on mathematical models that attempted to capture the concept of learning by reinforcement or association.

During the mid-1950s and early 1960s, a class of so-called *learning machines* originated by Rosenblatt [1959, 1962] caused a great deal of excitement among researchers and practitioners of pattern recognition. The reason for the interest in these machines, called *perceptrons*, was the development of mathematical proofs showing that perceptrons, when trained with linearly separable training sets (i.e., training sets separable by a hyperplane), would converge to a solution in a finite number of iterative steps. The solution took the form of parameters (coefficients) of hyperplanes that were capable of correctly separating the classes represented by patterns of the training set.

Unfortunately, the expectations following discovery of what appeared to be a well-founded theoretical model of learning soon met with disappointment. The basic perceptron, and some of its generalizations, were inadequate for most pattern recognition tasks of practical significance. Subsequent attempts to extend the power of perceptron-like machines by considering multiple layers of these devices lacked effective training algorithms, such as those that had created interest in the perceptron itself. The state of the field of learning machines in the mid-1960s was summarized by Nilsson [1965]. A few years later, Minsky and Papert [1969] presented a discouraging analysis of the limitation of perceptron-like machines. This view was held as late as the mid-1980s, as evidenced by comments made by Simon [1986]. In this work, originally published in French in 1984, Simon dismisses the perceptron under the heading "Birth and Death of a Myth."

More recent results by Rumelhart, Hinton, and Williams [1986] dealing with the development of new training algorithms for multilayers of perceptron-like units have changed matters considerably. Their basic method, called *backpropagation* (*backprop* for short), provides an effective training method for multilayer networks. Although this training algorithm cannot be shown to converge to a solution in the sense of the proof for the single-layer perceptron, backpropagation is capable of generating results that have revolutionized the field of pattern recognition.

The approaches to pattern recognition we have studied up to this point rely on human-engineered techniques to transform raw data into formats suitable for computer processing. The methods of feature extraction we studied in Chapter 12 are examples of this. Unlike these approaches, neural networks can use backpropagation to automatically learn representations suitable for recognition, starting with raw data. Each layer in the network "refines" the representation into more abstract levels. This type of multilayered learning is commonly referred to as *deep learning*, and this capability is one of the underlying reasons why applications of neural networks have been so successful. As we noted at the beginning of this section, practical implementations of deep learning generally are associated with large data sets.

Of course, these are not "magical" systems that assemble themselves. Human intervention is still required for specifying parameters such as the number of layers, the number of artificial neurons per layer, and various coefficients that are problem

a b

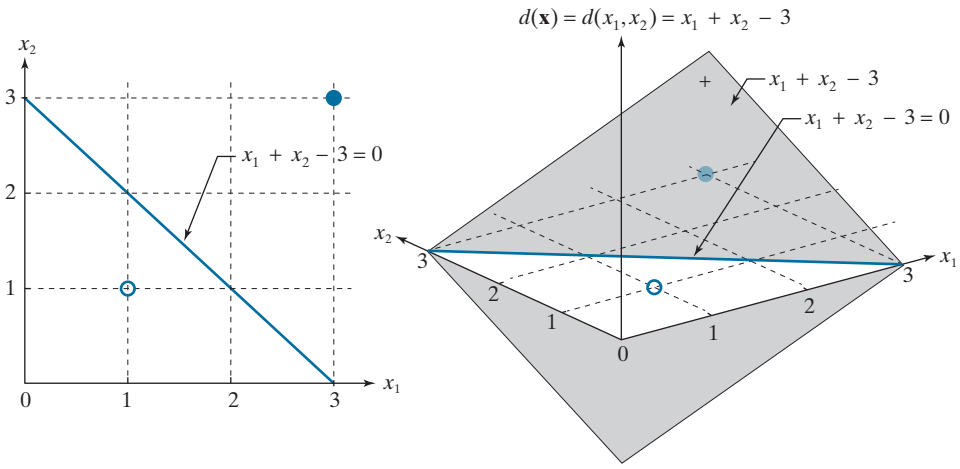


FIGURE 13.24 (a) Segment of the decision boundary learned by the perceptron algorithm. (b) Section of the decision surface. The decision boundary is the intersection of the decision surface with the x_1x_2-plane.

EXAMPLE 13.8: Using the perceptron to classify two sets of iris data measurements.

In Fig. 13.10 we showed a reduced set of the iris database in two dimensions, and mentioned that the only class that was separable from the others is the class of Iris setosa. As another illustration of the perceptron, we now find the full decision boundary between the Iris setosa and the Iris versicolor classes. As we mentioned when discussing Fig. 13.10, these are 4-D data sets. Letting $\alpha = 0.5$, and starting with all parameters equal to zero, the perceptron converged in only four epochs to the solution weight vector $\boldsymbol{w} = [0.65,\ 2.05,\ -2.60,\ -1.10,\ 0.50]^T$, where the last element is w_{n+1}.

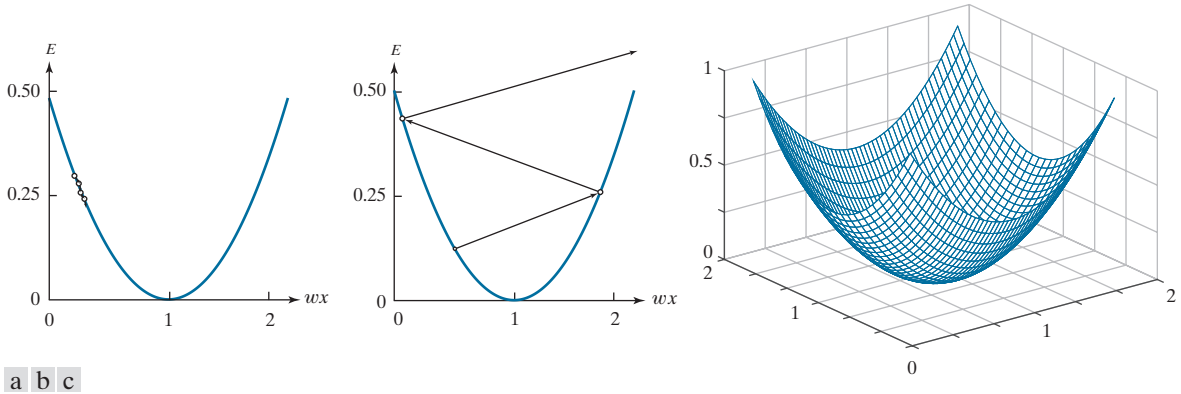
In practice, linearly separable pattern classes are rare, and a significant amount of research effort during the 1960s and 1970s went into developing techniques for dealing with nonseparable pattern classes. With recent advances in neural networks, many of those methods have become items of mere historical interest, and we will not dwell on them here. However, we mention briefly one approach because it is relevant to the discussion of neural networks in the next section. The method is based on minimizing the error between the actual and desired response at any training step.

