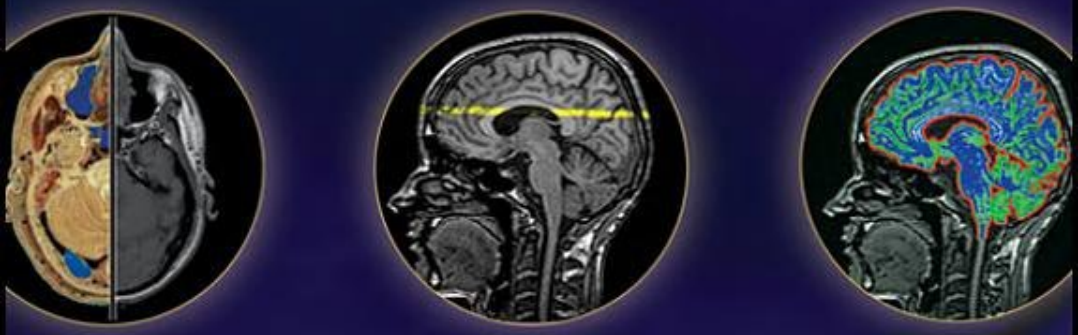
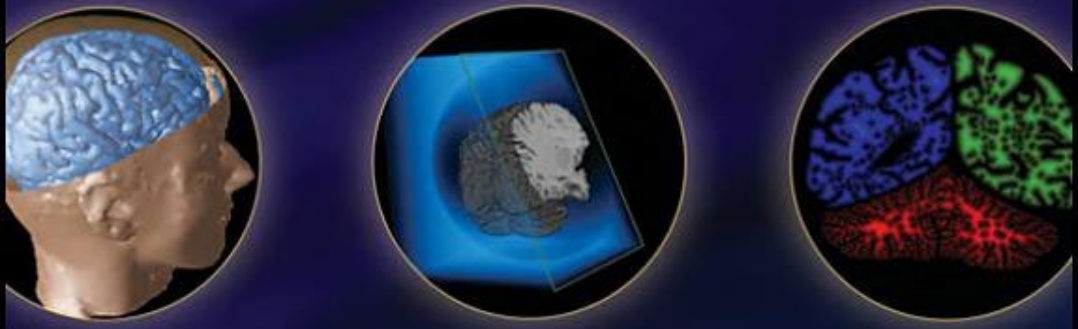


ADVANCED BIOMEDICAL IMAGE ANALYSIS



MARK A. HAIDEKKER

DVD-ROM INCLUDED

WILEY

ADVANCED BIOMEDICAL IMAGE ANALYSIS

ADVANCED BIOMEDICAL IMAGE ANALYSIS

MARK A. HAIDEKKER

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Haidekker, Mark A.

Advanced biomedical image analysis / Mark A. Haidekker

Includes bibliographical references and index.

ISBN 978-0-470-62458-6

Printed in Singapore

10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE

Medical imaging is one of the great revolutions in medicine. Traditionally, explorative surgery had to be performed to look inside a patient's body, even to perform a diagnosis. Slightly more than a century ago, x-rays were discovered. With it came the ability to look inside the body without surgery. X-ray imaging was rapidly adopted in medical centers worldwide. A new medical subject area was created—radiology. For decades, the radiologist had a basic set of tools, the x-ray tube, a fluoroscope, a film cassette with an image intensifier screen, and a light box. For decades, progress was incremental. X-ray tubes were improved, film and intensifier were made more sensitive, radiation exposure was reduced, and contrast agents were introduced and improved. It took a second, independent revolution to propel biomedical imaging to today's level: the invention of the programmable computer and its subsequent miniaturization. The availability of powerful digital data processing hardware and newly developed image processing methods paved the way for new imaging modalities: computed tomography, magnetic resonance imaging, ultrasound imaging, and functional imaging. These new imaging modalities had in common that computer-based data processing was required for image formation. Medical imaging experienced a second wave of rapid progress near the end of the twentieth century when tomography methods were developed and improved. *Tomography* means imaging by sections, and the origin of the word lies in the Greek τόμοσ for "to cut" and γράφω for "to write." With the availability of imaging modalities that produced three-dimensional reconstructions of a patient's body came the need for computerized image processing and computerized image visualization. The expertise of the radiologist in interpreting an image played–and still plays—a major role, but more and more tasks could be given to the computer, and the interpretation of computer-processed images became easier, more objective, and more accurate. Concurrently, scientists became interested in

programming computers with the "intelligence" to interpret and understand images. At the same time that computed tomography and magnetic resonance imaging were being invented, new computer methods for image analysis were being introduced.

The long-term vision is *computer-aided radiology*. Generalized to nonmedical fields, we could call it *computer-aided image interpretation*. After the first major wave of innovation in the 1970s to 1980s, computer algorithms for image interpretation have become more sophisticated and more complex. New mathematical methods emerged, such as, for example, the wavelet transform and set characterization by the fractal dimension, and were rapidly translated into advanced image analysis methods. Numerical methods that model physical processes—for example, differential equations for diffusion or motion—were applied to image processing tasks as diverse as noise reduction and segmentation. Artificial intelligence models are being used for computer analysis of high-dimensional feature vectors with the purpose of classifying the underlying pixels. Yet the philosopher's stone of image processing has not been discovered: to make a computer interpret an image with the same flexibility and immunity against artifacts as those of a human observer.

Although this book is well suited as a textbook for graduate-level image processing classes in the computer sciences and engineering fields, it is intended primarily as a reference book. The individual chapters are widely independent of the other chapters, with the exception of Chapters 2 and 3, which provide the foundation of basic image processing operations. The book is a comprehensive hands-on reference of topical image processing methods. *Hands-on* in this context indicates that not only are the theory, mathematical foundation, and basic description of an image processing operator provided, but performance features, advantages, and limitations are also discussed. Furthermore, key algorithms are provided in pseudocode to assist in implementation. The book aims at making advanced image processing operators accessible to those readers that have a basic familiarity with image processing. As such, the book can be seen as a toolbox in which each tool comes with a complete instruction manual. It is useful for readers who *use* the tools, because it helps them understand how the tools work. It is also useful for readers who *make* the tools, as it helps them design and optimize tools for their specific needs. And it is intended as a stepping-stone for those members of the imaging community who are reaching out to develop the next generation of tools.

The application focus of this book is biomedical. However, the same image processing and analysis principles also apply in many other fields. In the environmental and geological sciences, oceanography, soil sciences, forensic sciences, and anthropology, image analysis plays an important role. Satellite imagery and astrophotography, for example, pose the same image processing challenges as those of a magnetic resonance scan: the need to reduce noise, to emphasize the details of interest, and to make the important objects in the image accessible for subjective evaluation or objective measurement.

Acknowledgments

A number of people helped in a significant way to make this book a reality. First and foremost, I want to give my special thanks to my acquisition editor, Wayne Yuhasz,

for insisting on this book, for his confidence, and for his continual support throughout the writing process. I would also like to thank George J. Telecki, Lucy Hitz, and Angioline Loredo from John Wiley & Sons for their support during manuscript preparation and production. I particularly appreciated the help of my research specialist, Darcy Lichlyter, who spent many hours and after-hours in the library searching for literature no matter how well hidden it was. Adnan Mustafic, one of my graduate students, showed his competence, initiative, and skill by preparing the live DVD that accompanies this book. Professor Paul J. Friedman kindly provided me with a number of image samples. I also want to express my most sincere thanks to Professor Geoff Dougherty for reviewing the book manuscript and for providing numerous helpful and excellent suggestions for improving the book, as well as to Professor Michael Covington for many additional valuable comments. Last, but not least, I would like to thank my wife, Silke, who not only encouraged me to write the book and who endured some 80 weekends of my quasi-absence, but who also applied her expert skill as an accredited editor in the life sciences to proofread and copyedit the entire manuscript.

MARK A. HAIDEKKER

CUSTOMER NOTE: IF THIS BOOK IS ACCOMPANIED BY SOFTWARE, PLEASE READ THE FOLLOWING BEFORE OPENING THE PACKAGE.

This software contains files to help you utilize the models described in the accompanying book. By opening the package, you are agreeing to be bound by the following agreement:

1

IMAGE ANALYSIS: A PERSPECTIVE

The history of biomedical imaging is comparatively short. In 1895, Wilhelm Conrad Röntgen discovered a new type of radiation, which he called the x-ray. The discovery caused a revolution in medicine, because for the first time it became possible to see inside the human body without surgery. Use of x-rays in medical centers spread rapidly, but despite their vast popularity, little progress was made for over half a century. Soon after the discovery of x-rays, materials were discovered that exhibited visible-light fluorescence when illuminated by x-rays. With such materials, the quantum efficiency of film-based x-ray imaging could be improved and the exposure of patients to radiation thus reduced. Contrast agents were introduced around 1906 to allow imaging of some soft tissues (namely, intestines), which show low x-ray contrast. For about six decades, x-ray tubes, film, and x-ray intensifying materials were improved incrementally, but no fundamental innovation was made.

After World War II, the next important development in biomedical imaging finally arrived—ultrasound imaging. The medical technology was derived from military technology: namely, sonar (sound navigation and ranging), which makes use of sound propagation in water. Applying the same principles to patients, sound echos made visible on oscilloscope-like cathode ray screens allowed views into a patient's body without the use of ionizing radiation. The mere simplicity of creating sound waves and amplifying reflected sound made it possible to generate images with analog electronics—in the early stages with vacuum tubes. Electronic x-ray image intensifiers were a concurrent development. X-ray image intensifiers are electronic devices that are based on a conversion layer that emits electrons upon x-ray exposure.

Advanced Biomedical Image Analysis, By Mark A. Haidekker

These electrons are collected and amplified, then directed on a luminescent phosphor. Here, the image is formed with visible light and can be picked up by a video camera. Electronic intensifiers made it possible to further reduce patient exposure to x-rays and speed up the imaging process to a point where real-time imaging became possible. At this time, video cameras could be used to record x-ray images and display them instantly on video screens. Interventional radiology and image-guided surgery became possible.

The next major steps in biomedical imaging required an independent development: the evolution of digital electronics and the microprocessor. Milestones were the invention of the transistor (1948),[1] the integrated circuit as a prerequisite for miniaturization (1959), and the first single-chip microprocessor (1971).[20] Related to these inventions was the first integrated-circuit random-access memory (RAM; 1970).[62] Although the microprocessor itself was built on the principle of the programmable computer devised by Conrad Zuse in 1936, the miniaturization was instrumental in accumulating both computing power and memory in a reasonable space. Early digital computers used core memory, which got its name from small ferrite rings (cores) that could store 1 bit of information because of their magnetic remanence. Core memory was already a considerable achievement, with densities of up to 100 bits/cm^2. Early RAM chips held 10 times the memory capacity on the same chip surface area. In addition, integrated-circuit RAM did away with one disadvantage of core memory: the fact that a core memory read operation destroyed the information in the ferrite rings. Consequently, read and write operations with integrated-circuit RAM were many times faster. For four decades, integration density, and with it both memory storage density and processing power, has grown exponentially, a phenomenon known as *Moore's law*. Today's memory chips easily hold 1 trillion bits per square centimeter.*

The evolution of digital electronic circuits and computers had a direct impact on computer imaging. Image processing is memory-intensive and requires a high degree of computational effort. With the growing availability of computers, methods were developed to process images digitally. Many fundamental operators[15,18,24,32,36,43,64,72] were developed in the 1960s and 1970s. Most of these algorithms are in common use today, although memory restrictions at that time prevented widespread use. A medical image of moderate resolution (e.g., 256×256 pixels) posed a serious challenge for a mainframe computer with 4096 words of core memory, but today's central processing units (CPUs) would effortlessly fit the same image in their built-in fast cache memory without even having to access the computer's main memory. A convolution of the 256×256-pixel image with a 3×3 kernel requires almost 600,000 multiplications and the same number of additions. Computers in the 1970s were capable of executing on the order of 100,000 to 500,000 instructions per second (multiplication usually requires multiple instructions), and the convolution above would have cost several

*Today's most dense memory chips pose one challenge reminiscent of the information loss in core memory: Since the information is stored as a charge in a capacitor, which tends to leak and discharge slowly, the information needs to be read and rewritten several thousand times per second, a process known as *memory refresh.*

seconds of CPU time. On today's computers, the same convolution operation would be completed within a few milliseconds.

The availability of early mainframe computers and minicomputers for data processing enabled new revolutionary imaging modalities. In 1917, mathematician J. Radon stipulated that a manifold can be represented (transformed) by an infinite number of line integrals.[60] Almost 50 years later, when mainframe computers became widely accessible, A. M. Cormack developed an algorithm based on Radon's idea,[13,14] which in turn helped G. Hounsfield develop the computed tomography (CT) scanner.[37] Cormack and Hounsfield shared a Nobel prize in 1979 for development of the CT scanner. In fact, CT was a completely new type of imaging modality because it *requires* computed data processing for image formation: The x-ray projections collected during a CT scan need to be reconstructed to yield a cross-sectional image, and the reconstruction step takes place with the help of a computer.[42] Other imaging modalities, such as single-photon emission computed tomography (SPECT) and magnetic resonance imaging (MRI) also require the assistance of a computer for image formation.

Another important development in biomedical imaging resulted from the use of radioactively labeled markers. One such example is indium pentetreotide, a compound that acts as an analog for somatostatin and tends to accumulate in neuroendocrine tumors of the brain.[69] Indium pentetreotide can be labeled with radioactive ^{111}In, a gamma emitter. Another example is fluorodeoxyglucose, a glucose analog. Fluorodeoxyglucose accumulates at sites of high metabolic activity. When fluorodeoxyglucose is labeled with ^{18}F, it becomes a positron emitter. Radiation emission becomes stronger near active sites where the radiolabeled markers accumulate, and with suitable devices, tomographic images of the concentration of the radioactive compounds can be gathered. The use of positron emitters that create gamma rays as a consequence of electron–positron annihilation events was proposed in 1951[78] and eventually led to positron emission tomography (PET).[6] With radiolabeled physiologically active compounds (radiopharmaceuticals), it became possible to obtain images of physiological processes. These imaging methods not only improved the diagnosis of carcinomas, but also helped in our understanding of physiological processes, most notably brain activity. Functional imaging has become a key tool in medical diagnosis and research.

Subsequent research and development aimed at the improvement of image quality (e.g., improvement of resolution, better contrast, less noise). Current trends also include an increase in three-dimensional images and the involvement of computers in image processing and image analysis. A detailed overview of current trends is given in section 1.3.

1.1. MAIN BIOMEDICAL IMAGING MODALITIES

A number of fundamentally different methods of obtaining images from tissue, called *imaging modalities*, emerged during the historical development of biomedical imaging, and the information that these modalities provide differs among modalities.

It is outside our scope here to provide a detailed description of the physical and engineering foundations of the modalities, but a short overview is provided for completeness.

X-ray Imaging X-ray imaging is a projection method. The patient is illuminated by x-rays, high-energy photons that penetrate the body. Some of the x-rays are absorbed in the tissue. X-rays predominantly follow a straight path. The absorption process can be described by the *Lambert–Beer law*:

$$\ln I(x,y) = \ln I_0 - \int_s \mu(\vec{r})\, d\vec{r} \tag{1.1}$$

where I is the x-ray intensity that passes through a patient's body, I_0 the incident x-ray intensity, $\mu(\vec{r})$ the x-ray absorption coefficient at any spatial location $\vec{r}$, and the integration takes place along a straight line s, which intersects with the x-ray film at (x,y). At this location, the film is blackened by the x-rays, and the more x-rays that pass through the body, the higher the optical density of the film. At the end of this process, the film contains a two-dimensional distribution of optical density that relates to the tissue distribution inside the patient. If path s passes through bone, for example, the optical density at the end of that path is lower than that of a neighboring path, s', that traverses only soft tissue. In the case of film-based x-ray imaging, the film needs to be digitized with a film scanner to obtain a digital image. Filmless x-ray imaging with a digital detector is becoming more common.