Let r denote the response we want the perceptron to have for any pattern during training. The output of our perceptron is either +1 or –1, so these are the two possible values that r can have. We want to find the augmented weight vector, $\boldsymbol{w}$, that minimizes the mean squared error (MSE) between the desired and actual responses of the perceptron. The function should be differentiable and have a unique minimum. The function of choice for this purpose is a quadratic of the form

$$E(\boldsymbol{w}) = \frac{1}{2}\left(r - \boldsymbol{w}^T\mathbf{x}\right)^2 \tag{13-47}$$

The 1/2 is used to cancel out the 2 that will result from taking the derivative of this expression. Also, remember that $\boldsymbol{w}^T\mathbf{x}$ is a scalar.

where E is our error measure, $\boldsymbol{w}$ is the weight vector we are seeking, $\mathbf{x}$ is any pattern from the training set, and r is the response we desire for that pattern. Both $\boldsymbol{w}$ and $\mathbf{x}$ are augmented vectors.



a b c

FIGURE 13.25 Plots of E as a function of wx for $r = 1$. (a) A value of α that is too small can slow down convergence. (b) If α is too large, large oscillations or divergence may occur. (c) Shape of the error function in 2-D.

We find the minimum of $E(\boldsymbol{w})$ using an iterative gradient descent algorithm, whose form is

Note that the right side of this equation is the gradient of $E(\boldsymbol{w})$.

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \alpha\left[\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}\right]_{\boldsymbol{w}=\boldsymbol{w}(k)} \tag{13-48}$$

where the starting weight vector is arbitrary, and $\alpha > 0$.

Figure 13.25(a) shows a plot of E for scalar values, w and x, of $\boldsymbol{w}$ and $\mathbf{x}$. We want to move w incrementally so $E(w)$ approaches a minimum, which implies that E should stop changing or, equivalently, that $\partial E(w)/\partial w = 0$. Equation (13-48) does precisely this. If $\partial E(w)/\partial w > 0$, a portion of this quantity (determined by the value of the learning increment α) is subtracted from $w(k)$ to create a new, updated value $w(k+1)$, of the weight. The opposite happens if $\partial E(w)/\partial w < 0$. If $\partial E(w)/\partial w = 0$, the weight is unchanged, meaning that we have arrived at a minimum, which is the solution we are seeking. The value of α determines the relative magnitude of the correction in weight value. If α is too small, the step changes will be correspondingly small and the weight would move slowly toward convergence, as Fig. 13.25(a) illustrates. On the other hand, choosing α too large could cause large oscillations on either side of the minimum, or even become unstable, as Fig. 13.25(b) illustrates. There is no general rule for choosing α. However, a logical approach is to start small and experiment by increasing α to determine its influence on a particular set of training patterns. Figure 13.25(c) shows the shape of the error function for two variables.

Because the error function is given analytically and it is differentiable, we can express Eq. (13-48) in a form that does not require computing the gradient explicitly at every step. The partial of $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ is

$$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\left(r - \boldsymbol{w}^T\mathbf{x}\right)\mathbf{x} \tag{13-49}$$

a b

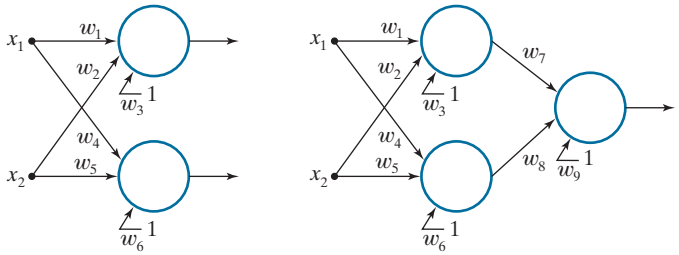


FIGURE 13.28
(a) Minimum perceptron solution to the XOR problem in 2-D. (b) A solution that implements the XOR truth table in Fig. 13.27(a).

Natural questions at this point are: Can more than one perceptron solve the XOR problem? If so, what is the minimum number of units required? We know that a single perceptron can implement one straight line, and we need to implement two lines, so the obvious answers are: yes to the first question, and two units to the second. Figure 13.28(a) shows the solution for two variables, which requires a total of six coefficients because we need two lines. The solution coefficients are such that, for either of the two patterns from class c_1, one output is true (1) and the other is false (0). The opposite condition must hold for either pattern from class c_2. This solution requires that we analyze two outputs. If we want to implement the truth table, meaning that a single output should give the same response as the XOR function [the third column in Fig. 13.27(a)], then we need one additional perceptron. Figure 13.28(b) shows the architecture for this solution. Here, one perceptron in the first layer maps any input from one class into a 1, and the other perceptron maps a pattern from the other class into a 0. This reduces the four possible inputs into two outputs, which is a two-point problem. As you know from Fig. 13.24, a single perceptron can solve this problem. Therefore, we need three perceptrons to implement the XOR table, as in Fig. 13.28(b).

With a little work, we could determine by inspection the coefficients needed to implement either solution in Fig. 13.28. However, rather than dwell on that, we focus attention in the following section on a more general, layered architecture, of which the XOR solution is a trivial, special case.

MULTILAYER FEEDFORWARD NEURAL NETWORKS

In this section, we discuss the architecture and operation of multilayer neural networks, and derive the equations of backpropagation used to train them. We then give several examples illustrating the capabilities of neural nets

Model of an Artificial Neuron

Neural networks are interconnected perceptron-like computing elements called *artificial neurons*. These neurons perform the same computations as the perceptron, but they differ from the latter in how they process the result of the computations. As illustrated in Fig. 13.23, the perceptron uses a "hard" thresholding function that outputs two values, such as +1 and −1, to perform classification. Suppose that in a network of perceptrons, the output before thresholding of one of the perceptrons is infinitesimally greater than zero. When thresholded, this very small signal will be turned into a +1. But a similarly small signal with the opposite sign would cause

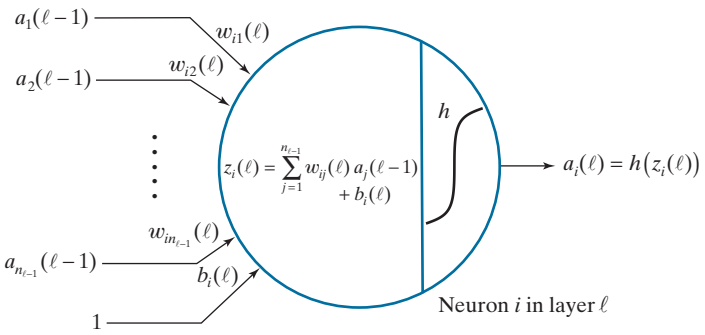


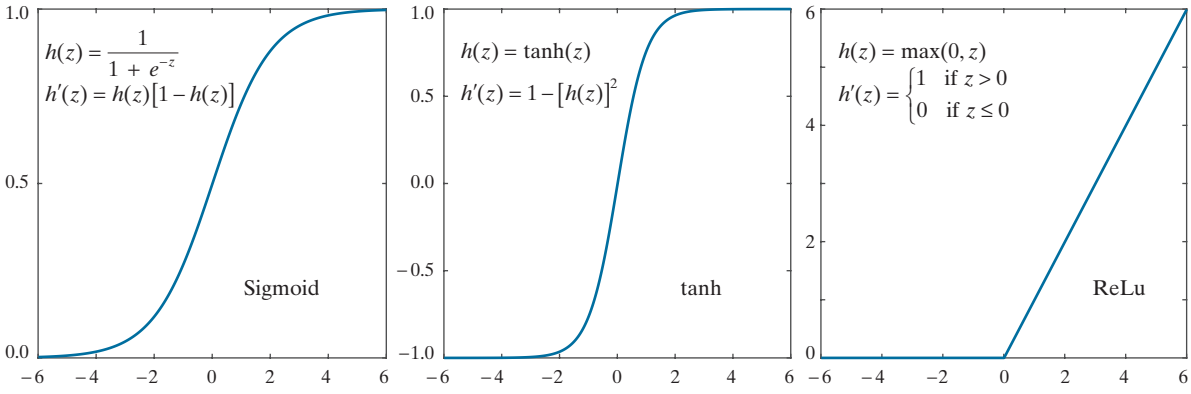
FIGURE 13.29 Model of an artificial neuron, showing all the operations it performs. The "ℓ" is used to denote a particular layer in a layered network.

a large swing in value from +1 to –1. Neural networks are formed from layers of computing units, in which the output of one unit affects the behavior of all units following it. The perceptron's sensitivity to the sign of small signals can cause serious stability problems in an interconnected system of such units, making perceptrons unsuitable for layered architectures.

The solution is to change the characteristic of the activation function from a hard-limiter to a smooth function. Figure 13.29 shows an example based on using the activation function

$$h(z) = \frac{1}{1 + e^{-z}} \tag{13-51}$$

where z is the result of the computation performed by the neuron, as shown in Fig. 13.29. Except for more complicated notation, and the use of a smooth function rather than a hard threshold, this model performs the same *sum-of-products* operations as in Eq. (13-36) for the perceptron. Note that the *bias* term is denoted by b instead



a b c

FIGURE 13.30 Various activation functions. (a) Sigmoid. (b) Hyperbolic tangent (also has a sigmoid shape, but it is centered about 0 in both dimensions). (c) Rectifier linear unit (ReLU).

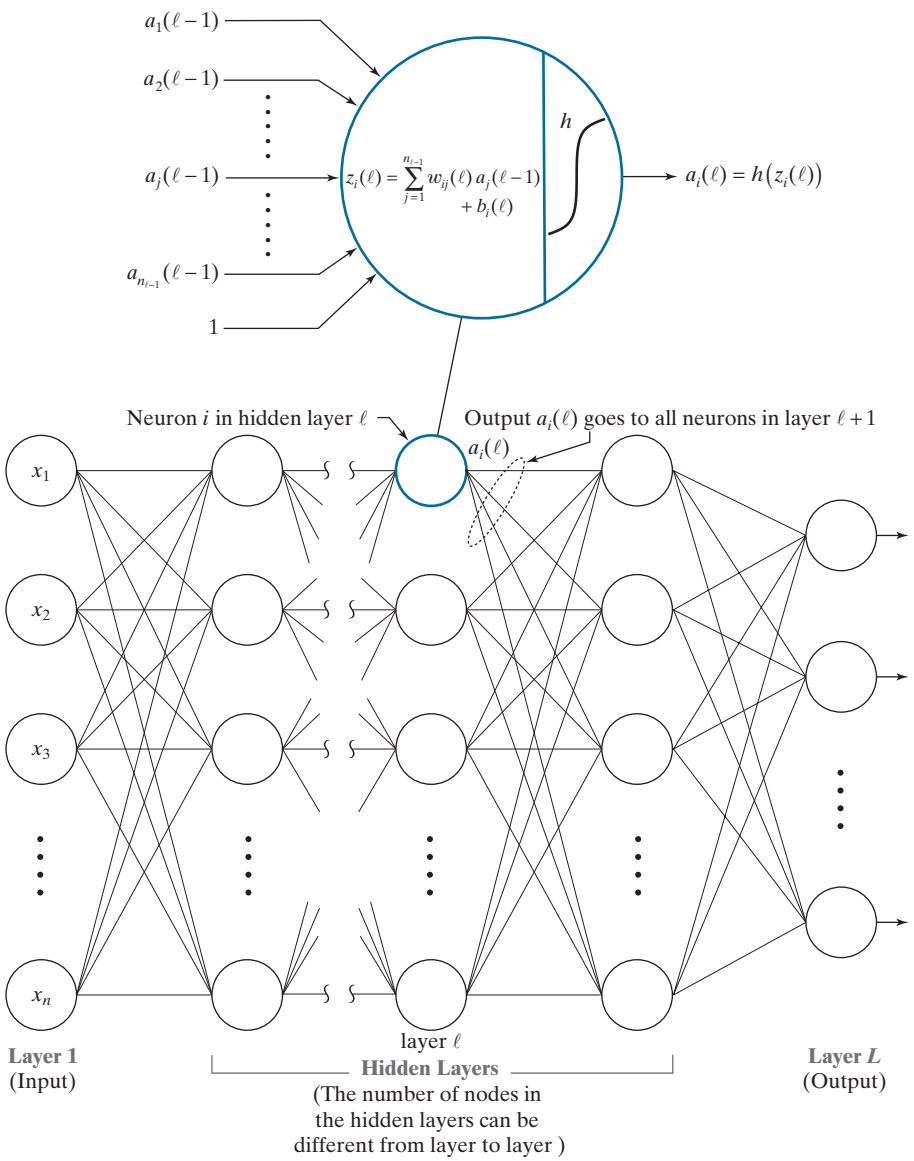


FIGURE 13.31 General model of a feedforward, fully connected neural net. The neuron is the same as in Fig. 13.29. Note how the output of each neuron goes to the input of all neurons in the following layer, hence the name *fully connected* for this type of architecture.

sometimes you will see the words "shallow" and "deep" used subjectively to denote networks with a "few" and with "many" layers, respectively.