Computed Tomography Computed tomography (CT) is an x-ray-based imaging method used to obtain a two- or three-dimensional map of absorbers inside the imaged object. The principle behind CT is to collect many projections, following equation (1.1), at various angles θ relative to the imaged object. One projection consists of measured attenuation values along parallel beams that are displaced a distance t from the center of rotation. When the incident beam intensity is known, the line integral along s can be represented by the computed attenuation p at detector position t and angle θ. Let us assume that the Fourier transform of the absorption map $\mu(x,y)$ is $M(u,v) = \mathcal{F}\{\mu(x,y)\}$, where the symbol $\mathcal{F}$ denotes the Fourier transform and u and v are the axes of the frequency-domain coordinate system (a detailed explanation is provided in Chapter 3). It can be shown that the one-dimensional Fourier transform of the projection with respect to t, $\mathcal{F}\{p(t,\theta)\}$, is identical to a one-dimensional cross section of the Fourier transform of the absorber map $M(u,v)$ subtending an angle θ with the u-axis. This relationship is known as the *Fourier slice theorem*. In CT, the projections $p(t,\theta)$ are obtained during the scanning process, but the absorption map $\mu(x,y)$ is unknown. The purpose of the scanning process is therefore to obtain many projection scans $p(t,\theta)$, to perform a Fourier transform, and to enter them at the angle θ into a placeholder $M(u,v)$, thereby filling as many elements of $M(u,v)$ as possible. The cross-sectional slice $\mu(x,y)$ is then obtained by computing the inverse Fourier transform of $M(u,v)$. Other reconstruction methods also exist (a comprehensive overview of CT reconstruction techniques is presented by Kak and

Slaney[42]), as well as reconstruction algorithms for beams that are not parallel but fan- or cone-shaped. To obtain projections at different angles, a CT scanner contains an x-ray source and a detector array mounted on opposite sides of a large ring (the gantry). The patient is placed in the center of the ring and the source-detector system rotates around the patient, collecting projections. The patient can be moved in the axial direction on a patient tray. The patient tray not only allows patient positioning but also the acquisition of three-dimensional images.

Magnetic Resonance Imaging Magnetic resonance imaging (MRI) is another modality that requires the use of a computer for image formation. In a strong magnetic field, protons orient their spins along the magnetic field. The magnetic moments are not perfectly aligned, but rather, precess around the external field lines with an angular frequency that is proportional to the external field. The precession frequency is known as the *Larmor frequency*. With an externally introduced radio-frequency (RF) signal in resonance, that is, at the Larmor frequency, the orientation of the electron spins can be manipulated, but after cessation of the RF signal, the spins return to their original position. During this process, the spins emit a weak RF signal (echo) that can be picked up by an antenna. The time is takes for the spins to return to their original position depends on the tissue. Magnetic gradients allow us to change the precession frequency and precession phase angle along the spatial axes, and the spatial origin of a RF echo component can be reconstructed by Fourier analysis of the signal. In fact, the task of any MRI pulse sequence (i.e., the sequence of RF signals that manipulates spin precession) is to fill a frequency-domain placeholder, called a *k-space matrix*, with data. Inverse Fourier transform of the *k*-space matrix yields the cross-sectional image. Depending on the pulse sequence, different information can be obtained from the tissue. Three tissue constants are the relaxation times T_1 and T_2 and the proton density (water content). These tissue constants can vary strongly between different types of soft tissue, and for this reason, MRI provides excellent tissue–tissue contrast.

Ultrasound Imaging Ultrasound imaging makes use of the physics of sound propagation in tissue. Sound waves propagate at a certain, tissue-dependent velocity. At the interface between two tissues, some of the sound is reflected, and the sound echo can be picked up by a receiver. The round-trip time of the echo can be translated into the depth of the echo source because the speed of sound is known. An *A-mode scan* (the echo strength as a function of depth) is obtained by emitting a short burst of sound into the tissue and recording the echos for a short period of time. Sound generation and recording are carried out by transducers made of a piezoelectric material, that is, crystals that deform under the influence of an electric field and that generate an electrostatic field when deformed. An A-mode scan can be represented as a thin line on a screen where the intensity depends on the echo strength. By directing the incident sound wave in different directions, a *B-mode scan* can be obtained. A B-mode scan consists of several parallel or fan-shaped A-mode scans. It is also possible to record A-mode scans as a function of time, which is referred to as an *M-mode* (motion

mode). Although ultrasound imaging could be performed with purely analog circuits, today's ultrasound devices use digital signal and image processing. One disadvantage of ultrasound imaging is its widely qualitative nature, although size measurements are possible with moderate accuracy. However, ultrasound can quantitatively measure motion (notably, blood flow) through the Doppler effect.

Single-Photon Emission Computed Tomography (SPECT) SPECT is analogous to CT but uses a radiation source internal to a patient rather than the external x-ray generator used in CT. The radiation source is usually a radiopharmaceutical that emits gamma rays. External cameras obtain projections of the radiation strength analogous to the projection $p(t,\theta)$ in CT. Radiation is kept weak, and highly sensitive cameras need to be used to capture as many radioactive events as possible. Since the radioactive events are random processes, the image has a very high noise component. Therefore, SPECT image reconstruction methods are optimized for noisy images. Furthermore, the resolution is much lower than that of CT. However, SPECT allows us to image the accumulation of radioactive tracers at physiologically active sites. Whereas CT provides information about the structure of tissue, SPECT provides information about function and physiological activity.

Positron Emission Tomography (PET) Like SPECT, PET uses radiolabeled markers to image physiological activity. Whereas SPECT uses primarily gamma emitters, the radiolabeled compounds in PET are positron emitters (such as ^{18}F, ^{15}O, ^{124}I, and ^{89}Zr). The positron (i.e., antimatter electron) collides with an electron a short distance from its emission site. In this annihilation event, two high-energy gamma photons are emitted in opposite directions. These photon pairs are captured in a detector ring. The occurrence of photon pairs is critical for PET imaging, because the time-of-flight difference, together with the location of the registering detectors, allows us to determine the exact location of the annihilation event in the scanned slice. Moreover, spontaneous background events can be excluded; only coincident gamma pairs are recorded. A sufficient number of recorded events create an image of spatially resolved physiological activity. Like SPECT images, PET images have low spatial resolution and low signal-to-noise ratio.

Visible-Light Imaging Visible-light imaging with conventional digital cameras or specialized scanning elements also play an important role in biomedical imaging. Often, visible-light imaging is found in conjunction with microscopy. A large number of techniques provide information about the imaged object, such as a cell layer or tissue sample. Tissue can be stained (histology) or labeled fluorescently. Fluorescent markers can provide physiological information on the microscopic level in a complementary fashion to SPECT and PET, which provide physiological information on a macroscopic level. Confocal microscopy allows us to obtain three-dimensional volumetric images.

1.2. BIOMEDICAL IMAGE ANALYSIS

Biomedical image analysis is a highly interdisciplinary field, being at the interface of computer sciences, physics, medicine, biology, and engineering. Fundamentally, biomedical image analysis is the application of image processing techniques to biological or medical problems. However, in biomedical image analysis, a number of other fields play an important role:

- *Anatomy*. Knowledge of shape, structure, and proximity to other anatomical objects can help identify features in images and determine abnormalities.
- *Physiology*. Physiology plays a role in functional imaging, where functional imaging should be defined very broadly, ranging from blood flow imaged with Doppler ultrasound to cell physiology imaged with microscopy and fluorescent probes. The term *functional imaging* is often used for modern methods to image physiological processes by means of functional MRI, PET, or SPECT.
- *Physics of the imaging modality*. Depending on the imaging modality, the image values represent fundamentally different properties of the imaged object. Examples include x-ray attenuation in CT, optical light scattering and absorption in microscopy and optical coherence tomography (OCT), and magnetic spin relaxation time constants in MRI.
- *Instrumentation*. Even within the same modality, images of the same object can be markedly different. One example is x-ray imaging and CT, where the anode voltage and the introduction of beam-hardening filters influence the apparent x-ray density, whereas beam collimation determines the amount of haze and blur in the image. Another example is the exposure time and source brightness in optical modalities (microscopy, OCT), which may produce contrast in different regions of the object. In addition, every imaging instrument introduces some amount of noise.
- *Medical application*. The medical application in diagnosis or intervention provides the foundation and motivation for biomedical image analysis. The selection of an imaging modality and of possible image processing steps depends on many medical factors, such as the suspected disease or the type of tissue to be imaged.

In addition, several fields of computer science exist on top of image processing that play a role in biomedical image analysis, most notably artificial intelligence and computer modeling. Artificial intelligence approaches find their ways into biomedical image analysis in the form of fuzzy logic,[2,26,48] evolutionary computing,[9,73] computer learning,[44,59] and artificial neural networks.[28,79,82] Computer models play a key role in advanced segmentation techniques and in the description of time-course dynamics.[35,50,70]

Biomedical image analysis consists of four distinct stages, where each stage is generally a prerequisite for the next stage, but at any stage the chain can end to

allow a human observer to make a decision or record results. These stages are image acquisition, image enhancement and restoration, image segmentation, and image quantification.

1.2.1. Image Acquisition

The first stage is to gather information about the object, such as a suspect tissue in a patient. In the context of an image, the information is spatially resolved. This means that the image is a map of one or more tissue properties on the nodes of a discrete rectangular grid. The grid nodes coincide with integer coordinates, and the image value on an integer coordinate is termed a *pixel* (picture element) or *voxel* (volume element) in three-dimensional images. The image values are stored in finite memory; therefore, the image values themselves are also discrete. In many cases, image values are limited to integer values. Noninteger values can be used as image values if a floating-point representation for a pixel is used, but floating-point values have limited precision as well.

The image values themselves generally have physical meaning. To name a few examples, photography and microscopy provide image values that are proportional to light intensity. Computed tomography provides image values that are proportional to local x-ray absorption. In magnetic resonance imaging, the image values can represent a variety of tissue properties, depending on the acquisition sequence, such as local echo decay times or proton density.

The goal of the image acquisition stage is to obtain contrast. To use x-ray imaging as an example, let us assume a patient with a suspected clavicular hairline fracture. X-rays passing through the clavicle are strongly attenuated, and film optical density is low. X-rays along neighboring paths that lead though soft tissue are less attenuated, and the optical density at corresponding locations of the film is higher. Contrast between bone and surrounding soft tissue is generally high. Some x-rays will pass though the fracture, where there is less bone to pass through, and the corresponding areas of the x-ray film show a slightly higher optical density—they appear darker. In this example it is crucial that the contrast created by x-rays passing through the intact clavicle and through the fracture is high enough to become discernible. This task is made more difficult by the inhomogeneity of other tissue regions that are traversed by the x-rays and cause unwanted contrast. *Unwanted contrast* can be classified as noise in the broader sense (in a stricter sense, noise is a random deviation of a pixel value from an idealized value), and distinguishing between contrast related to the suspected disease—in this case the clavicular fracture—and contrast related to other contrast sources leads immediately to the notion of the signal-to-noise ratio. *Signal* refers to information (i.e., contrast) related to the feature of interest, whereas *noise* refers to information not related to the feature of interest.

The human eye is extremely good at identifying meaningful contrast, even in situations with poor signal-to-noise ratios. Human vision allows instant recognition of spatial relationships and makes it possible to notice subtle variations in density and to filter the feature from the noise. A trained radiologist will have no difficulty in identifying the subtle shadow caused by a hairline fracture, a microscopic region of

lower optical density in a mammogram, or a slight deviation from the normal shape of a ventricle, to name a few examples. However, these tasks may pose a considerable challenge for a computer. This is where the next steps of the image processing chain come into play.

1.2.2. Image Enhancement

Image enhancement can serve two purposes: to improve the visibility of features to allow a human observer (radiologist) to make a more accurate diagnosis or better extract information, or to prepare the image for the next processing steps. The most common image enhancement operators are:

- *Pixel value remapping*. This includes linear or nonlinear contrast enhancement, histogram stretching, and histogram equalization.
- *Filtering*. Filters amplify or attenuate specific characteristics of an image, and filters make use of the pixel neighborhood. Filters that operate on a limited pixel neighborhood (often based on the discrete convolution operation) are referred to as *spatial-domain filters*. Other filters use a specific transform, such as the Fourier transform, which describes the image information in terms of periodic components (frequency-domain filters). To name a few examples, filters can be used to sharpen edges or smooth an image, to suppress periodic artifacts, or to remove an inhomogeneous background intensity distribution.

A specific form of image enhancement is *image restoration*, a specific filtering technique where a degradation process is assumed to be known. Under this assumption, the image acquisition process is modeled as the acquisition of an idealized, undegraded image that cannot be accessed, followed by the degradation process. A restoration filter is a filter designed to reverse the degradation process in such a manner that some error metric (such as the mean-squared error) is minimized between the idealized unknown image and the restored image. The restored image is the degraded image subjected to the restoration filter. Since the idealized image is not accessible, the design of restoration filters often involves computer simulations or computer modeling.

Whether for enhancement or restoration, filter design is a critical step in image processing. Typically, the degradation process introduces two components: blur and noise. In some cases, such as microscopy, inhomogeneous illumination may also play a role. Illumination may change over time, or motion artifacts may be introduced. Unfortunately, filters often require balancing of design criteria. Filters to counteract blurring and filters for local contrast enhancement tend to amplify the noise component and therefore reduce the signal-to-noise ratio. Conversely, noise-reducing filters negatively affect edges and detail texture: Whereas these filters increase the signal-to-noise ratio, image details may get blurred and lost. Moreover, filter design goals depend on the next steps of the image processing chain. To enhance an image for a human observer, the noise component plays a less critical role, because the human eye can recognize details despite noise. On the other hand, automated processing

such as segmentation requires that the image contains as little noise as possible even at the expense of some edge and texture detail.

1.2.3. Image Segmentation

Image segmentation is the step where an object of interest in the image is separated from the background. Background in this definition may include other objects. To perform image segmentation successfully, the object of interest must be distinguishable from background in some way: for example, by a difference in image intensity, by a delineating boundary, or by a difference in texture. Sometimes, a priori knowledge such as a known shape can help in the segmentation process. The goal of this process can be either a *mask* or an outline. A mask is an image where one pixel value (usually, 1) corresponds to an object pixel, while another pixel value (usually, 0) corresponds to background. An *outline* can be a parametric curve or a set of curves, such as a polygonal approximation of an object's shape. There seems to be almost no limit to the complexity of segmentation approaches, and there is certainly a trend toward more complex segmentation methods being more application-specific. An overview of the most popular segmentation methods follows.

Intensity-Based Segmentation When the intensity of the object of interest differs sufficiently from the intensity of the background, an intensity threshold value can be found to separate the object from its background. Most often, an object is a more or less convex shape, and pixel connectivity rules can be used to improve the segmentation results. Intensity-based segmentation methods that make use of connectivity are region growing and hysteresis thresholding. Intensity thresholds can be either global (for the entire image) or locally adaptive. To some extent, intensity-based thresholding methods can be applied to images where the object texture (i.e., the local pixel intensity distribution) differs from background, because the image can be filtered to convert texture features into image intensity values. The latter is a good example of how image filtering can be used in preparation for image segmentation.

Edge-Based Segmentation Sometimes, objects are delineated by an intensity gradient rather than by a consistent intensity difference throughout the object. In such a case, a local contrast filter (edge detector) can be used to create an image where the outline of the object has a higher intensity than the background. Intensity-based thresholding then isolates the edge. Images containing the edge of the object are prerequisites for parametric segmentation methods such as boundary tracking and active contours. Parametric shapes (i.e., lines, circles, ellipses, or polygonal approximations of a shape) can also be extracted from the edge image using the Hough transform.

Region-Based Segmentation Some segmentation approaches make more extensive use of local similarity metrics. Regions may be similar with respect to intensity or texture. In fact, a feature vector can be extracted for each pixel that contains diverse elements, including intensity, local intensity variations, or directional variations. Similarity could be defined as the Euclidean distance between feature

vectors. Unsupervised region-based methods are region splitting and region merging or region growing. Region splitting starts with the entire image as one region and subdivides the image recursively into squares with dissimilar regions, whereas region growing starts at the pixel level and joins similar regions. The two methods can be combined to form split-merge segmentation, where splitting and merging alternate.

Clustering If feature pixels are already separated from the background, clustering is a method to group the feature pixels into clusters. Assignment of individual pixels to clusters can be based on the Euclidean distance to the cluster center or on other similarity metrics. Most widely used are the *k*-means clustering method, where each pixel is assigned to exactly one cluster and fuzzy *c*-means clustering, where cluster membership is continuous (fuzzy) and to some degree, each pixel belongs to multiple clusters.

Neural Networks In medical imaging, a widely used approach is to train artificial neural networks with feature vectors that have been manually assigned to classes. Once a neural network has been trained, it can then segment similar images in an unsupervised manner. Although unsupervised segmentation is desirable, some manual interaction can greatly facilitate the segmentation task. Examples of limited manual interaction are the placement of seed points for region growing, and crude object delineation for active contour models and live-wire techniques.