We used the notation in Eq. (13-37) to label all the inputs and weights of a perceptron. In a neural network, the notation is more complicated because we have to account for neuron weights, inputs, and outputs within a layer, and also from layer to layer. Ignoring layer notation for a moment, we denote by w_{ij} the weight that associates the link connecting the *output* of neuron j to the input of neuron i. That is,

TABLE 13.2
Steps in the matrix computation of a forward pass through a fully connected, feedforward multilayer neural net.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Feedforward	For $\ell = 2, \ldots, L$, compute $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell)$ and $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$
Step 3	Output	$\mathbf{A}(L) = h(\mathbf{Z}(L))$

size $n_{\ell-1} \times n_p$, $\mathbf{B}(\ell)$ is of size $n_\ell \times n_p$, and $\mathbf{A}(\ell)$ is of size $n_\ell \times n_p$. Table 13.2 summarizes the matrix formulation for the forward pass through a fully connected, feedforward neural network for all pattern vectors. Implementing these operations in a matrix-oriented language like MATLAB is a trivial undertaking. Performance can be improved significantly by using dedicated hardware, such as one or more graphics processing units (GPUs).

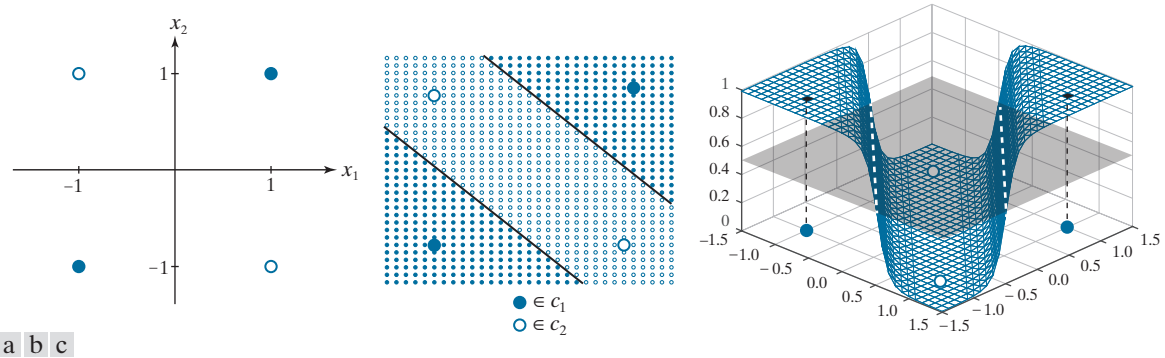
The equations in Table 13.2 are used to classify each of a set of patterns into one of n_L pattern classes. Each column of output matrix $\mathbf{A}(L)$ contains the activation values of the n_L output neurons for a specific pattern vector. The class membership of that pattern is given by the location of the output neuron with the highest activation value. Of course, this assumes we know the weights and biases of the network. These are obtained during training using backpropagation, as we explain next.

USING BACKPROPAGATION TO TRAIN DEEP NEURAL NETWORKS

A neural network is defined completely by its weights, biases, and activation function. Training a neural network refers to using one or more sets of training patterns to estimate these parameters. During training, we know the desired response of every output neuron of a multilayer neural net. However, we have no way of knowing what the values of the outputs of hidden neurons should be. In this section, we develop the equations of *backpropagation*, the tool of choice for finding the value of the weights and biases in a multilayer network. This *training by backpropagation* involves four basic steps: (1) inputting the pattern vectors; (2) a forward pass through the network to classify all the patterns of the training set and determine the classification error; (3) a backward (backpropagation) pass that feeds the output error back through the network to compute the changes required to update the parameters; and (4) updating the weights and biases in the network. These steps are repeated until the error reaches an acceptable level. We will provide a summary of all principal results derived in this section at the end of the discussion (see Table 13.3). As you will see shortly, the principal mathematical tool needed to derive the equations of backpropagation is the chain rule from basic calculus.

The Equations of Backpropagation

Given a set of training patterns and a multilayer feedforward neural network architecture, the approach in the following discussion is to find the network parameters



a b c

FIGURE 13.34 Neural net solution to the XOR problem. (a) Four patterns in an XOR arrangement. (b) Results of classifying additional points in the range −1.5 to 1.5 in increments of 0.1. All solid points were classified as belonging to class c_1 and all open circles were classified as belonging to class c_2. Together, the two lines separating the regions constitute the decision boundary [compare with Fig. 13.27(b)]. (c) Decision surface, shown as a mesh. The decision boundary is the pair of dashed, white lines in the intersection of the surface and a plane perpendicular to the vertical axis, intersecting that axis at 0.5. (Figure (c) is shown in a different perspective than (b) in order to make all four patterns visible.)

$$\mathbf{W}(2) = \begin{bmatrix} 4.792 & 4.792 \\ 4.486 & 4.486 \end{bmatrix};\ \mathbf{b}(2) = \begin{bmatrix} 4.590 \\ -4.486 \end{bmatrix};\ \mathbf{W}(3) = \begin{bmatrix} -9.180 & 9.429 \\ 9.178 & -9.427 \end{bmatrix};\ \mathbf{b}(3) = \begin{bmatrix} 4.420 \\ -4.419 \end{bmatrix}$$

Figure 13.35 shows the neural net based on these values.

When presented with the four training patterns after training was completed, the results at the two outputs should have been equal to the values in **R**. Instead, the values were close:

$$\mathbf{A}(3) = \begin{bmatrix} 0.987 & 0.990 & 0.010 & 0.010 \\ 0.013 & 0.010 & 0.990 & 0.990 \end{bmatrix}$$

These weights and biases, along with the sigmoid activation function, completely specify our trained neural network. To test its performance with values other than the training patterns, which we know it classifies correctly, we created a set of 2-D test patterns by subdividing the pattern space into increments of 0.1, from −1.5 to 1.5 in both directions, and classified the resulting points using a forward pass through