If the segmentation result is a binary mask, some postprocessing may improve the segmentation result. Examples of postprocessing steps are removal of isolated pixels or small clusters, morphological thinning and extraction of a single pixel-wide skeleton, morphological operations to reduce boundary irregularities, filling of interior holes or gaps, and the separation of clusters with weak connectivity.

1.2.4. Image Quantification

Similar to the way that a radiologist uses an image to assess the degree of a disease for a diagnosis, image quantification encompasses methods to classify objects or to measure the properties of an object. Image quantification requires that the object be segmented. The goal of image quantification is either to classify an object (e.g., as diseased or healthy) or to extract a continuous descriptor (e.g., tumor size or progression). The advantage of computerized image quantification is its objectivity and speed.

Examples of continuous variables include the measurement of intensity, density, size, or position. As an example, bone mineral density is a crucial determinant of bone strength, and people (especially women) lose bone mineral with age, a condition that may lead to osteoporosis. X-ray imaging techniques (quantitative CT and dual-energy x-ray absorptiometry) are particularly suited to measuring bone mineral density.[29] The degree of osteoporosis and with it the risk of a patient to suffer spontaneous fractures is often determined by comparing bone density to an age-matched distribution.[25] Today, this measurement is usually highly automated, with unsupervised segmentation of

the bone examined (e.g., vertebrae or the calcaneus) and subsequent determination of the bone mineral density, measured in milligrams of calcium hydroxyapatite per milliliter of bone, from the x-ray attenuation.

Examples of classification include healthy/diseased or healthy tissue/benign lesion/malignant lesion. A typical example is the classification of suspicious masses in mammograms as benign or malignant. This classification can be based on the shape of the segmented lesion[61] or on the texture.[67] To illuminate the classification process, let us look at texture analysis, a process that is representative of a large number of similar approaches to classifying lesions in x-ray mammograms. Sahiner et al. propose segmenting the lesion and extracting its outline, then transforming a narrow band of pixels perpendicular to the outline onto a rectangular region where descriptive metrics can be extracted from the texture.[67] In their study a total of 41 scalar values were extracted from each image using methods based on the co-occurrence matrix and run-length analysis. The 41 values formed a descriptive feature vector, and a classification scheme based on computer learning (Fischer's linear discriminant classifier[45]) was used by radiologists to produce a malignancy rating from 1 to 10.[10] The generation of high-dimensional feature vectors and the use of artificial intelligence methods to obtain a relatively simple decision from the feature vector is very widespread in image quantification.

1.3. CURRENT TRENDS IN BIOMEDICAL IMAGING

Once computers started to play an instrumental role in image formation in modalities such as CT and MRI, the next step was indeed a small one: to use the same computers for image enhancement. Operations such as contrast enhancement, sharpening, and noise reduction became integrated functions in the imaging software. A solid body of image processing operations has been developed over the last 30 years, and many of them provide the foundation for today's advanced image processing and analysis operations. A continuous long-term goal, however, remains: to use computers to aid a radiologist in diagnosing a disease. Much progress toward this goal has been made. As mentioned above, established computer-aided imaging methods to determine bone density and therefore indicate the degree of osteoporosis are in clinical use. CT scans can be used to find colon polyps and help diagnose colon cancer. Optical coherence tomography has rapidly been established in ophthalmology to diagnose retinal diseases. Yet there are many more areas where increasing computing power combined with more elaborate computational methods hold some promise of helping a radiologist with the diagnosis, but the trained observer proves to be superior to computerized image analysis. An example is computerized mammography, where a lot of progress has been made but no single method has entered mainstream medical practice. With the vision of computer-aided radiology, where computers provide an objective analysis of images and assume tedious parts of the image evaluation, advanced biomedical image analysis is—and will remain for a long time—an area of intense research activity.

Progress in image analysis is aided by a dramatic increase in the memory and processing power of today's computers. Personal computers with several gigabytes of memory are common, and hard disks have reached beyond the terabyte limit. Storing and processing a three-dimensional image of $512 \times 512 \times 512$ bytes is possible with almost any off-the-shelf personal computer. This processing power benefits not only computerized image analysis but also image acquisition. Volumetric imaging modalities generate images of unsurpassed resolution, contrast, and signal-to-noise ratio. Acquisition speed has also improved, giving rise to real-time imaging that allows motion measurements, for example, of the heart muscle.[54,89]

As the availability of tomographic scanners increases, multimodality imaging becomes more popular. Clinicians and researchers aim to obtain as much information on the tissue under examination as possible. Predominantly, one imaging modality that provides a high-resolution image of the tissue or organ (normally, CT or MRI) is combined with a functional imaging modality, such as PET.[51] Multimodality imaging is particularly popular in cancer diagnosis and treatment, because it is possible to place a radioactive label on tumor-specific antibodies. PET, and to a lesser extent, SPECT, are used to image antibody uptake by the tumor, whereas the exact localization of the tumor is found by combining the PET or SPECT image with MRI or CT (see pertinent reviews[33,41,53,55]). Multimodality imaging produces two or more images generally with different resolution and probably with different patient positioning between images. The different images need to be matched spatially, a process called *image registration*. Dual-modality imaging devices are available (e.g., a combined PET/CT scanner), but software registration is most often used, and new registration techniques are an active field of research.[71]

Another recent field of study is that of optical imaging techniques, more specifically tomographic imaging with visible or near-infrared light. A prerequisite for this development was the introduction of new light sources (specifically, lasers) and new mathematical models to describe photon propagation in diffusive tissues.[12,23] Unlike x-ray and CT imaging, visible light does not travel along a straight path in tissue because of the high scattering coefficient of tissue and because of tissue regions with different refractive index. Optical coherence tomography (OCT)[39] is often considered the optical equivalent of ultrasound imaging because the image is composed of A-mode scans. OCT has a low penetration depth of a few millimeters, but it provides good spatial resolution. Optical coherence tomography has found wide application in dermatology and ophthalmology (see reviews[22,38,63,68,84]), but its poor signal/noise ratio calls for advanced image enhancement methods. Optical transillumination tomography, the optical equivalent of CT, faces major challenges because of refractive index changes along the light rays. Progress has been made to use optical transillumination tomography to image bone and soft tissues,[74,87] but spatial resolution and contrast remain limited. Attempts have been made to correct the refractive index mismatch in software[30] and to reject scattered photons,[11,34] but major improvements are needed before this modality enters medical practice. The third major optical tomography method is diffuse optical tomography.[27] Its main challenge is the mathematical modeling of light-wave propagation, which is a

prerequisite for image reconstruction.[16] Presently, diffuse optical tomography requires crucial improvements in spatial resolution and signal-to-noise ratio before it becomes applicable in biomedical imaging. These challenges notwithstanding, optical imaging methods enjoy strong research efforts because they promise fast and radiation-free image acquisition with relatively inexpensive instrumentation.

A special focus of imaging and image analysis is the brain. Brain imaging studies have been driven in part by the availability of MRI, which does not expose study subjects to ionizing radiation, and in part by functional imaging techniques, which make it possible to localize areas of brain activity.[40,83] Another important subject is the development of anatomical brain atlases, which allow mapping of images with high interindividual variability onto known anatomical models.[3,56] Although our understanding of the brain is still rudimentary, biomedical imaging has helped enormously to find the loci of brain activity, to understand cognitive functions, and to link images to disease (see pertinent articles and reviews[5,7,46,52,66]).

On the general image processing side, new methods and operators of higher complexity also tend to be more application- and modality-specific. Three recent articles highlight the challenges: Masutani et al.[50] review image modalities and image processing methods specifically for the diagnosis and treatment of liver diseases; Hangartner[31] demonstrates how a key step in segmentation—threshold selection—affects the quantitative determination of density and geometry in CT images; and Sinha and Sinha[70] present MRI-based imaging techniques for breast lesions. All three examples have in common the fact that the methods and conclusions cannot readily be translated into other modalities or applications. The main reason for this very common phenomenon is the inability of computers to understand an image in the same way that a human observer does. A computer typically examines a limited pixel neighborhood and attempts to work its way up toward more global image features. Conversely, a human observer examines the entire scene and discovers features in the scene in a top-down approach. The problem of image understanding, allowing computers to recognize parts of an image similar to the way that a human observer does, has been approached with algorithms that involve learning[8] and, more recently, with a top-down analysis of statistical properties of the scene layout[57] and with imitation of the human visual system through genetic algorithms.[88] Image understanding is not limited to biomedical imaging but also affects related fields of computer vision and robotics and is therefore another area of intensive research.

Related to image understanding is the problem of image segmentation. Meaningful unsupervised image segmentation requires certain image understanding by the computer. The scope of most image segmentation algorithms is limited to special cases (e.g., where the object of interest differs in intensity from the background). The main reason is the extreme variability of medical images, which makes it difficult to provide a consistent definition of successful segmentation. Learning algorithms, artificial neural networks, and rule-based systems are examples of state-of-the art approaches to segmentation.[17,90] More recently, new methods to compare segmentation algorithms objectively have been proposed,[81] and a database with benchmark segmentation problems has been created.[49] These examples illuminate the present search for a more unified segmentation paradigm.

The development of new filters is another area of research. Early spatial- and frequency-domain filters used fixed filter parameters. Subsequently, filter parameters became dependent on the local properties of the image. These filters are called *adaptive filters*. Many recently developed filters are tuned toward specific modalities, with examples such as a noise filter for charge-coupled-device (CCD) cameras,[21] an adaptive filter to remove noise in color images,[47] a speckle reduction filter for optical coherence tomography,[58] or a fuzzy filter for the measurement of blood flow in phase-contrast MRI.[76] Novel filters are highly sought after because a good filter can often make possible an otherwise impossible segmentation and quantification task.

A final example for emerging areas in image processing is the use of image data for modeling. The use of finite-element models to predict bone strength[80] is a particularly good example because CT image values depend strongly on mineral content, and it is hypothesized that bone strength and mineral content are strongly related. However, those models need further improvement before they become useful in clinical practice.[4] Image-based computational fluid dynamics can be used to compute blood flow and wall shear stress in arteries that are frequently affected by arteriosclerosis.[86] One recent example is a study by Sui et al.[75] in which MR images of the carotid artery were used to calculate wall shear stress. Experiments with cell culture indicate that shear stress gradients enhance cell proliferation and therefore contribute to arteriosclerosis,[85] and image-based flow simulations are a suitable tool to further elucidate the disease and perhaps aid in the prediction and early diagnosis of arteriosclerosis.[77]

1.4. ABOUT THIS BOOK

The overall aim of this book is to provide the reader with a comprehensive reference and self-study guide that covers advanced techniques of quantitative image analysis with a focus on biomedical applications. The book addresses researchers, professionals, teachers, and students in all areas related to imaging. Ideally, the reader has some prior basic knowledge of image processing. Any reader who has an interest or involvement in imaging may use this book for a conceptual understanding of the subject, and to use equipment and software more efficiently. The reader will gain an overview of advanced image analysis techniques that were recently established or are still in active research, and the book illuminates the inner workings of image processing software, which often comes with such imaging devices as scientific cameras, microscopes, CT scanners, and optical coherence tomography devices. Furthermore, readers will gain the ability to understand and use the algorithms in a meaningful way and the ability to design their own algorithms based on understanding gained from this book. Readers with programming experience in C, C++, Python, Java, Matlab, or other languages can use this book to implement and refine custom algorithms for advanced image processing and unsupervised analysis. A survey of software programs for image analysis and image visualization is provided in Chapter 14, and the focus is placed on free software such as ImageJ and OpenDX, which the reader can freely download and put to immediate use.

Each chapter provides the mathematical background for an image processing operator with the purpose of explaining how it works. We then proceed to applications and limitations and to their realization in software. For key algorithms, a pseudocode implementation is provided. The pseudocode can be translated readily into many programming languages and is also well suited to explain the "inner workings" of an image processing operator. In addition, each chapter includes application examples in the biomedical field. Although the application focus is biomedical, the methods described are not restricted to the biomedical field. Some areas in which advanced image analysis plays a key role are satellite imaging, oceanography, environmental and geological sciences, soil sciences, anthropology, forensic sciences, and astronomy.

It was mentioned that a basic understanding of image processing is beneficial for readers of this book. Several books are available that provide such a basic understanding. Two notable examples are an image processing handbook by Russ[65] and a medical imaging book by Dougherty.[19] The main strength of the first book derives from the numerous examples from various imaging-related fields, and the main strength of the second book is its combined coverage of medical imaging modalities and medical image processing.

In the present book, the topics follow the outline given in Section 1.2. Chapter 2 provides an overview of established and fundamental image processing operators and can be used to review the main topics of basic image processing. In Chapter 3 we introduce the Fourier transform and image filtering in the frequency domain. The subject of Chapter 3 is fundamental and established, yet its extremely widespread use and importance warrants detailed coverage in a separate chapter. There is considerable overlap between Chapters 2 and 3 and the two book examples mentioned above.

In Chapter 4 we introduce the wavelet transform and explain filters that use the wavelet transform. Unlike the Fourier transform, the wavelet transform retains spatial information and gives rise to new and very powerful image filters. Since the introduction of the wavelet transform, wavelet-based filters have rapidly found their way into mainstream image processing. In Chapter 5 we explain the concepts and examples of adaptive filters. Spatial-domain filters were introduced in Chapter 2, but conventional filters have fixed parameters. Adaptive filters adjust their filter parameters to local image properties and can achieve a superior balance between noise removal and detail preservation.

Chapters 6 and 7 present two different approaches to segmentation: active contours and the Hough transform. Active contours (the two-dimensional versions are referred to as "snakes") are physical models of energy functionals that use image features as an external energy term. An active contour behaves like a rubber band that snaps onto prominent image features such as edges. Active contours are generally used as supervised segmentation methods that yield a parametric representation of a shape. The Hough transform (Chapter 7) is a popular method used to find parametric shapes in images, such as lines, circles, or ellipses. The strength of the Hough transform is that the shape does not need to be complete, and the transform can be used to find the shape even if the image is strongly corrupted by noise.

Chapters 8, 9, and 10 present advanced techniques for image quantification. In Chapter 8 we explain methods for texture analysis and texture quantification. Texture

refers to local gray-scale variations in an image, and texture information can be used for feature extraction, segmentation, and for quantitative analysis that is, for example, related to a diagnosis. Analogously, in Chapter 9 we explain methods used to describe and classify the shape of segmented objects. Texture and shape analysis are very powerful tools for extracting image information in an unsupervised manner. In many images, structures exhibit some apparent self-similarity; that is, a shape or texture repeats itself on smaller scales. Self-similarity is related to the fractal dimension, and fractal methods have been used widely to quantify medical images. In Chapter 10 we provide an introduction, overview, and in-depth analysis of fractal methods for shape and texture quantification.

Chapters 11 and 12 are more focused on special biomedical problems. In Chapter 11 we explain the principles of image registration, that is, methods used to match the exact resolution and spatial position of two images that were taken with different modalities or with the same modality when a patient shifted. As multimodality imaging grows even more popular, image registration gains in importance. Image compression, storage, and transportation are covered in Chapter 12. Medical imaging produces an exponentially increasing volume of data through a growing number of medical procedures and higher resolution. With archiving required and telemedicine evolving, approaches to handling increasing data volumes are introduced in Chapter 12.

Chapter 13 covers image visualization, an important step in preparing an image for analysis by a human observer. The image can be two- or three-dimensional, or it can be a time-course sequence. Depending on the application, brightness and contrast manipulation, false coloring, and three-dimensional rendering can emphasize important aspects of an image and facilitate the analysis task of a human observer.

Chapter 14 provides a link to the practical application of topics covered in the book. A number of software programs for image processing, analysis, and visualization are presented briefly. Two popular packages, ImageJ and OpenDX, are covered in more detail, as is Crystal Image, the author's software on the accompanying DVD. This software was used to create most of the examples in this book and to test the algorithms covered in the book. Most of the software in Chapter 14 is free and can be downloaded and put to use immediately.

With these elements, the book provides a solid and in-depth foundation toward understanding advanced image analysis techniques and developing new image analysis operators.

REFERENCES

1. Bardeen J, Brattain WH. The transistor, a semi-conductor triode. Phys Rev 1948; 74(2):230–231.
2. Bezdek JC, Hall LO, Clark MC, Goldgof DB, Clarke LP. Medical image analysis with fuzzy models. Stat Methods Med Res 1997; 6(3):191–214.
3. Bohm C, Greitz T, Thurfjell L. The role of anatomic information in quantifying functional neuroimaging data. J Neural Transm Suppl 1992; 37:67–78.

4. Bonnick SL. Noninvasive assessments of bone strength. Curr Opin Endocrinol Diabetes Obes 2007; 14(6):451–457.
5. Brown GG, Eyler LT. Methodological and conceptual issues in functional magnetic resonance imaging: applications to schizophrenia research. Annu Rev Clin Psychol 2006; 2:51–81.
6. Brownell GH, Burnham CA. MGH positron camera. In: Freedman GS, editor. Tomographic Imaging in Nuclear Medicine. New York: Society for Nuclear Medicine, 1972; 154–164.
7. Bullmore E, Sporns O. Complex brain networks: graph theoretical analysis of structural and functional systems. Nat Rev Neurosci 2009; 10(3):186–198.
8. Caelli T, Bischof WF. Machine learning paradigms for pattern recognition and image understanding. Spat Vis 1996; 10(1):87–103.
9. Cagnoni S, Bergenti F, Mordonini M, Adorni G. Evolving binary classifiers through parallel computation of multiple fitness cases. IEEE Trans Syst Man Cybern B 2005; 35(3):548–555.
10. Chan HP, Sahiner B, Helvie MA, Petrick N, Roubidoux MA, Wilson TE, Adler DD, Paramagul C, Newman JS, Sanjay-Gopal S. Improvement of radiologists' characterization of mammographic masses by using computer-aided diagnosis: an ROC study. Radiology 1999; 212(3):817–827.
11. Chen K, Perelman LT, Zhang Q, Dasari RR, Feld MS. Optical computed tomography in a turbid medium using early arriving photons. J Biomed Opt 2000; 5(2):144–154.
12. Colak SB, Papaioannou DG, 't Hooft GW, van der Mark MB, Schomberg H, Paasschens JCJ, Melissen JBM, van Asten NAAJ. Tomographic image reconstruction from optical projections in light-diffusing media. Appl Opt 1997; 36(1):180–213.
13. Cormack AM. Representation of a function by its line integrals, with some radiological applications I. J Appl Phys 1963; 34(9):2722–2727.
14. Cormack AM. Representation of a function by its line integrals, with some radiological applications II. J Appl Phys 1964; 35(10):2908–2913.
15. Danielson GC, Lanczos C. Some improvements in practical Fourier analysis and their applications to x-ray scattering from liquids. J Franklin Inst 1942; 233:365–380.
16. Dehghani H, Srinivasan S, Pogue BW, Gibson A. Numerical modelling and image reconstruction in diffuse optical tomography. Philos Trans A 2009; 367(1900):3073–3093.
17. Dhawan AP. Medical Image Analysis. Piscataway, NJ: IEEE Press, 2003.
18. Dijkstra EW. A note on two problems in connection with graphs. Numer Math 1959; 1:269–271.
19. Dougherty G. Digital Image Processing for Medical Applications. Cambridge, UK: Cambridge University Press, 2009.
20. Faggin F, Hoff ME, Mazor S, Shima M. The history of the 4004. IEEE Micro 1996; 16(6):10–20.
21. Faraji H, MacLean WJ. CCD noise removal in digital images. IEEE Trans Image Process 2006; 15(9):2676–2685.
22. Farkas DL, Becker D. Applications of spectral imaging: detection and analysis of human melanoma and its precursors. Pigment Cell Res 2001; 14(1):2–8.
23. Franceschini MA, Moesta KT, Fantini S, Gaida G, Gratton E, Jess H, Mantulin WW, Seeber M, Schlag PM, Kaschke M. Frequency-domain techniques enhance optical mammography: initial clinical results. Proc Natl Acad Sci U S A 1997; 94(12):6468–6473.

24. Galloway MM. Texture analysis using gray level run lengths. Comput Graph Image Process 1975; 4:172–179.
25. Genant HK, Ettinger B, Harris ST, Block JE, Steiger P. Radiology in osteoporosis. In: Riggs BL, Melton J, editors. Osteoporosis: Etiology, Diagnosis, and Management. New York: Raven, 1988:221–249.
26. Ghosh A, Shankar BU, Meher SK. A novel approach to neuro-fuzzy classification. Neural Netw 2009; 22(1):100–109.
27. Gibson A, Dehghani H. Diffuse optical imaging. Philos Trans A 2009; 367(1900): 3055–3072.
28. Gil J, Wu HS. Applications of image analysis to anatomic pathology: realities and promises. Cancer Invest 2003; 21(6):950–959.
29. Guglielmi G, Gluer CC, Majumdar S, Blunt BA, Genant HK. Current methods and advances in bone densitometry. Eur Radiol 1995; 5(2):129–139.
30. Haidekker MA. Optical transillumination tomography with tolerance against refraction mismatch. Comput Methods Programs Biomed 2005; 80(3):225–235.
31. Hangartner TN. Thresholding technique for accurate analysis of density and geometry in QCT, pQCT and microCT images. J Musculoskel Neuronal Interact 2007; 7(1):9–16.
32. Haralick RM, Shanmugam K, Dinstein I. Textural features for image classification. IEEE Trans Syst Man Cybern 1973; 3(6):610–621.
33. Hay R, Cao B, Tsarfaty I, Tsarfaty G, Resau J, Woude GV. Grappling with metastatic risk: bringing molecular imaging of Met expression toward clinical use. J Cell Biochem Suppl 2002; 39:184–193.
34. Hebden JC, Arridge SR, Delpy DT. Optical imaging in medicine: I. Experimental techniques. Phys Med Biol 1997; 42(5):825–840.
35. Hedrick TL. Software techniques for two- and three-dimensional kinematic measurements of biological and biomimetic systems. Bioinspir Biomim 2008; 3(3):34001.
36. Hough PVC. Method and means for recognizing complex patterns. US patent 3,069,654. 1962.
37. Hounsfield GN. Computerized transverse axial scanning (tomography): 1. Description of system. Br J Radiol 1973; 46(552):1016–1022.
38. Hrynchak P, Simpson T. Optical coherence tomography: an introduction to the technique and its use. Optom Vis Sci 2000; 77(7):347–356.
39. Huang D, Swanson EA, Lin CP, Schuman JS, Stinson WG, Chang W, Hee MR, Flotte T, Gregory K, Puliafito CA. Optical coherence tomography. Science 1991; 254(5035):1178–1181.
40. Jezzard P, Buxton RB. The clinical potential of functional magnetic resonance imaging. J Magn Reson Imaging 2006; 23(6):787–793.
41. Jones MJ, Koeneman KS. Local-regional prostate cancer. Urol Oncol 2008; 26(5):516–521.
42. Kak AC, Slaney M. Principles of Computerized Tomographic Imaging. New York: IEEE Press, 1988. Electronic edition, 1999.
43. Kirsch R. Computer determination of the constituent structure of biological images. Comput Biomed Res 1971; 4:315–328.
44. Klenk JA. Characteristics and value of machine learning for imaging in high content screening. Methods Mol Biol 2007; 356:83–94.

45. Lachenbruch PA. Discriminant Analysis. New York: Hafner, 1975.
46. Liu PK, Mandeville JB, Guangping D, Jenkins BG, Kim YR, Liu CH. Transcription MRI: a new view of the living brain. Neuroscientist 2008; 14(5):503–520.
47. Ma Z, Wu HR, Feng D. Partition-based vector filtering technique for suppression of noise in digital color images. IEEE Trans Image Process 2006; 15(8):2324–2342.
48. Marengo E, Robotti E, Antonucci F, Cecconi D, Campostrini N, Righetti PG. Numerical approaches for quantitative analysis of two-dimensional maps: a review of commercial software and home-made systems. Proteomics 2005; 5(3):654–666.
49. Martin D, Fowlkes C, Tal D, Malik J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Proc 8th Intl Conf Comput Vis 2001; 2: 416–423.
50. Masutani Y, Uozumi K, Akahane M, Ohtomo K. Liver CT image processing: a short introduction of the technical elements. Eur J Radiol 2006; 58(2):246–251.
51. Mawlawi O, Townsend DW. Multimodality imaging: an update on PET/CT technology. Eur J Nucl Med Mol Imaging 2009; 36 (Suppl 1):S15–S29.
52. May A, Matharu M. New insights into migraine: application of functional and structural imaging. Curr Opin Neurol 2007; 20(3):306–309.
53. Mironov S, Akin O, Pandit-Taskar N, Hann LE. Ovarian cancer. Radiol Clin North Am 2007; 45(1):149–166.
54. Moore CC, McVeigh ER, Zerhouni EA. Quantitative tagged magnetic resonance imaging of the normal human left ventricle. Top Magn Reson Imaging 2000; 11(6):359–371.
55. Niu G, Cai W, Chen X. Molecular imaging of human epidermal growth factor receptor 2 (HER-2) expression. Front Biosci 2008; 13:790–805.
56. Nowinski WL. The cerefy brain atlases: continuous enhancement of the electronic talairach-tournoux brain atlas. Neuroinformatics 2005; 3(4):293–300.
57. Oliva A, Torralba A. Building the gist of a scene: the role of global image features in recognition. Prog Brain Res 2006; 155:23–36.
58. Ozcan A, Bilenca A, Desjardins AE, Bouma BE, Tearney GJ. Speckle reduction in optical coherence tomography images using digital filtering. J Opt Soc Am A 2007; 24(7):1901–1910.
59. Pereira F, Mitchell T, Botvinick M. Machine learning classifiers and fMRI: a tutorial overview. Neuroimage 2009; 45 (1 Suppl):S199–S209.
60. Radon J. On the determination of functions through line integrals along certain manifolds. Berlin, Saxon Acad Sci 1917; 29:262–279.
61. Rangayyan RM, El-Faramawy NM, Desautels JEL, Alim OA. Measures of acutance and shape for classification of breast tumors. IEEE Trans Med Imaging 1997; 16(6):799–810.
62. Regitz W, Karp J. A three transistor-cell, 1024-bit, 500 NS MOS RAM. Proc IEEE Conf ISSCC 1970; 8: 42–43.
63. Ripandelli G, Coppe AM, Capaldo A, Stirpe M. Optical coherence tomography. Semin Ophthalmol 1998; 13(4):199–202.
64. Roberts LG. Machine perception of three-dimensional solids. In: Tippet JT, editor. Optical and Electro-optical Information Processing. Cambridge, MA: MIT Press, 1965: 159–197.
65. Russ JC. The Image Processing Handbook, 5th ed. Boca Raton, FL: Taylor & Francis, 2006.

66. Rykhlevskaia E, Gratton G, Fabiani M. Combining structural and functional neuroimaging data for studying brain connectivity: a review. Psychophysiology 2008; 45(2): 173–187.
67. Sahiner B, Chan HP, Petrick N, Helvie MA, Goodsitt MM. Computerized characterization of masses on mammograms: the rubber band straightening transform and texture analysis. Med Phys 1998; 25(4):516–526.
68. Schaudig U. Optische Kohärenztomographie [Optical coherence tomography]. Ophthalmologe 2001; 98(1):26–34.
69. Seregni E, Chiti A, Bombardieri E. Radionuclide imaging of neuroendocrine tumours: biological basis and diagnostic results. Eur J Nucl Med 1998; 25(6):639–658.
70. Sinha S, Sinha U. Recent advances in breast MRI and MRS. NMR Biomed 2009; 22(1):3–16.
71. Slomka PJ, Baum RP. Multimodality image registration with software: state-of-the-art. Eur J Nucl Med Mol Imaging 2009; 36 (Suppl 1):S44–S55.
72. Sobel I. Camera Models and Machine Perception. AIM-21. Palo Alto, CA: Stanford Artificial Intelligence Laboratory, 1970.
73. Song A, Ciesielski V. Texture segmentation by genetic programming. Evol Comput 2008; 16(4):461–481.
74. Srinivasan R, Singh M. Laser backscattering and transillumination imaging of human tissues and their equivalent phantoms. IEEE Trans Biomed Eng 2003; 50(6):724–730.
75. Sui B, Gao P, Lin Y, Qin H, Liu L, Liu G. Noninvasive determination of spatial distribution and temporal gradient of wall shear stress at common carotid artery. J Biomech 2008; 41(14):3024–3030.
76. Sundareswaran KS, Frakes DH, Fogel MA, Soerensen DD, Oshinski JN, Yoganathan AP. Optimum fuzzy filters for phase-contrast magnetic resonance imaging segmentation. J Magn Reson Imaging 2009; 29(1):155–165.
77. Suo J, Oshinski JN, Giddens DP. Blood flow patterns in the proximal human coronary arteries: relationship to atherosclerotic plaque occurrence. Mol Cell Biomech 2008; 5(1): 9–18.
78. Sweet WH. The uses of nuclear disintegration in the diagnosis and treatment of brain tumor. N Engl J Med 1951; 245(23):875–878.
79. Swietlik D, Bandurski T, Lass P. Artificial neural networks in nuclear medicine. Nucl Med Rev Cent East Eur 2004; 7(1):59–67.
80. Templeton A, Liebschner M. A hierarchical approach to finite element modeling of the human spine. Crit Rev Eukaryot Gene Expr 2004; 14(4):317–328.
81. Unnikrishnan R, Pantofaru C, Hebert M. Toward objective evaluation of image segmentation algorithms. IEEE Trans Pattern Anal Mach Intell 2007; 29(6):929–944.
82. Wang D. The time dimension for scene analysis. IEEE Trans Neural Netw 2005; 16(6):1401–1426.
83. Weiskopf N, Sitaram R, Josephs O, Veit R, Scharnowski F, Goebel R, Birbaumer N, Deichmann R, Mathiak K. Real-time functional magnetic resonance imaging: methods and applications. Magn Reson Imaging 2007; 25(6):989–1003.
84. Welzel J. Optical coherence tomography in dermatology: a review. Skin Res Technol 2001; 7(1):1–9.

85. White CR, Stevens HY, Haidekker M, Frangos JA. Temporal gradients in shear, but not spatial gradients, stimulate ERK1/2 activation in human endothelial cells. Am J Physiol Heart Circ Physiol 2005; 289(6):H2350–H2355.
86. Xu XY, Collins MW. A review of the numerical analysis of blood flow in arterial bifurcations. Proc Inst Mech Eng H 1990; 204(4):205–216.
87. Yuasa T, Tanosaki S, Tagaki M, Sasaki Y, Taniguchi H, Devaraj B, Akatsuka T. Transillumination optical sensing for biomedicine and diagnostics: feasibility of early diagnosis for rheumatoid arthritis. Anal Sci 2001; 17:i515–i518.
88. Zhang S, Abbey CK, Eckstein MP. Virtual evolution for visual search in natural images results in behavioral receptive fields with inhibitory surrounds. Vis Neurosci 2009; 26(1):93–108.
89. Zhao W, Choi JH, Hong GR, Vannan MA. Left ventricular relaxation. Heart Fail Clin 2008; 4(1):37–46.
90. Zurada JM. Introduction to Artificial Neural Systems. Boston: West Publishing, 1992.

2

SURVEY OF FUNDAMENTAL IMAGE PROCESSING OPERATORS

Many image processing operators covered in this book are based on, or derived from, more fundamental image processing operators that are in wide use. A survey of these operators is presented in this chapter to create a foundation on which subsequent chapters build.

For the purpose of this book, the image is a matrix of spatially discrete image values. The matrix may be two- or three-dimensional. Higher dimensions are possible, but images of more than four dimensions are rarely found. Three-dimensional images may be either volumetric images or time sequences of two-dimensional images (stacks). Four-dimensional images are generally time sequences of three-dimensional images. Depending on the modality, the matrix elements may be arranged on an isotropic grid (i.e., the distance to neighbors is the same in all directions) or anisotropically. In volumetric image modalities such as computed tomography and magnetic resonance imaging, the distance to the axial neighbors is often much larger than the distance to the neighbors in the main image plane. Although the image values are known only on the discrete coordinates, images are generally represented as if the image values extend halfway to the nearest neighbor. When images are displayed, the image values are represented on a gray scale or in color shades. Image representation is covered in more detail in Chapter 13.