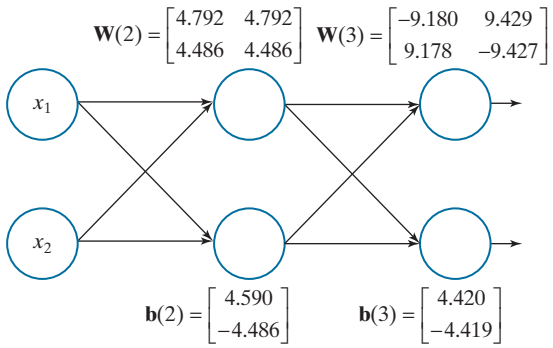
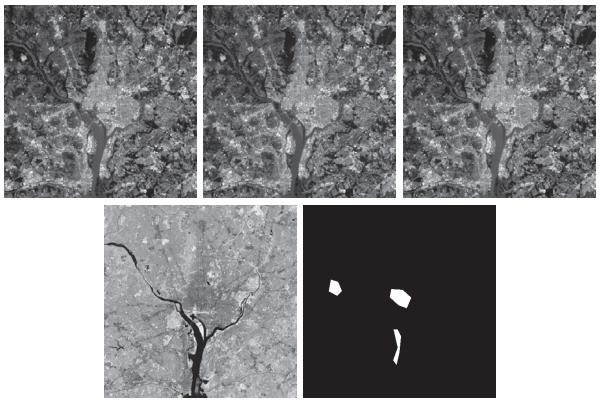
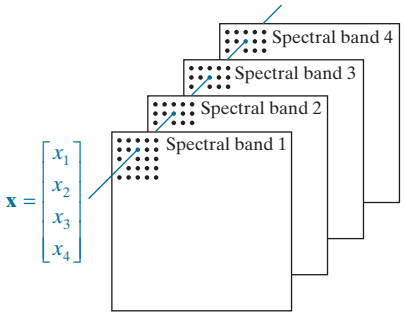


FIGURE 13.35 Neural net used to solve the XOR problem, showing the weights and biases learned via training using the equations in Table 13.3.



(a) Images in spectral bands 1–4 and binary mask used to extract training samples



(b) Approach used to extract pattern vectors

FIGURE 13.37 (a) Starting with the leftmost image: blue, green, red, near infrared, and binary mask images. In the mask, the lower region is for water, the center region is for the urban area, and the left mask corresponds to vegetation. All images are of size 512×512 pixels. (b) Approach used for generating 4-D pattern vectors from a stack of the four multispectral images. (Multispectral images courtesy of NASA.)

and *vegetation*. Figure 13.37 shows the four multispectral images used in the experiment, the masks used to extract the training and test samples, and the approach used to generate the 4-D pattern vectors.

As in Example 13.6, we extracted a total of 1900 training pattern vectors and 1887 test pattern vectors (see Table 13.1 for a listing of vectors by class). After preliminary runs with the training data to establish that the mean squared error was decreasing as a function of epoch, we determined that a neural net with one hidden layer of two nodes achieved stable learning with $\alpha = 0.001$ and 1,000 training epochs. Keeping those two parameters fixed, we varied the number of nodes in the internal layer, as listed in Table 13.4. The objective of these preliminary runs was to determine the smallest neural net that would give the best recognition rate. As you can see from the results in the table, [4 3 3] is clearly the architecture of choice in this case. Figure 13.38 shows this neural net, along with the parameters learned during training.

After the basic architecture was defined, we kept the learning rate constant at $\alpha = 0.001$ and varied the number of epochs to determine the best recognition rate with the architecture in Fig. 13.38. Table 13.5 shows the results. As you can see, the recognition rate improved slowly as a function of epoch, reaching a plateau at around 50,000 epochs. In fact, as Fig. 13.39 shows, the MSE decreased quickly up to about 800 training epochs and decreased slowly after that, explaining why the correct recognition rate changed so little after about 2,000 epochs. Similar results were obtained with $\alpha = 0.01$, but decreasing

TABLE 13.4
Recognition rate as a function of neural net architecture for $\alpha = 0.001$ and 1,000 training epochs. The network architecture is defined by the numbers in brackets. The first and last number inside each bracket refer to the number of input and output nodes, respectively. The inner entries give the number of nodes in each hidden layer.

Network Architecture	**[4 2 3]**	**[4 3 3]**	**[4 4 3]**	**[4 5 3]**	**[4 2 2 3]**	**[4 4 3 3]**	**[4 4 4 3]**	**[4 10 3 3]**	**[4 10 10 3]**
Recognition Rate	95.8%	96.2%	95.9%	96.1%	74.6%	90.8%	87.1%	84.9%	89.7%

this parameter to $\alpha = 0.1$ resulted in a drop of the best correct recognition rate to 49.1%. Based on the preceding results, we used $\alpha = 0.001$ and 50,000 epochs to train the network.

The parameters in Fig. 13.38 were the result of training. The recognition rate for the training data using these parameters was 97%. We achieved a recognition rate of 95.6% on the test set using the same parameters. The difference between these two figures, and the 96.4% and 96.2%, respectively, obtained for the same data with the Bayes classifier (see Example 13.6), are statistically insignificant.

The fact that our neural networks achieved results comparable to those obtained with the Bayes classifier is not surprising. It can be shown (Duda, Hart, and Stork [2001]) that a three-layer neural net, trained by backpropagation using a sum of errors squared criterion, approximates the Bayes decision functions in the limit, as the number of training samples approaches infinity. Although our training sets were small, the data were well behaved enough to yield results that are close to what theory predicts.

13.6 DEEP CONVOLUTIONAL NEURAL NETWORKS

Up to this point, we have organized pattern features as vectors. Generally, this assumes that the form of those features has been specified (i.e., "engineered" by a human designer) and extracted from images prior to being input to a neural network (Example 13.13 is an illustration of this approach). But one of the strengths of neural networks is that they are capable of learning pattern features directly from training data. What we would like to do is input a set of training images directly into a neural network, and have the network learn the necessary features *on its own*. One way to do this would be to convert images to vectors directly by organizing the pixels based on a linear index (see Fig. 13.1), and then letting each element (pixel) of the linear index be an element of the vector. However, this approach does not utilize any spatial relationships that may exist between pixels in an image, such as pixel arrangements into corners, the presence of edge segments, and other features that may help to differentiate one image from another. In this section, we present a class of neural networks called *deep convolutional neural networks* (*CNNs* or *ConvNets* for short) that accept images as inputs and are ideally suited for automatic learning and image classification. In order to differentiate between CNNs and the neural nets we studied in Section 13.5, we will refer to the latter as "fully connected" neural networks.

A BASIC CNN ARCHITECTURE

In the following discussion, we use a *LeNet* architecture (see references at the end of this chapter) to introduce convolutional nets. We do this for two main reasons: First, the LeNet architecture is reasonably simple to understand. This makes it ideal for introducing basic CNN concepts. Second, our real interest is in deriving the equations of backpropagation for convolutional networks, a task that is simplified by the intuitiveness of LeNets.

To simplify the explanation of the CNN in Fig. 13.40, we focus attention initially on a single image input. Multiple input images are a trivial extension we will consider later in our discussion.