Since computer memory is limited, the image values themselves are also discrete. It is very common to allocate 8 bits for one image element (termed a *pixel*, or in volumetric images, a *voxel*). Eight bits allow for $2^8 = 256$ discrete values in any pixel. In-between values are rounded to the nearest allowable 8-bit value. Sometimes,

Advanced Biomedical Image Analysis, By Mark A. Haidekker

image modalities provide a higher bit depth. Some digital cameras provide 10 or 12 bits/pixel. Most computed tomography devices provide 12 bits/pixel. High-quality scanners used to digitize x-ray film provide up to 16 bits/pixel. Some image processing operations yield fractional values, and floating-point storage for the pixel value is useful (although few image processing software support floating-point data). However, even floating-point values have limited precision, and rounding errors need to be taken into account.

The image formation process also introduces errors, with the consequence that the pixel value deviates from an ideal but inaccessible image value. Often, it is sufficient to consider Gaussian blurring and additive noise to model the image formation errors. A certain degree of blurring can be assumed when a small feature in the object (an idealized point source) is represented by a broader intensity peak in an image. An exaggerated example would be the digital photograph of a back-illuminated pinhole with an out-of-focus lens. The two-dimensional intensity function that is the image of an idealized point source, called a *point-spread function*, provides information on the level of detail that an image modality can provide. Moreover, the sensor elements and subsequent amplifiers introduce some noise. Often, this noise is sufficiently well described as an independent deviation of each pixel value from an idealized (but inaccessible) value by a small random displacement ϵ, whereby the displacements have zero mean and a Gaussian distribution (additive Gaussian noise). In addition, the rounding of the (usually analog) measured value to an integer image value introduces digitization noise.

2.1. STATISTICAL IMAGE DESCRIPTION

When each pixel value is regarded as an independent random process, the overall probability that a pixel assumes a specific intensity value can be computed. A first-order histogram is a representation of the intensity counts or probability values. In the simplest case, the number of pixels with an image value of I are counted and the counts are displayed over the image value I. Alternatively, the counts divided by the total number of pixels provide the probability histogram. In some cases it is impractical to provide individual probabilities for each possible image value. For example, in an image with 12-bit resolution, 4096 values are possible, with an average probability of only 0.24%. In such a case, individual values are grouped in bins, and each bin has a range of, for example, eight consecutive image values. In this example, the first bin contains the cumulative probability for values 0 through 7, the second contains the cumulative probability for values 8 through 15, and so on. The details of histogram computation are given in Algorithm 2.1.

The example histogram shown in Figure 2.1 illustrates the image value distribution of the image region (without background) of an ultrasound image of the pancreas. The image values (abscissa) relate to image gray values, as indicated by the gray-scale gradient underneath the abscissa. Several properties of the image can be obtained from the histogram. Special histogram values are the image's minimum and maximum intensity values, the median value (which divides the area under the histogram into

```
// Part 1: Determine image minimum and maximum values and bin size
nbins=256;                                          // Specify number of bins
imin=IM(0,0); imax=imin;                            // initial values
for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)
      if (imin>IM(x,y)) then imin=IM(x,y);
      if (imax<IM(x,y)) then imax=IM(x,y);
   endfor;
endfor;
delta = nbins/(imax-imin);                          // delta = size of one bin

allocate histx[nbins], histy[nbins],
         histc[nbins];                              // Create space for
                                                      histogram tables

// Part 2: Sum up image values to get histogram

for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)
      bin = (IM(x,y)-imin)*delta;
      if ((bin>=0) and (bin<nbins)) then  // Ensure that we don't
                                             have a runaway bin number
            histy[bin] = histy[bin]+1;      // Increment that bin
      endif;
   endfor;
endfor;

// Part 3: Normalize histogram and compute the bin center in histx
//         and also compute the cumulative histogram in histc

cumul=0;
for (i=0 while i<nbins increment i=i+1)
   histy[i] = histy[i] / (xmax*ymax);       // normalization
   cumul = cumul+histy[i];
   histc[i] = cumul;                        // cumulative histogram
   histx[i] = imin + (i+0.5)*delta;         // bin center
endfor;
```

Algorithm 2.1 Computation of an image histogram. The input image `IM(x,y)` with size `xmax` and `ymax` is not restricted to 8 bits and may have arbitrary values. Therefore, a number of bins, `nbins`, must be specified. The output is a table with the x values (bin center values) in `histx` and the y values (probabilities) in `histy`. A cumulative histogram is computed in `histc`.

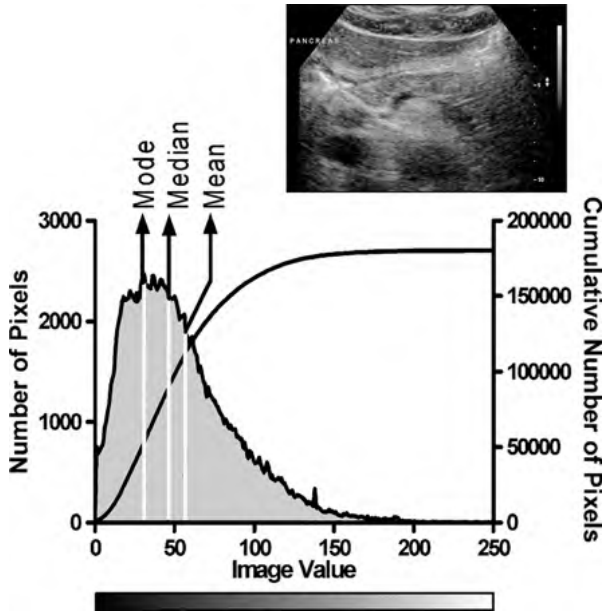


FIGURE 2.1 Example of a histogram. Shown is the gray-value distribution of an ultrasound image of the pancreas of a healthy volunteer (inset). Underneath the abscissa is a gray-value gradient that relates the image value to the gray shade. Two prominent values exist in a histogram: the median value, which splits the area under the histogram into two equal parts, and the mode value, the image value with the highest incidence. In addition, the mean value is indicated, but the mean value is not obvious from the histogram. The solid, monotonically increasing line (right *y*-axis) represents the cumulative histogram (i.e., the area under the histogram).

two equal parts), the mode value (the peak value of the histogram), and information on how the image values are distributed: for example, in a Gaussian or hyperbolic shape. In this example, the histogram is not Gaussian in shape but, rather, is skewed to the right. Two additional important values are the mean value μ and the standard deviation σ:

$$\mu = \frac{1}{N}\sum_{i=0}^{I_{\max}} i n_i \qquad \sigma^2 = \frac{1}{N}\sum_{i=0}^{I_{\max}} (i-\mu)^2 n_i \tag{2.1}$$

where i indicates intensity values ranging from 0 to $I_{\max}$ and n_i is the number of pixels with the intensity value i. With N being the total number of pixels, the value of n_i/N is the probability of intensity i. The mean intensity μ coincides with the median value only in perfectly symmetric histograms. The standard deviation provides information about the intensity spread. Particularly in digital photography (and related image sources such as light microscopy), the histogram provides instant information about

the quality of the exposure. Images with low contrast have narrow histograms. Broad histograms indicate images with good contrast because they make good use of the image value range available. Figure 2.1 also shows the cumulative histogram $N_c(I)$, defined as

$$N_c(I) = \sum_{i=0}^{I} n_i \tag{2.2}$$

The function $N_c(I)$ increases monotonically with I. A histogram with broadly distributed values shows a cumulative histogram with a homogeneous slope, while histograms with narrow regions of high probability have few sections with a steep slope and wide regions with almost zero slope, as is the case toward the high-intensity values in Figure 2.1.

We mentioned in Chapter 1 that image values often have physical relevance depending on the medical image modality. An example can be seen in Figure 2.2. Shown is the histogram of image values in a computed tomography slice of the chest. The image values are calibrated in Hounsfield units (HU), a metric for x-ray absorption relative to water as defined by

$$I(\text{HU}) = 1000 \cdot \frac{\mu - \mu_{\text{water}}}{\mu_{\text{water}}} \tag{2.3}$$

where I is the image value in Hounsfield units, μ the corresponding x-ray absorption coefficient, and μ_{water} the x-ray absorption coefficient of water. Under this definition,

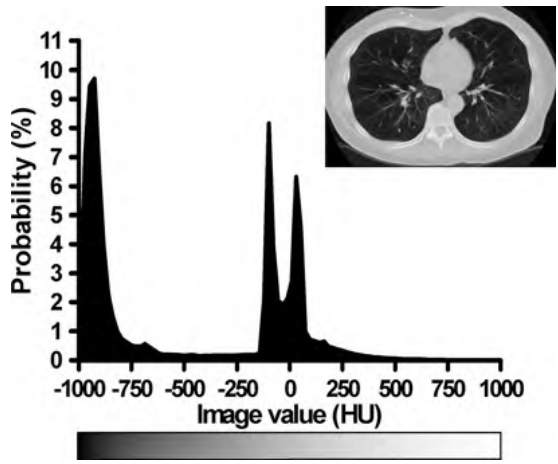


FIGURE 2.2 Sample histogram of a CT slice of the chest (inset). The image values are given in Hounsfield units (HU), and a gray-scale gradient below the abscissa relates image values to gray tones. The peak closest to −1000 HU represents the lung interior, which is mostly air. The two peaks around 0 HU represent adipose and muscular tissue.

water has 0 HU and air has −1000 HU. Since most of the image area covers air (outside the patient, inside the lungs), the most prominent peak is near −1000 HU, and its slight shift toward higher values is caused by the lung tissue. The other two peaks are related to adipose tissue and muscle tissue, respectively. Adipose tissue ranges from about −30 to −200 HU, and muscle tissue lies between 50 and 150 HU. Bone has high HU values, ranging typically from 300 to 1000 HU, but not enough bone is present in the image to cause a noticeable histogram peak. The bins of the histogram in Figure 2.2 are 22 HU wide, and a total of 90 bins are displayed. A histogram such as the example in Figure 2.2 is called *multimodal*, because each entity (i.e., air, tissues) has its own distinct histogram mode. In such a case, suitable threshold values can be determined to separate the features by image intensity. Intensity-based thresholding is covered in Section 2.4.

2.2. BRIGHTNESS AND CONTRAST MANIPULATION

Software for viewing images often makes it possible to manipulate brightness and contrast. The displayed image values are remapped by applying a linear or nonlinear function f in a manner that

$$I'(x,y) = f\left(\frac{I(x,y) - I_{\min}}{I_{\max} - I_{\min}}\right) D_{\max} \tag{2.4}$$

where $I(x,y)$ is the original image value, $I'(x,y)$ the remapped image value, and $I_{\min}$ and $I_{\max}$ the smallest and largest intensity values in the histogram, respectively. The argument of f is the image value, normalized to the range 0 to 1, and f would typically return values in the same range. $D_{\max}$ is the maximum allowable image value in the display unit. Most display units (such as computer monitors and printers) allow the value to range from 0 (black) to 255 (white), and consequently, $D_{\max} = 255$. In medical imaging, the contrast or brightness function is often referred to as a window or center. These functions are frequently used for visualization purposes and do not change the actual image values. A closer look at manual contrast enhancement functions is provided in Section 13.1.

Image contrast can also be enhanced permanently. In this case the original image value is replaced by the remapped value computed by Equation (2.4). In the simplest case of contrast enhancement, a narrow histogram can be broadened by *histogram stretching*, a technique whereby image values are remapped according to the equation

$$I'(x,y) = \frac{I(x,y) - I_{\min}}{I_{\max} - I_{\min}} M \tag{2.5}$$

where M determines contrast. If M coincides with $D_{\max}$ in Equation (2.4), no contrast enhancement takes place. If M becomes larger than $D_{\max}$, the histogram becomes stretched and its center contrast is amplified. With this amplification, some image values may fall outside the permissible value range and need to be clamped (saturation). Due to the discrete nature of the image values, the remapped histogram will be

sparse, with many values that do not occur in the remapped image. Although contrast is enhanced, no information is gained.

A related remapping operation creates a histogram with approximately equal probabilities for all values; this operation is defined as

$$I' = \sum_{i=0}^{I} P(i) \tag{2.6}$$

where the intensity I gets remapped to the new intensity I' and $P(i)$ is the histogram probability of intensity i. The remapping function defined in Equation (2.6) is designed to provide the most uniform slope possible in the cumulative histogram with the given image value distribution. An implementation for the histogram equalization operation is given in Algorithm 2.2.

```
// Part 1: Set up a translation table
allocate xlat[nbins];                                  // image value translation table
for (i=0 while i<nbins increment i=i+1)
   xlat[i] = histc[i]*(imax-imin)+imin;
endfor;

// Part 2: Run over the entire image and translate image values. Interpolate between bins.

for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)
      buf = IM(x,y);                          // Extract image value to manipulate
      bin = int((buf - imin)*delta);          // the bin number (integer)
      t = (buf - (bin/delta+imin))*delta;     // for interpolation, 0 ≤ t < 1
      buf = (1-t)*xlat[bin]+t*xlat[bin+1];    // replacement value
      IM(x,y) = buf;                          // Actually replace the image
   endfor;                                      value
endfor;
delete (xlat);                                 // translation table no longer needed
```

Algorithm 2.2 Histogram equalization as an example of image value remapping [Equation (2.6)]. This algorithm relies on the computation of `histc`, the cumulative histogram, as well as `imin`, `imax`, `delta`, and `nbins` in Algorithm 2.1. The input image `IM(x,y)`, with size `xmax` and `ymax`, is the same as for Algorithm 2.1, and its values get modified by the histogram equalization process. Note that the final value of `buf` needs to be rounded if `IM(x,y)` has integer discretization.

2.3. IMAGE ENHANCEMENT AND RESTORATION

Image enhancement and restoration use similar operators but are driven by different goals. Image restoration is a process specifically designed to counteract known image degradation: for example, to improve the overall point-spread function of an

image. Image enhancement is a user-driven process to meet specific image quality criteria: for example, noise reduction, sharpening, or edge enhancement. Operators for image enhancement and restoration are called *filters*. Two different types of filters exist: spatial-domain filters, which are generally based on convolution operations, and frequency-domain filters, which apply a defined frequency-response function. Frequency-domain filters are covered in more detail in Section 3.2; in this section we focus on convolution-based filters. The one-dimensional convolution of a function $f(t)$ with another function, $g(t)$, often referred to as the *kernel*, is defined as

$$(f \circledast g)(t) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)\, d\tau \tag{2.7}$$

where the symbol $\circledast$ indicates the convolution operation. The integral in Equation (2.7) needs to be evaluated for all possible values of t. It can be seen from the equation that f and g are shifted against each other for different values of t. The center of the function g is always where $t = \tau$, and the center of f is always where $\tau = 0$. The convolution operation becomes more intuitive in the discrete form, particularly when a kernel with finite support ($-m$ to $+m$) is assumed:

$$(f \circledast g)_k = \sum_{i=k-m}^{k+m} f_i g_{i-k} \tag{2.8}$$

Let us assume two kernel examples, with $m = 1$ in both cases. For kernel g_1, the element at index -1 is 0.25, the element at index 0 is 0.5, and the element at index 1 is 0.25. Kernel g_2 contains the elements -1, 0, and 1 at index -1, 0, and 1, respectively. With $m = 1$, the summation takes place for $i = k - 1$, k, and $k + 1$. In this special case, the convolution in Equation (2.8) becomes the summation

$$\begin{aligned}(f \circledast g_1)_k &= 0.25 f_{k-1} + 0.5 f_k + 0.25 f_{k+1} \\ (f \circledast g_2)_k &= -f_{k-1} + f_{k+1}\end{aligned} \tag{2.9}$$

The first example, with kernel g_1, is a moving-window weighted average; the kth element of the convolution result is computed from the weighted average of the kth element of the original time series and its two neighbors. The second example, with kernel g_2, turns out to be a central-difference formulation that computes the first derivative except for a scaling factor.