The CNN in Fig. 13.40 contains all the basic elements of a LeNet architecture, and we use it without loss of generality. A key difference between this architecture and the neural net architectures we studied in the previous section is that inputs to CNNs are 2-D arrays (images), while inputs to our fully connected neural networks are vectors. However, as you will see shortly, the computations performed by both networks are very similar: (1) a sum of products is formed, (2) a bias value is added,

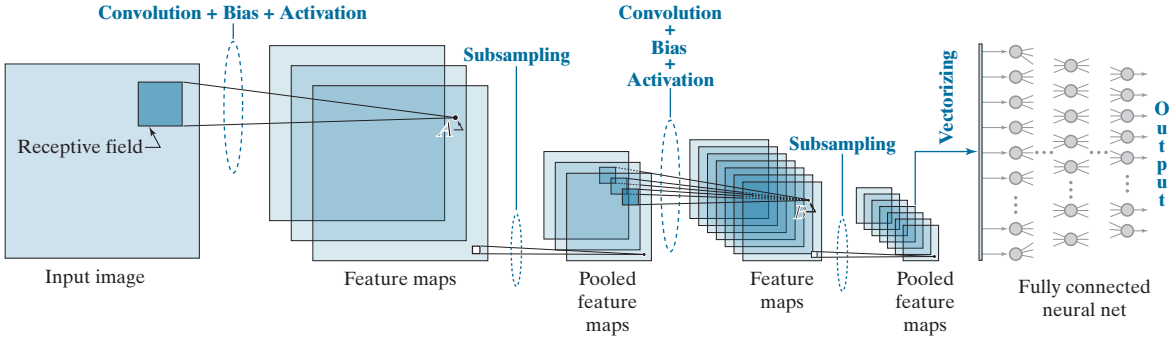


FIGURE 13.40 A CNN containing all the basic elements of a LeNet architecture. Points A and B are specific values to be addressed later in this section. The last pooled feature maps are vectorized and serve as the input to a fully connected neural network. The class to which the input image belongs is determined by the output neuron with the highest value.

(3) the result is passed through an activation function, and (4) the activation value becomes a single input to a following layer.

Despite the fact that the computations performed by CNNs and fully connected neural nets are similar, there are some basic differences between the two, beyond their input formats being 2-D versus vectors. An important difference is that CNNs are capable of learning 2-D features directly from raw image data, as mentioned earlier. Because the tools for systematically engineering comprehensive feature sets for complex image recognition tasks do not exist, having a system that can learn its own image features from raw image data is a crucial advantage of CNNs. Another major difference is in the way in which layers are connected. In a fully connected neural net, we feed the output of *every* neuron in a layer directly into the input of *every* neuron in the next layer. By contrast, in a CNN we feed into every input of a layer, a *single* value, determined by the convolution (hence the name *convolutional* neural net) over a spatial neighborhood in the output of the previous layer. Therefore, CNNs are not fully connected in the sense defined in the last section. Another difference is that the 2-D arrays from one layer to the next are subsampled to reduce sensitivity to translational variations in the input. These differences and their meaning will become clear as we look at various CNN configurations in the following discussion.

Basics of How a CNN Operates

We will discuss in the next subsection the exact form of neural computations in a CNN, and show they are equivalent in form to the computations performed by neurons in a fully connected neural net.

As noted above, the type of neighborhood processing in CNNs is spatial convolution. We explained the mechanics of spatial convolution in Fig. 3.36, and expressed it mathematically in Eq. (3-44). As that equation shows, convolution computes a sum of products between pixels and a set of kernel weights. This operation is carried out at every spatial location in the input image. The result at each location (x, y) in the input is a scalar value. Think of this value as the output of a neuron in a layer of a fully connected neural net. If we add a bias and pass the result through an activation function (see Fig. 13.29), we have a complete analogy between the

and

$$a_{x,y}(\ell) = h\left(z_{x,y}(\ell)\right) \tag{13-92}$$

for $\ell = 1, 2, \ldots, L_c$, where L_c is the number of convolutional layers, and $a_{x,y}(\ell)$ denotes the values of pooled features in convolutional layer ℓ. When $\ell = 1$,

$$a_{x,y}(0) = \{\text{values of pixels in the input image(s)}\} \tag{13-93}$$

When $\ell = L_c$,

$$a_{x,y}(L_c) = \{\text{values of pooled features in last layer of the CNN}\} \tag{13-94}$$

Note that ℓ starts at 1 instead of 2, as we did in Section 13.5. The reason is that we are naming layers, as in "convolutional layer ℓ." It would be confusing to start at convolutional layer 2. Finally, we note that the pooling does not require any convolutions. The only function of pooling is to reduce the spatial dimensions of the feature map preceding it, so we do not include explicit pooling equations here.

Equations (13-91) through (13-94) are all we need to compute all values in a forward pass through the convolutional section of a CNN. As described in Fig. 13.40, the values of the pooled features of the last layer are vectorized and fed into a fully connected feedforward neural network, whose forward propagation is explained in Eqs. (13-54) and (13-55) or, in matrix form, in Table 13.2.

THE EQUATIONS OF BACKPROPAGATION USED TO TRAIN CNNs

As you saw in the previous section, the feedforward equations of a CNN are similar to those of a fully connected neural net, but with multiplication replaced by convolution, and notation that reflects the fact that CNNs are not fully connected in the sense defined in Section 13.5. As you will see in this section, the equations of backpropagation also are similar in many respects to those in fully connected neural nets.

As in the derivation of backpropagation in Section 13.5, we start with the definition of how the output error of our CNN changes with respect to each neuron in the network. The form of the error is the same as for fully connected neural nets, but now it is a function of x and y instead of j:

$$\delta_{x,y}(\ell) = \frac{\partial E}{\partial z_{x,y}(\ell)} \tag{13-95}$$

As in Section 13.5, we want to relate this quantity to $\delta_{xy}(\ell + 1)$, which we again do using the chain rule:

$$\delta_{x,y}(\ell) = \frac{\partial E}{\partial z_{x,y}(\ell)} = \sum_u \sum_v \frac{\partial E}{\partial z_{u,v}(\ell+1)} \frac{\partial z_{u,v}(\ell+1)}{\partial z_{x,y}(\ell)} \tag{13-96}$$

FIGURE 13.48 Samples similar to those available in the NIST and MNIST databases. Each character subimage is of size 28×28 pixels. (Individual images courtesy of NIST.)

significant variability in the characters—and this is just a small sampling of the 70,000 characters available for experimentation.

Figure 13.49 shows the architecture of the CNN we trained to recognize the ten digits in the MNIST database. We trained the system for 200 epochs using $\alpha = 1.0$. Figure 13.50 shows the training MSE as a function of epoch for the 60,000 training images in the MNIST database.