In two dimensions, the convolution equation extends to

$$(f \circledast g)_{x,y} = \sum_{j=y-m}^{y+m} \sum_{i=x-m}^{x+m} I(i,j)g(i - x, j - y) \tag{2.10}$$

where $I(x,y)$ is the original image convolved with a two-dimensional kernel g that has $(2m + 1)^2$ elements. Depending on the kernel, convolution operations can be

used, for example, to reduce noise, to sharpen the image, or to extract edges. Most often, the kernel g is quadratic and symmetric, so that the convolution becomes rotation-independent. The moving-average kernel, introduced in one dimension above [Equation (2.9)], can be extended to two dimensions and becomes

$$G(x,y) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.11}$$

This kernel is a rough approximation of a Gaussian function with a standard deviation $\sigma = 1.2$, restricted to a 3×3 support and represented by integer elements. Note that the kernel is normalized so that the sum of its elements is unity. This normalization is necessary to conserve the amplitude of the image convolved with this kernel. Another consideration is the behavior of the convolution at the edges of the image. When a 3×3 kernel is used, the computation of the first and last image rows and columns requires that we access image values outside the image range (e.g., at $x = -1$ and $y = -1$). Since image data at these coordinates generally are not available, edge behavior needs to be defined. Four different methods are commonly used. In the first method, all values outside the image area can simply be assumed to be zero. This *zero-padding method*, although very simple to implement, has the disadvantage that it tends to create a discontinuity at the image edge. This discontinuity would negatively affect smoothing filters or edge detectors. Alternatively, a second method continues the closest edge value. Under this assumption, for example, $I(x,y) = I(0,y)$ for all $x < 0$. This edge definition is frequently implemented, as it has minimal negative effects on convolution filters. The third alternative is a mirror definition; that is, in the example of the left boundary, $I(x,y) = I(-x,y)$ for all $x < 0$. This boundary definition replicates the texture irregularity (e.g., frequency components, noise) of the pixels near the boundary. In most cases it behaves in a manner very similar to the second definition. A fourth definition "tiles" the image, that is, $I(x,y) = I(x + N, y)$ for all $-N < x < 0$, where N is the number of pixels in the x-direction. This edge definition is not commonly used except in some cases where the periodicity assumption that underlies the Fourier transform plays an important role. The main disadvantage of the fourth definition is the possible occurrence of discontinuities at the boundary, very much like the zero-padding method.

The kernel in Equation (2.11) is commonly used for weak noise reduction and moderate blurring, and the convolution with this kernel is often referred to as a *smoothing operation*. Stronger smoothing is possible with different kernel values to approximate Gaussian functions with larger standard deviations. When staying within the 3×3 confines of commonly used kernels, the extreme case would consist of a kernel with $G(x,y) = 1$ for all x,y inside the 3×3 support and a normalization factor of $\frac{1}{9}$. Although this "box kernel" has a stronger smoothing action than the kernel in Equation (2.11), its frequency response shows undesirable characteristics. The convolution theorem (see Section 3.2) stipulates that a convolution of two functions corresponds to multiplication of the Fourier transforms of these functions. As can be seen in Figure 2.3, the box kernel with its abrupt transition from 1 to 0 has a

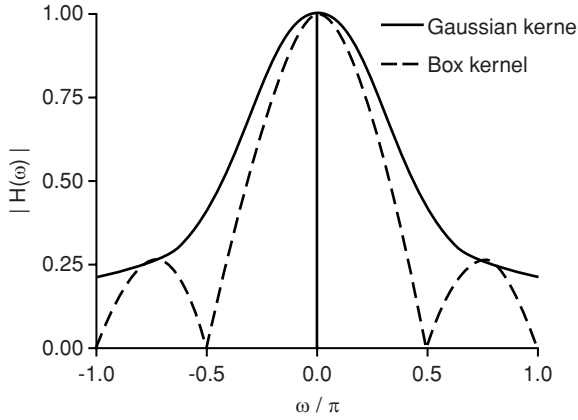


FIGURE 2.3 Frequency responses of the Gaussian and the box smoothing kernel. While the box kernel shows a steeper drop-off near low spatial frequencies ($\omega = 0$), it completely suppresses certain frequencies and allows higher frequencies to pass again. The Gaussian kernel has less smoothing action, but it does not show undesirable behavior in the stopband.

steeper transition from the low frequencies $H(\omega) = 1$ (those that are allowed to pass, the passband) to higher frequencies (those that are stopped, the stopband). However, after the box kernel reaches zero and fully suppresses the associated spatial frequency, even higher frequencies are allowed to pass again, albeit attenuated. The Gaussian kernel does not exhibit this behavior. The frequency response drops monotonically toward higher frequencies, although it drops off slower than the box kernel, and therefore its filtering action is weaker. To avoid the undesirable frequency response of the box kernel, yet provide a stronger filter with a steeper transition from the passband to the stopband, a larger kernel size is necessary to allow a smooth drop-off of the kernel values while allowing a large standard deviation. A suggested Gaussian approximation with a kernel size of 5×5 computed from a Gaussian function with a standard deviation of $\sigma = 1.7$ is

$$G(x,y) = \frac{1}{164}\begin{bmatrix} 1 & 3 & 5 & 3 & 1 \\ 3 & 10 & 14 & 10 & 3 \\ 5 & 14 & 20 & 14 & 5 \\ 3 & 10 & 14 & 10 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix} \tag{2.12}$$

The smoothing action of a convolution with the kernel in Equation (2.12) is stronger than a convolution with the kernel in Equation (2.11). This conclusion can be reached by a different consideration. If the kernel in Equation (2.11) is applied twice, the linearity of the convolution operation allows the identity

$$(I \circledast g) \circledast g = I \circledast (g \circledast g) \tag{2.13}$$

The kernel in Equation (2.11) convolved with itself leads to a 5×5 kernel somewhat similar to the 5×5 kernel in Equation (2.12). Even larger smoothing kernels can be designed in the same manner. However, computational effort for the convolution increases with the square of the kernel size, specifically with N^2m^2, where the image is of size $N \times N$ and the kernel is of size $m \times m$. Although the convolution can be implemented with fast integer operations, filtering in the frequency domain becomes more efficient when kernel sizes grow very large.

The operation complementary to smoothing is usually referred to as *sharpening*. A physical motivation for the design of a sharpening kernel was given by Russ,[22] who suggested that blurring can be modeled as a diffusion process that follows the partial differential equation

$$\frac{\partial I}{\partial t} = k \Delta I \tag{2.14}$$

where Δ is the Laplacian operator and k is the diffusion constant. In its time-discrete form, Equation (2.14) becomes $I_{n+1} = I_n + \tau k \, \Delta I_n$, where τ is the time step from iteration n to $n + 1$. An approximate reversal of the diffusion process in time would be $I_{n-1} = I_n - \tau k \, \Delta I_n$, where the Laplacian of the blurred image gets subtracted from the image. The Laplacian operator, or second-derivative operator, can be discretized by finite differences. One-dimensional kernels of $[-1 \;\; 1 \;\; 0]$ and $[0 \;\; -1 \;\; 1]$ realize discrete asymmetrical differences. The difference of the first-order differences becomes $[1 \;\; -2 \;\; 1]$. In two dimensions, the Laplacian operator is often approximated by kernel L_4 or L_8:

$$L_4(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad L_8(x,y) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.15}$$

To enhance a blurred image, its convolution with the Laplacian kernel gets subtracted from the image ($I' = I - kI \circledast L_8$), and because of the linearity of the convolution, the operation can be combined into a single convolution with the sharpening kernel S defined for $k = 1$:

$$S(x,y) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.16}$$

In some software packages (NIH Image and ImageJ), the convolution with S in Equation (2.16) is termed "sharpen more," and the regular sharpening operation uses a kernel with weaker action where $k = 0.25$:

$$S(x,y) = \frac{1}{4} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.17}$$

A convolution with a sharpening kernel has a highpass character; that is, high spatial frequencies get amplified. Small image detail and edges contain high-frequency components. Therefore, edges become more prominent after a sharpening operation. In addition, noise gets amplified.

Two more convolution-based filters with a highpass character are unsharp masking and DoG operators. *Unsharp masking* is an operation to remove inhomogeneous background from the image. The image is convolved with a very large Gaussian kernel so that ideally all detail information is removed by blurring. The blurred image is then subtracted form the original image. The *difference-of-Gaussian* (DoG) *operator* is closely related to the Laplacian operator. Again, larger kernel sizes are used. The DoG kernel is a Gaussian function with lower peak value and larger standard deviation subtracted from another Gaussian function that has a smaller standard deviation and a larger peak value.

The noise-amplifying property of highpass filters is even more pronounced with the Laplacian kernel L in Equation (2.15). For this reason, a convolution with L rarely yields satisfactory results. However, Gaussian smoothing (for noise reduction) and computation of the second derivative with the Laplacian operator can be combined into a commonly used edge detection method, the *Laplacian-of-Gaussian* (or LoG) *operator*. An image first gets convolved with a large Gaussian kernel, and the result is convolved with the Laplacian. With the linearity of the convolution operation, the Gaussian and Laplacian kernels can be joined into a precomputed LoG kernel. Because a strong smoothing action is desired, LoG kernels are usually large. The principle of the LoG-based edge detector is demonstrated in Figure 2.4. It is fundamental to have strong blurring to create a soft transition instead of a sharp edge. This does not only suppress noise. The second derivative of the blurred edge crosses

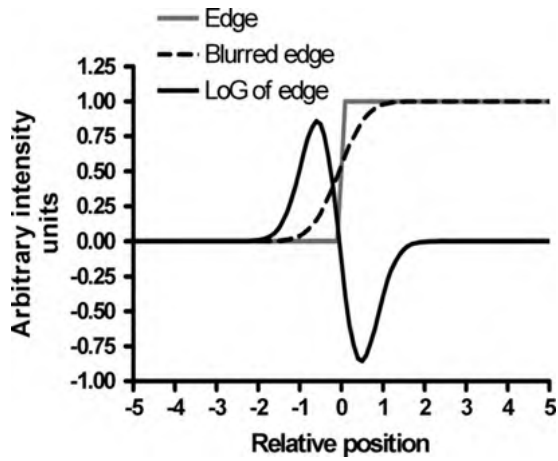


FIGURE 2.4 Edge detection with the Laplacian-of-Gaussian (LoG) operator. A discontinuity (edge) is blurred by a convolution with a Gaussian kernel. The second derivative of the blurred edge crosses through zero exactly at the position of the edge.

```
lx=int(kx/2); ly=int(ky/2);                    // kernel offset values
allocate IM2[xmax,ymax];                       // Prepare space for output image
w=0;                                           // Prepare kernel weight
for (y=0 while y<ky increment y=y+1)
   for (x=0 while x<kx increment x=x+1)
      w = w+K(x,y);
   endfor;
endfor;

// Run over the entire image. For each pixel, compute neighborhood sum with kernel.

for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)

      // This is the inner summation loop that realizes Equation (2.10)

      sum=0;
      for (y1=y-ly while y1<=y+ly increment y1=y1+1)
         for (x1=x-lx while x1<=x+lx increment x1=x1+1)
            x2=x1; y2=y1;
            if (x1<0) then                     // edge handling for x
              x2=0;
            elseif (x1>=xmax) then
               x2=xmax-1;
            endif;
            if (y1<0) then                     // edge handling for y
              y2=0;
            elseif (y1>=ymax) then
               y2=ymax-1;
            endif;
            sum = sum + IM(x2,y2)*K(x1-x+lx,y1-y+ly);
         endfor;
      endfor;

      // Store the convolution result (weighted sum) for this pixel

      IM2(x,y) = sum/w;

   endfor;
endfor;
```

Algorithm 2.3 Convolution of an image `IM(x,y)` with a kernel `K(x,y)`. The image size is `xmax` and `ymax`. The kernel size is `kx` and `ky`, both of which need to be odd-valued integers. A convolution cannot be performed in-place. Therefore, the result is stored in `IM2(x,y)`. At the image edges, the image is assumed to continue its edge value indefinitely. The kernel does not need to be normalized; this is handled with the variable `w`. The kernel `K(x,y)` needs to be filled with values in advance: for example, from Equation (2.11) or (2.15).

through zero at the location of the edge, and a wider (less steep) transition allows for easier detection of the zero-valued edge points. An example of how a convolution of an image with an arbitrary kernel can be implemented is shown in Algorithm 2.3.

Edge enhancement and edge detection are very important operations in image processing. Many edge detection operators were devised early in the age of computerized image processing, and most build on finite differences. In all cases, special provisions are taken to ensure that edges are enhanced irrespective of their orientation. One of the earliest edge detectors is Roberts' cross,[21] a finite-difference operator that computes two gradients, G_1 and G_2, in perpendicular directions by using forward differences:

$$\begin{aligned} G_1(x,y) &= \sqrt{I(x+1,\, y+1)} - \sqrt{I(x,y)} \\ G_2(x,y) &= \sqrt{I(x+1,\, y)} - \sqrt{I(x,\, y+1)} \end{aligned} \tag{2.18}$$

The square roots in Equation (2.18) are intended to approximate a gamma-style contrast enhancement for the darker tones.[21] The two gradient pixels were then combined with the final gradient $G(x,y) = \sqrt{G_1(x,y)^2 + G_2(x,y)^2}$. Since the computation of a square root was a very computationally expensive operation, alternative formulations used the maximum of the absolute values of G_1 and G_2 or the sum of the absolute values of G_1 and G_2. These formulations made Roberts' cross dependent on the orientation of the edge. Furthermore, this operator is very sensitive to noise, and the use of discrete differences with a single-pixel distance shifted the edge by one-half a pixel. By using central differences, the pixel shift is eliminated and the noise sensitivity is somewhat reduced, because the central difference is the average of the forward and backward differences, as can be demonstrated in one dimension as:

$$\frac{f(t+\Delta t) - f(t-\Delta t)}{2\Delta t} = \frac{1}{2}\left[\frac{f(t+\Delta t) - f(t)}{\Delta t} + \frac{f(t) - f(t-\Delta t)}{\Delta t}\right] \tag{2.19}$$

The use of central differences leads directly to the Sobel, Prewitt, compass, and Kirsch operators, which are related. Prewitt[19] and Sobel[24] proposed averaging neighboring values perpendicular to the direction of the central difference and suggested a convolution with the gradient detector kernels G_X and G_Y to obtain convolution results with emphasized edges of horizontal and vertical orientation, respectively:

$$G_X(x,y) = \begin{bmatrix} -1 & -a & -1 \\ 0 & 0 & 0 \\ 1 & a & 1 \end{bmatrix} \qquad G_Y(x,y) = \begin{bmatrix} -1 & 0 & 1 \\ -a & 0 & a \\ -1 & 0 & 1 \end{bmatrix} \tag{2.20}$$

where $a = 1$ for the Prewitt operator and $a = 2$ for the Sobel operator. For edge enhancement, the original image is convolved with G_X and G_Y separately to yield an image I_X, where horizontally oriented edges are enhanced, and an image I_Y, where vertically oriented edges are enhanced. The final gradient image $G(x,y)$ is computed

through $G(x,y) = \sqrt{I_X(x,y)^2 + I_Y(x,y)^2}$. To compute the exact edge magnitude, a normalization factor for G_X and G_Y of $\frac{1}{6}$ for the Prewitt operator and $\frac{1}{8}$ for the Sobel operator is needed, but in practical implementations, this factor is rarely found.

Each Sobel kernel can be interpreted as a rough approximation of the first derivative of a Gaussian function in one direction. G_X is a derivative kernel that detects discontinuities in the vertical direction (i.e., horizontal edges), and G_Y is a derivative that detects discontinuities in the horizontal direction. Analogous to the Gaussian kernel [Equations (2.11) and (2.12)], larger kernel sizes for edge detection are possible. For example, the partial derivative of the two-dimensional Gaussian function $g(x,y)$ toward x,

$$\frac{\partial g(x,y)}{\partial x} = -\frac{2x}{\sigma^2} \exp\left(-\frac{x^2+y^2}{\sigma^2}\right) \tag{2.21}$$

can be used to obtain a 7×7 kernel that approximates the values for Equation (2.21) with $\sigma = 2$:

$$\frac{\partial g(x,y)}{\partial x} \approx \begin{bmatrix} 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 3 & 6 & 7 & 0 & -7 & -6 & -3 \\ 6 & 14 & 15 & 0 & -15 & -14 & -6 \\ 8 & 18 & 20 & 0 & -20 & -18 & -8 \\ 6 & 14 & 15 & 0 & -15 & -14 & -6 \\ 3 & 6 & 7 & 0 & -7 & -6 & -3 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \end{bmatrix} \tag{2.22}$$

Such a kernel combines the derivative operation with a smoothing operation and therefore reduces noise in the final edge image. The corresponding partial derivative toward y (to detect horizontal edges) is obtained by rotating the matrix in Equation (2.22) by 90°. The combination of smoothing and derivative operations can again be explained by the linearity of the convolution operation. Analogous to Equation (2.13), the derivative operator can be moved in front of the convolution operation,

$$I(x,y) \circledast \frac{\partial g(x,y)}{\partial x} = \frac{\partial}{\partial x}[I(x,y) \circledast g(x,y)] \tag{2.23}$$

which indicates that the convolution of an image I with the partial first derivative of a Gaussian function is identical to computing the first derivative of the same image I convolved (smoothed) with the original Gaussian function. Extending this consideration to the second derivative leads to the Laplacian-of-Gaussian operator.

The Kirsch operator[14] and the related compass operator make use of the same principle as the Sobel operator, but instead of combining two perpendicular directions, the Kirsch and compass operators have eight kernels, K_1 through K_8, to define all

eight possible edge directions in a 3×3 neighborhood. The compass operator uses the same kernel structure as the Sobel operator. The first four kernels are:

$$K_1(x,y) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad K_2(x,y) = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$
$$K_3(x,y) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad K_4(x,y) = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \tag{2.24}$$

Each of the kernels has its elements rotated 45° from the preceding one, and the next four kernels have the opposite sign: $K_5 = -K_1$, $K_6 = -K_2$, $K_7 = -K_3$, and $K_8 = -K_4$. In a similar manner, the Kirsch operator[14] makes use of a different definition of kernels K_1 through K_4:

$$K_1(x,y) = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \qquad K_2(x,y) = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$
$$K_3(x,y) = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \qquad K_4(x,y) = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \tag{2.25}$$

The idea behind the Kirsch and compass operators is to avoid the computationally expensive square-root computation by accepting a larger number of convolutions, which is faster overall when integer arithtmetic is used. For each pixel, eight convolution values are computed (only four by actual convolution and four by sign change), and the maximum of the eight values is taken as the new image value.

A comprehensive approach to edge detection was developed by Canny,[2] who combined Gaussian smoothing, finite-difference edge detection, and nonmaximum suppression into his edge detection algorithm. Nonmaximum suppression is a component of the algorithm that removes pixels from the set of edge points when they either fall below a certain, selectable threshold value, or when the gradient magnitude in perpendicular directions is similar, or when a neighborhood pixel has the same edge direction but a higher edge magnitude. As a consequence, the edges are thinned. Finally, Canny proposed the use of hysteresis thresholding, which makes use of local connectedness (see Section 2.4) and removes isolated sets of low and intermediate edge magnitude. This comprehensive approach is comparatively costly in terms of computational effort, and Canny has suggested several efficiency improvements for the convolution, difference, and nonmaximum suppression stages.[3] In addition, Deriche developed a recursive filter implementation[6] that advertises itself for use in fast integer-based implementations and for implementations in hardware filters or digital signal processors.

Another comprehensive approach for edge detection was proposed by Frei and Chen.[7] By convolving the image with nine different kernels that form an orthogonal

system, each pixel is assigned a nine-dimensional feature vector. The nine convolution kernels are:

$$F_0(x,y) = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad F_1(x,y) = \frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & \sqrt{2} & -1 \end{bmatrix}$$

$$F_2(x,y) = \frac{1}{2\sqrt{2}}\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \qquad F_3(x,y) = \frac{1}{2\sqrt{2}}\begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix}$$

$$F_4(x,y) = \frac{1}{2\sqrt{2}}\begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix} \qquad F_5(x,y) = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$F_6(x,y) = \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix} \qquad F_7(x,y) = \frac{1}{6}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$F_8(x,y) = \frac{1}{6}\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} \tag{2.26}$$

It can be seen that kernels F_1 and F_2 are typical edge detector kernels for horizontal and vertical edges. Kernels F_3 and F_4 are designed to amplify ripples, and together with F_1 and F_2 they comprise the *edge space*. Kernels F_5 and F_6 are sensitive to lines of single-pixel width and diagonal or horizontal/vertical orientation. Finally, kernels F_7 and F_8 are approximations of discrete Laplacian functions. The four-dimensional subspace created by convolution kernels F_5 through F_8 is called the *line space*. Let $\vec{B} = [b_0, b_1, \ldots, b_8]$ be the feature vector for one pixel of the image, where the components b_0 through b_8 have been generated by convolution of the image with kernels F_0 through F_8. The vector can be projected into edge space or line space:

$$\cos\theta_E = \sqrt{\frac{\sum_{i=1}^{4} b_i^2}{\sum_{k=0}^{8} b_k^2}} \qquad \cos\theta_L = \sqrt{\frac{\sum_{i=5}^{8} b_i^2}{\sum_{k=0}^{8} b_k^2}} \tag{2.27}$$

where θ_E is the projection angle into edge space and θ_L is the projection angle into line space. A low projection angle θ_E indicates a high probability that the associated pixel is part of an edge. Conversely, a projection angle of θ_E close to 90° indicates a low probability of the pixel being part of an edge. For lines, θ_L carries similar information. The projection of additional subspaces is possible in a similar manner. For example, Frei and Chen demonstrated the effects of the projection of "ripple space" and "Laplacian space,"[7] but these projections are used infrequently.

In addition to the edge magnitude, direction-dependent filter masks can provide the edge direction. In the case of the Kirsch and compass kernels, the kernel that produces the highest convolved pixel value determines the edge direction: If convolution with

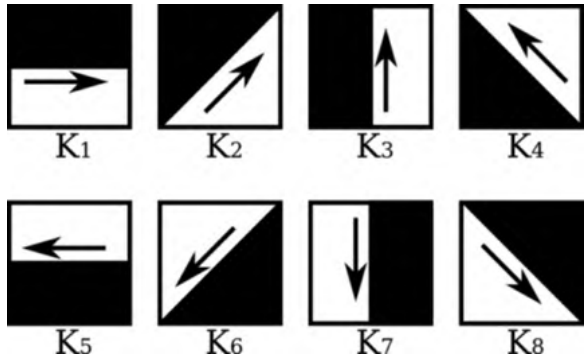


FIGURE 2.5 Eight possible edge orientations that can be detected with the compass operator [Equation (2.24)]. White indicates a higher image value than black, and the edge direction is defined so that the higher value is to the right of the direction arrow.

K_1 in Equation (2.24) produces the highest pixel value, the edge is horizontal; for K_2 it is diagonal (southwest to northeast), for K_3 it is vertical, and for K_4 it is diagonal northwest to southeast. K_5 to K_8 act in a similar manner, but the side of the higher image value is mirrored. Figure 2.5 illustrates the eight possible directions that can be detected with the compass operator in Equation (2.24). The Sobel and Prewitt operators provide edge directions as well. In this case, the arctangent function can be used:

$$\phi(x,y) = \text{atan2}\left(\frac{I_y(x,y)}{I_x(x,y)}\right) \tag{2.28}$$

where atan2 is the four-quadrant arctangent function as implemented in many programming languages, and I_x and I_y are the intermediate convolution results of the image with G_X and G_Y as defined in Equation (2.20). When using Equation (2.28), the edge direction is continuous, as opposed to the eight discrete directions obtained from the Kirsch and compass operators.

Images that underwent edge enhancement usually need additional processing steps to extract clean edges. Any edge enhancement operation is highly noise-sensitive, and image noise may produce many pixels of high magnitude that are not part of an actual edge. These processing steps may include histogram analysis and thresholding to retain only the most prominent edges, morphological operators (such as elimination of isolated pixels or median-axis thinning), or feature detection such as straight-line fitting as proposed by Roberts[21] or the Hough transform (Chapter 7).

In the context of image enhancement, the median filter needs to be introduced. The median filter is a powerful nonlinear noise-reduction filter for noise that is not Gaussian. Convolution with a Gaussian kernel (smoothing operation) is most suitable for reducing additive noise with a Gaussian distribution and—more important—zero mean value. Other types of noise, particularly *salt-and-pepper noise* (also called *shot noise*, i.e., each pixel has a certain probability of being either black or white), cannot

be filtered well with Gaussian filters because extreme values of salt-and-pepper noise get "smeared" over the neighborhood. A median filter is ideally suitable to remove runaway values. A median filter acts on a $n \times n$ neighborhood but is not convolution-based. Rather, all image values in the neighborhood are sorted ascendingly, and the central pixel is replaced by the median value of the neighborhood pixels. Intuitively, the median filter replaces the central pixel by a value that is more typical for the neighborhood (namely, the median value) and thus eliminates pixels with runaway intensities. Since a median filter has a very high probability of changing the value of the central pixel, attenuated versions exist where the value of the center pixel is repeated k times in the sorted list for median determination.[15] Such a center-weighted median filter has a higher probability of not changing the pixel value. A more in-depth analysis of the median filter and variations that adapt to local image properties to better preserve the image values is given in Section 5.1. The effect of the conventional median filter and center-weighted median filter on salt-and-pepper noise is compared with Gaussian blurring in Figure 2.6.

The effects of various convolution-based filters are shown in Figures 2.7 and 2.8. In Figure 2.7, various lowpass and highpass operations are demonstrated, and in Figure 2.8, edge detection by the Sobel and compass operators with color-coded edge direction, edge detection with the LoG operator, and edge and line detection with the Frei–Chen operator are shown.

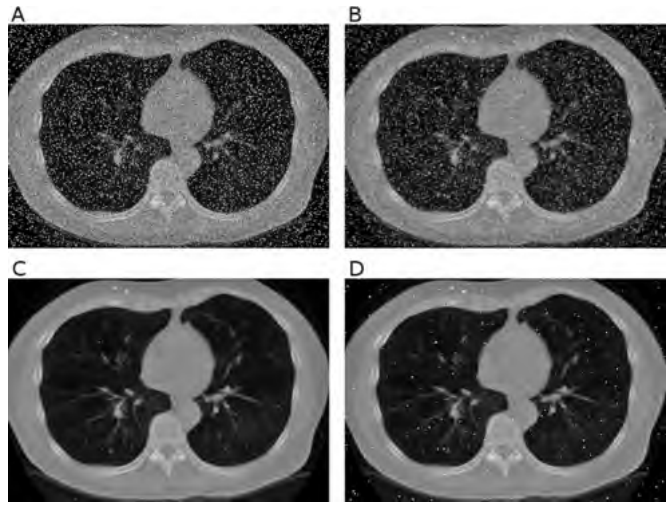


FIGURE 2.6 Effect of median filtering on salt-and-pepper noise. (A) An image spoiled with 5% salt-and-pepper noise where each pixel has a 2.5% probability of containing either a black or a white noise pixel. Gaussian blurring (B) is unsuitable for noise reduction because the extreme values are merely blurred over the neighborhood. The median filter almost completely eliminates the salt-and-pepper noise (C), but it also attenuates texture on homogeneous regions (compare to Figure 2.7A). A center-weighted median filter is less effective in removing a strong noise component but has better texture-preserving properties.

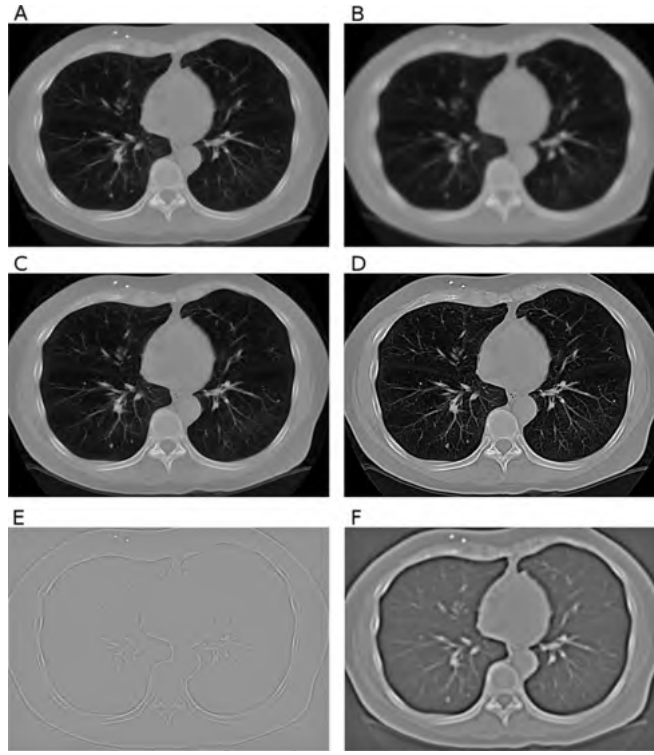


FIGURE 2.7 Demonstration of some convolution filters. The original image A is a CT slice of the chest and is 500 × 350 pixels in size. Image B was convolved with a Gaussian kernel with $\sigma = 2.1$ in a 13 × 13 neighborhood. Images C and D show the effect of sharpening: In image C the weaker sharpening mask of Equation (2.17) was used, and in image D, Equation (2.16) was applied. Image E is the result of a convolution with the Laplacian mask in Equation (2.15). Finally, image F demonstrates the effect of the DoG operator where the original image, convolved with a Gaussian kernel with $\sigma = 4$, was subtracted from the original image convolved with a Gaussian kernel with $\sigma = 1.6$. The result closely resembles the LoG operator (Figure 2.8C).

2.4. INTENSITY-BASED SEGMENTATION (THRESHOLDING)

The purpose of segmentation is to separate one or more regions of interest in an image from regions that do not contain relevant information. Regions that do not contain relevant information are called *background*. Depending on the image, segmentation can be a very complex process, and a comprehensive overview of the most relevant segmentation techniques could fill an entire book. For our purposes in this chapter, an overview of simple intensity-based techniques is given. The underlying assumption is that pixels belonging to the features of interest occupy a different value range than that of background pixels. Without loss of generality, all examples in this

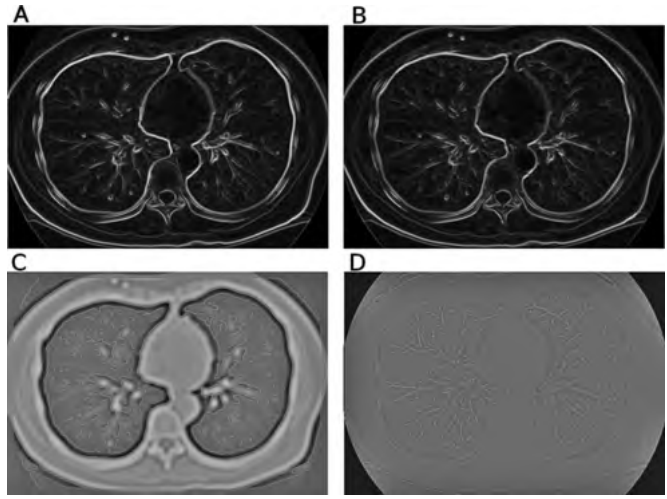


FIGURE 2.8 Demonstration of some edge detection operators. The original image is the CT slice in Figure 2.7A. Images A and B show the results of the Sobel and compass edge detectors with color-coded edge directions. Whereas the edge magnitude is very similar, the discrete nature of the edge directions in the compass operator becomes clearly visible. In image C the LoG operator with $\sigma = 3.0$ was applied, and zero crossings (edge locations) are marked in red. An apparent double edge appears at the steepest edges. Image D shows a color-coded version of the Frei–Chen operator. Red hues indicate high values (low projection angles) in edge space, with some similarity to the Sobel and compass operators. Blue–green hues indicate high values in line space, most prominent in the vascular structure and the thin traces of the ribs. (*See insert for color representation of the figure*.)

chapter assume that background pixels have lower values than feature pixels. If we assume further that the distribution of feature pixels and background pixels is approximately Gaussian, a characteristic intensity distribution with two peaks in the histogram emerges (Figure 2.9). Such a distribution is called *bimodal* because there are two mode values: one for the background and one for the feature. Intensities are normally spread around the modal values because of the additive noise and intensity inhomogeneities of the features. The simplest approach for segmentation would be the selection of a suitable intensity threshold, as indicated in Figure 2.9. All pixels with a value higher than the threshold value are classified as *feature pixels*, and all pixels with a lower value are classified as *background pixels*. Most commonly, a new image is created by using

$$I_T(x,y) = \begin{cases} 1 & \text{for } I(x,y) \geq T \\ 0 & \text{for } I(x,y) < T \end{cases} \tag{2.29}$$

where $I(x,y)$ are the original image pixels and $I_T(x,y)$ is the thresholded image. Since I_T contains only two values (1 for foreground pixels and 0 for background pixels),

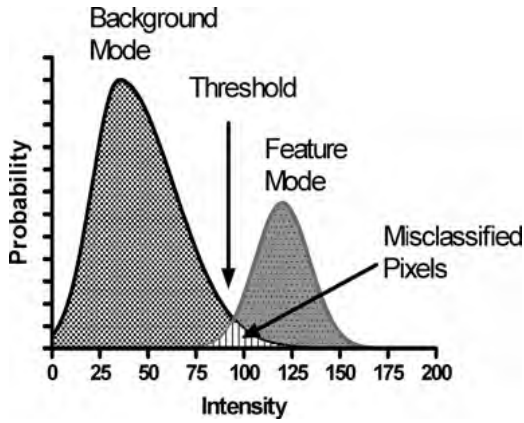


FIGURE 2.9 Bimodal histogram. In this example, background pixels are generally darker than feature pixels. However, the intensities spread because of noise and intensity inhomogeneities. Intensity-based separation can be carried out with a suitable threshold, but some pixels are misclassified.

it is called a *binary image*. It can serve as a mask because each location (x,y) of the original image has a value of 1 in the mask if this is a feature pixel.