Training was done using mini batches of 50 images at a time to improve the learning rate (see the discussion in Section 13.7). We also classified all images of the training set and all images of the test set after each epoch of training. The objective of doing this was to see how quickly the system was learning the characteristics of the data. Figure 13.51 shows the results. A high level of correct recognition performance was achieved after relatively few epochs for both data sets, with approximately 98% correct recognition achieved after about 40 epochs. This is consistent with the training MSE in Fig. 13.50, which dropped quickly, then began a slow descent after about 40 epochs. Another 160 epochs of training were required for the system to achieve recognition of about 99.9%. These are impressive results for such a small CNN.

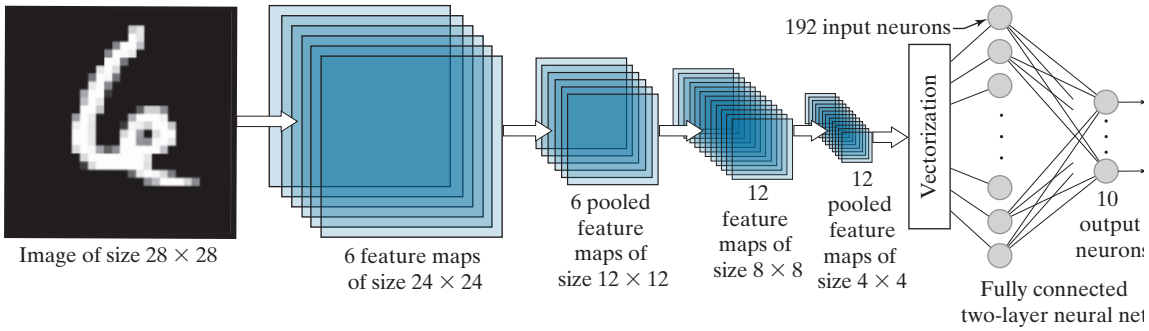


FIGURE 13.49 CNN used to recognize the ten digits in the MNIST database. The system was trained with 60,000 numerical character images of the same size as the image shown on the left. This architecture is the same as the architecture we used in Fig. 13.42. (Image courtesy of NIST.)

FIGURE 13.53 Kernels of the first layer after 200 epochs of training, shown as images.

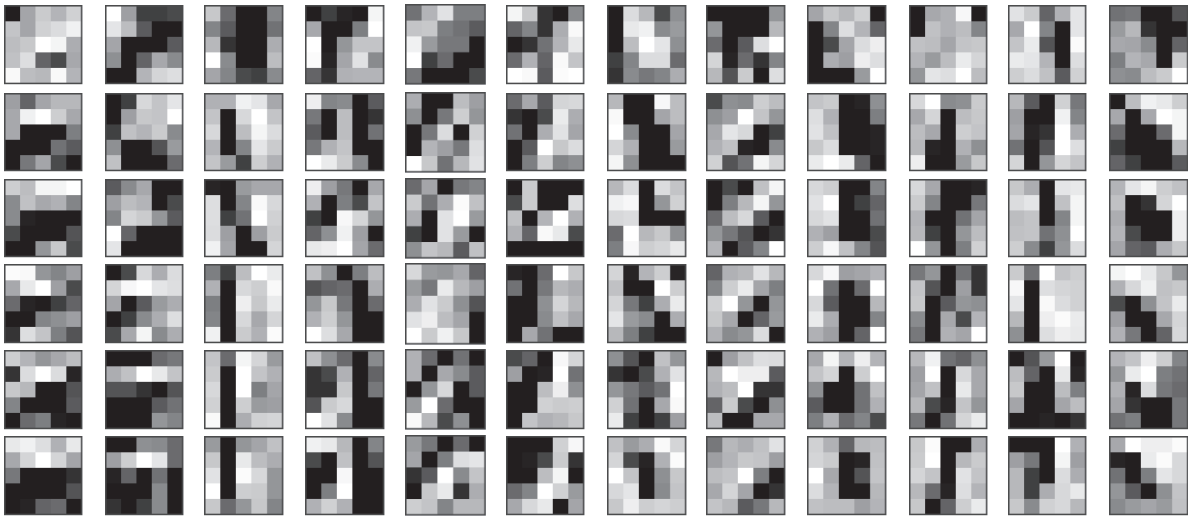


FIGURE 13.54 Kernels of the second layer after 200 epochs of training, displayed as images of size 5×5. There are six inputs (pooled feature maps) into the second layer. Because there are twelve feature maps in the second layer, the CNN learned the weights of $6 \times 12 = 72$ kernels.

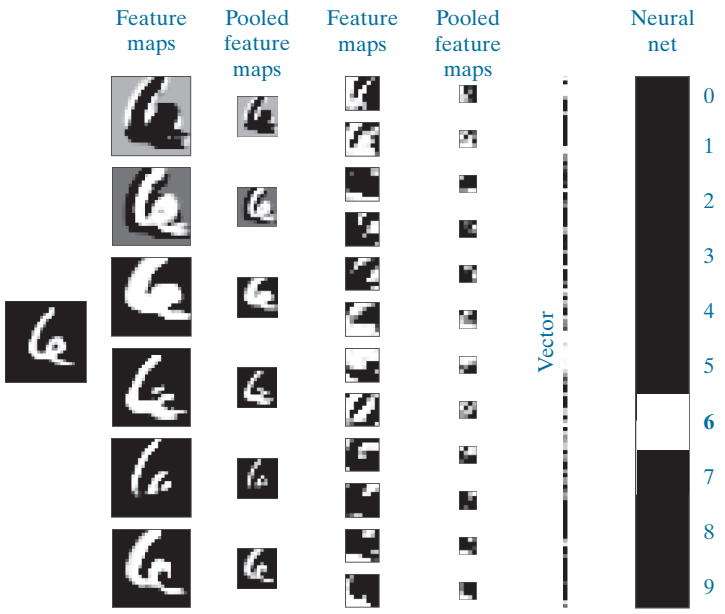


FIGURE 13.55 Results of a forward pass for one digit image through the CNN in Fig. 13.49 after training. The feature maps were generated using the kernels from Figs. 13.53 and 13.54, followed by pooling. The neural net is the two-layer neural network from Fig. 13.49. The output high value (in white) indicates that the CNN recognized the input properly. (This figure is the same as Fig. 13.44.)

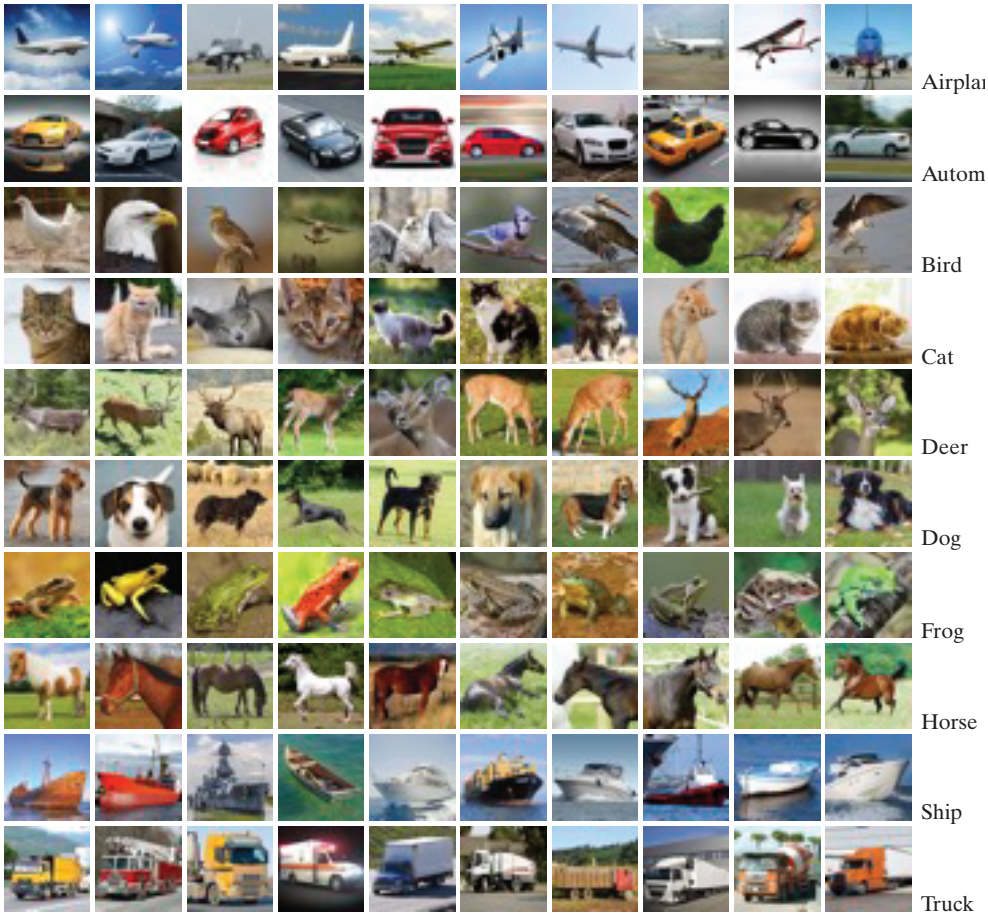


FIGURE 13.56 Mini images of size 32×32 pixels, representative of the 50,000 training and 10,000 test images in the CIFAR-10 database (the 10 stands for ten classes). The class names are shown on the right. (Images courtesy of Pearson Education.)

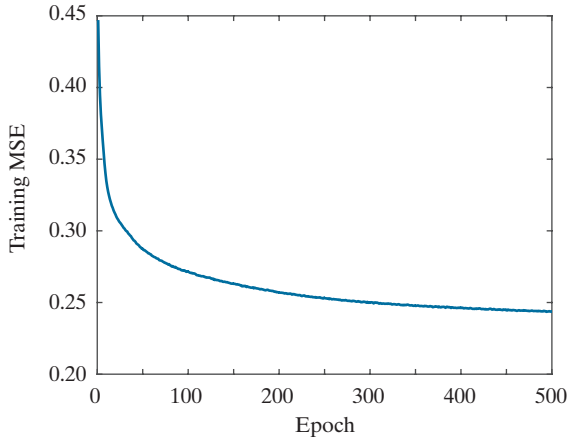


FIGURE 13.57 Training mean squared error as a function of the number of epochs for a training set of 50,000 CIFAR-10 images.

in the second layer is 6, and that the size of the pooling neighborhoods is again 2×2. What are the dimensions of the vectors that result from vectorizing the last layer of the CNN? Assume that vectorization is done using linear indexing.

13.31 Suppose the input images to a CNN are padded to compensate for the size reduction caused by convolution and subsampling (pooling). Let P denote the thickness of the padding border, let V denote the width of the (square) input images, let S denote the stride, and let F denote the width of the (square) receptive field.

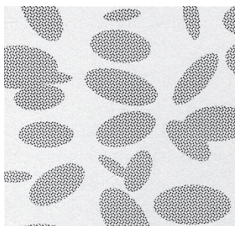
(a) Show that the number, N, of neurons in each row in the resulting feature map is

$$N = \frac{V + 2P - F}{S} + 1$$

(b)* How would you interpret a result using this equation that is not an integer?

13.32* Show the validity of Eq. (13-106).

13.33 An experiment produces binary images of blobs that are nearly elliptical in shape, as the following example image shows. The blobs are of three sizes, with the average values of the principal axes of the ellipses being (1.3, 0.7), (1.0, 0.5), and (0.75, 0.25). The dimensions of these axes vary ±10% about their average values.



Develop an image processing system capable of rejecting incomplete or overlapping ellipses, then classifying the remaining single ellipses into one of the three given size classes. Show your solution in block diagram form, giving specific details regarding the operation of each block. Solve the classification problem using a minimum distance classifier, indicating clearly how you would go about obtaining training samples, and how you would use these samples to train the classifier.

13.34 A factory mass-produces small American flags for sporting events. The quality assurance team has observed that, during periods of peak production, some printing machines have a tendency to drop (randomly) between one and three stars and one or two entire stripes. Aside from these errors, the flags are perfect in every other way. Although the flags containing errors represent a small percentage of total production, the plant manager decides to solve the problem. After much investigation, she concludes that automatic inspection using image processing techniques is the most economical approach. The basic specifications are as follows: The flags are approximately 7.5 cm by 12.5 cm in size. They move lengthwise down the production line (individually, but with a ±15% variation in orientation) at approximately 50 cm/s, with a separation between flags of approximately 5 cm. In all cases, "approximately" means ±5%. The plant manager employs you to design an image processing system for each production line. You are told that cost and simplicity are important parameters in determining the viability of your approach. Design a complete system based on the model of Fig. 1.23. Document your solution (including assumptions and specifications) in a brief (but clear) written report addressed to the plant manager. You can use any of the methods discussed in the book.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

13.1 Minimum-distance classifier.

(a) Write a function **minDistClass4e** that implements the minimum-distance classifier discussed in Section 13.3. Your function should have two modes of operation: **'train'**, in which the function computes the mean (prototype) vector of each class using a set of training patterns, and **'classify'**, in which the function