In many cases, some feature pixels will have intensity values below the threshold value, and some background pixels will lie above the threshold value because of image inhomogeneities and additive noise. With pure intensity-based thresholding, these pixels cannot be classified correctly (misclassified pixels in Figure 2.9). Two questions need to be answered. First, is there any way to determine the optimum threshold value automatically, and second, what steps can be taken to decrease the number of misclassified pixels?

2.4.1. Automated Threshold-Finding Methods

A number of methods exist to find the valley between the two modal peaks in a histogram, and three representative examples are introduced: the isodata method[20] (also called the iterative thresholding method), Otsu's method,[17] and the entropy maximization method.[13] Iterative thresholding is arguably the easiest method to implement, and Otsu's method is one of the most widely used methods. Both methods, and other methods not described here, work under the assumption of a bimodal histogram with a well-defined valley; in practice, different thresholding methods find very similar values in ideal cases but may vary when the assumptions are not met. All methods will provide a threshold value when acting on a single-modal histogram, but the resulting threshold value may not be useful.

For *iterative thresholding*, an initial threshold $T_{k=0}$ must be specified, and a good choice is the mean gray value of the image. Ridler and Calvard[20] originally proposed

using rectangular regions at the four corners of the image as background and the rest of the image as foreground, but the final threshold is independent of the initial choice, and almost any convenient initial threshold may be used. For the iterative process, the image is then partitioned into a set of pixels S_1 with values below the threshold T_k of the current iteration and a set of pixels S_2 with values above or equal to T_k. Next, the mean gray values μ_1 and μ_2 of the pixel sets S_1 and S_2 are computed, and a new threshold, T_{k+1}, is computed as $(\mu_1 + \mu_2)/2$. The partitioning is now repeated with the new threshold value until the iteration converges with $|T_k - T_{k+1}| < \epsilon$. The convergence criterion ϵ may be a suitable small value: for example, 0.5 for integer-valued images.

Otsu's method assumes a Gaussian distribution of the image values around both modes, and the goal of Otsu's strategy is to maximize the between-group variance σ_B^2, that is, to choose a threshold that maximizes the variance between feature and background pixels. Otsu showed that this maximization also minimizes the combined variance within feature and background classes. Let n_I be the number of pixels with image value I and N the total number of pixels. The probability of image value I is then $P_I = n_I/N$. If we consider a threshold T that divides the image into the pixel sets S_1 (below the threshold) and S_2 (above or equal to the threshold), the cumulative probabilities P_1 and P_2 of the sets S_1 and S_2 are defined as

$$P_1(T) = \sum_{i=0}^{T-1} P_i \qquad P_2(T) = \sum_{i=T}^{I_{\max}} P_i \tag{2.30}$$

The first and second statistical moments are the mean values and within-class variances for the two classes S_1 and S_2:

$$\begin{aligned} \mu_1(T) &= \sum_{i=0}^{T-1} \frac{iP_i}{P_1} \qquad \mu_2(T) = \sum_{i=T}^{I_{\max}} \frac{iP_i}{P_2} \\ \sigma_1^2(T) &= \sum_{i=0}^{T-1} \frac{(i-\mu_1)^2 P_i}{P_1} \qquad \sigma_2^2(T) = \sum_{i=T}^{I_{\max}} \frac{(i-\mu_2)^2 P_i}{P_2} \end{aligned} \tag{2.31}$$

Furthermore, the mean value and variance of the entire image can be obtained through the equation

$$\mu_t = \sum_{i=0}^{I_{\max}} iP_i \qquad \sigma_t^2 = \sum_{i=0}^{I_{\max}} (i-\mu_t)^2 P_i \tag{2.32}$$

With these definitions, the combined within-class variance can be computed as

$$\sigma_W^2(T) = P_1(T)\sigma_1^2(T) + P_2(T)\sigma_2^2(T) \tag{2.33}$$

and the between-class variance σ_B^2 can be defined by

$$\begin{aligned}\sigma_B^2(T) &= P_1(T)[\mu_1(T) - \mu_t]^2 + P_2(T)[\mu_2(T) - \mu_t]^2 \\ &= P_1(T)P_2(T)[\mu_1(T) - \mu_2(T)]^2\end{aligned} \tag{2.34}$$

All values σ_1, σ_2, P_1, and P_2, and consequently, σ_W^2 and σ_B^2, are functions of the threshold T. Otsu proposed a threshold function $\eta(T)$ defined as $\eta(T) = \sigma_B^2/\sigma_t^2$, which can be used as a measure of the goodness of the threshold value T,[17] and the optimum threshold can be found by performing an exhaustive search of all possible T's for the value of T that maximizes $\eta(T)$. Since σ_t^2 is a constant, it is possible to search for a value of T that maximizes σ_B^2 instead. Furthermore, since $\sigma_W^2 + \sigma_B^2 = \sigma_t^2$, Equation (2.34) can be skipped completely and a value of T that minimizes σ_W^2 can be searched instead. All of these variations lead to the same value of T.

Methodically similar is a method to *maximize combined entropy*. Following the definitions in Equation (2.30), the entropies H_1 and H_2 of the background and foreground sets S_1 and S_2 are defined by

$$H_1(T) = -\sum_{i=0}^{T-1} P_i \log_2 P_i \qquad H_2(T) = -\sum_{i=T}^{I_{\max}} P_i \log_2 P_i \tag{2.35}$$

The optimum threshold can be found by searching exhaustively for a threshold T that maximizes the combined entropy $H(T) = H_1(T) + H_2(T)$.[13]

The second question concerns the number of misclassified pixels (Figure 2.9). The application of noise-reduction filters has the potential to narrow the pixel distributions. Here is where Gaussian blurring comes into play, which from a purely visual perspective seems counterintuitive. The application of a median filter or center-weighted median filter to remove extreme values, followed by Gaussian blurring, is a suitable preparation for intensity-based thresholding. The background peak distribution can be narrowed by applying unsharp masking or a homomorphic highpass filter (see Section 3.2) to remove background inhomogeneities. Furthermore, intensity-based thresholding can be improved by making use of connectedness. This gives rise to techniques such as region growing and hysteresis thresholding.

2.4.2. Thresholding with Multiple Thresholds

In cases where a feature with intermediate image values needs to be separated from a darker background and brighter features, the histogram would show three modes. The intermediate mode can be separated with two thresholds T_1 and T_2, where $T_1 < T_2$. The image with the thresholded feature is then converted into a binary mask:

$$I_T(x,y) = \begin{cases} 0 & \text{for } I(x,y) < T_1 \\ 1 & \text{for } T_1 \le I(x,y) \le T_2 \\ 0 & \text{for } I(x,y) > T_2 \end{cases} \tag{2.36}$$

Some methods to find an optimum threshold automatically, most notably Otsu's method,[17] can readily be extended to provide multiple thresholds. Equations (2.30) and (2.31) would be extended to reflect n classes and $n - 1$ thresholds [in Equation (2.36), $n = 3$], and the within-class variance to be minimized becomes

$$\sigma_W^2 = \sum_{i=1}^{n} P_i \sigma_i^2 \tag{2.37}$$

With multiple thresholds, computational efficiency is reduced, as a $(n - 1)$-dimensional space needs to be searched exhaustively for the combination of thresholds that minimizes σ_W^2. Furthermore, the search for multiple thresholds becomes less stable with increasing n and less credible thresholds. For the most common cases, $n = 2$ and $n = 3$, the method remains robust.

2.4.3. Region Growing

Normally, a feature is characterized not only by similar intensities but also by the proximity of the pixels. In Figure 2.7, for example, the lung region is surrounded by tissue with higher intensity values. Outside the patient's body is air with intensity values similar to those of the lung. Unless the connectivity of the lung region is taken into account, intensity-based thresholding cannot separate the lung region from the air outside the patient. Two adjacent feature pixels are considered connected and therefore considered belonging to the same region. Two possible definitions of connectedness exist: 4-connectedness, where only neighbors to the north, west, east, or south are considered to be connected and diagonal pixels are not connected, and 8-connectedness, where all eight neighbors of a pixel, including the diagonal neighbors, are considered to be connected. For connected regions, region growing is a suitable segmentation method that is almost unsupervised. Region growing requires a threshold criterion and one or more seed points. Seed points may be provided interactively or may be special points determined by local criteria (local maxima) or geometry. The four (or eight) neighbor pixels of each seed point are examined, and each neighbor pixel that meets the threshold criterion is added to the feature set and in turn becomes a seed point. This algorithm lends itself to an elegant recursive implementation called *floodfilling*, but for large areas, particularly for a three-dimensional version of the algorithm, the allowable software stack memory size may be exceeded. In this case, it is necessary to scan the entire image iteratively and add neighbor pixels to the feature set until no more pixels are added between iterations. An example implementation of the region-growing algorithm is provided in Algorithm 2.4. This example assumes 8-connectivity and implements a simple threshold condition where all pixel candidates for region growing need to exceed a specified threshold value. This threshold condition can easily be extended to include multiple thresholds or even additional conditions such as a range of the local variance. A demonstration of the effect of region growing is given in Figure 2.10. Both lungs and the air region that surrounds the patient are separated by tissue with higher CT values. Therefore, the lungs and the surrounding air are not connected and can readily be separated by region growing.

```
iteration=0;
repeat
   grown=0;
   for (y=0 while y<ymax-1 increment y=y+1)
      for (x=0 while x<xmax-1 increment x=x+1)
         x1=x; y1=y;
         dx=-1; dy=-1;
         if ((iteration AND 1)==1) then       // odd iteration number
            x1=xmax-x-1; y1=ymax-y-1;
            dx=1; dy=1;
         endif;
         if (IM(x1,y1) > T) then      // threshold condition. Is this pixel a candidate?
            if ( (MASK(x1+dx,y1)>0) // Check connectivity to region
               or (MASK(x1,y1+dy)>0)
               or (MASK(x1+dx,y1+dy)>0) then
                  MASK(x1,y1)=1;      // Add to feature in mask
                  grown=grown+1;      // Count the grown pixel
            endif;
         endif;
      endfor;
   endfor;
until (grown==0);                  // End region growing if no more pixels have been added
```

Algorithm 2.4 Region growing. Two images need to be provided: `IM(x,y)` as the image to be segmented, and `MASK(x,y)`, which initially contains zero-valued pixels with the exception of the seed points, where the pixels have a value of 1. Region growing starts from these seed points for regions with image values above a threshold `T`, and the output is stored in `MASK(x,y)`. Both images have the size `xmax` and `ymax`. Alternating the search direction between iterations accelerates the region-growing process considerably.

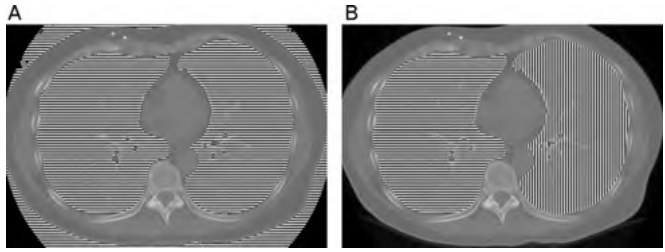


FIGURE 2.10 Demonstration of the difference between intensity-based thresholding (A) and region growing (B). Intensity-based thresholding does not make use of the connectedness of the region, and the air region outside the patient is incorrectly classified as belonging to the feature (white-striped area in A). Conversely, region growing that started with a seed point inside the right lung limits the feature to pixels connected to the seed pixel (horizontally striped region in B). A similar operation with the seed point inside the left lung then region-grows the entire left lung (vertically striped region in B).

2.4.4. Hysteresis Thresholding

Hysteresis thresholding is closely related to region growing. For hysteresis thresholding, two threshold values, T_1 and T_2, need to be defined where $T_1 > T_2$. Each pixel with $I(x,y) \geq T_1$ is selected as a seed point, and from each seed point, region growing is performed with the constraint that $I(x,y) \geq T_2$. In other words, hysteresis thresholding extracts connected regions where all image values are brighter than T_2 but which must contain at least one pixel brighter than T_1. The hysteresis is the difference $T_1 - T_2$. The effect of hysteresis thresholding is demonstrated in Figure 2.11 on a sample image from the Visible Human data set. Figure 2.11A shows the luminance channel of a photograph of the abdominal section where the goal was the segmentation of the adipose tissue of the torso only. Adipose tissue has the highest luminance value, but conventional thresholding ($T = 152$) leaves some adipose tissue from the extremities in the image, and several small pixel clusters within the muscle tissue are also classified as feature pixels (Figure 2.11B). With hysteresis thresholding ($T_1 = 224$ and $T_2 = 152$), only regions connected to the brightest pixels, which lie inside the torso, remain (Figure 2.11C). A similar result can be obtained by region growing with a pixel inside the torso as a seed pixel, but hysteresis thresholding acts in an unsupervised manner. Hysteresis thresholding can be implemented very easily. Algorithm 2.5, in combination with Algorithm 2.4, performs hysteresis thresholding by creating a mask of seed pixels from the higher threshold value T_1. This step is followed immediately by Algorithm 2.4, which uses the variables created by Algorithm 2.5 and performs region growing from all initial seed pixels.

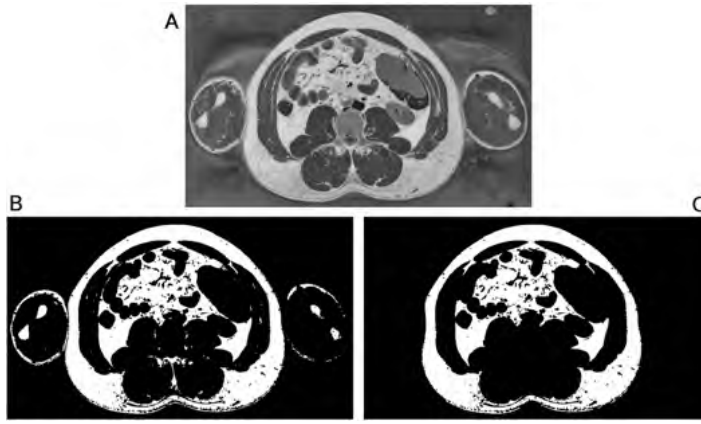


FIGURE 2.11 Comparison of conventional and hysteresis thresholding. Image A shows the luminance channel of a slice from the Visible Human data set. The brightest regions are adipose tissue. Conventional thresholding (B) classifies all adipose tissue, including tissue in the arms and some small clusters in the muscles, as feature pixels, whereas hysteresis thresholding (C) makes use of the connectedness constraint and accepts only adipose tissue from the torso as a feature.

```
allocate MASK[xmax,ymax];          // Reserve space for the mask
T1 = 250;                          // some suitable value for the high threshold
T2 = 128;                          // some suitable value for the low threshold

for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)
      if (IM(x,y) > T1) then
         MASK(x,y)=1;
      else
         MASK(x,y)=0;
      endif;
   endfor;
endfor;

T = T2;                                    // in preparation for Algorithm 2.4
// Append Algorithm 2.4 right here
```

Algorithm 2.5 Hysteresis thresholding. The input image is `IM(x,y)` with size `xmax` and `ymax`. This algorithm prepares the initial seed pixels in `MASK(x,y)` by simple thresholding with `T1`. It is followed immediately by Algorithm 2.4 for the region-growing process where `T = T2`.

2.5. MULTIDIMENSIONAL THRESHOLDING

The idea of thresholding can be extended to encompass multiple criteria. For example, color images contain different information in the various color channels. Gray-scale images contain additional information in the local neighborhood of each pixel: for example, the local standard deviation or the local contrast. For each pixel, multiple criteria can be defined to determine whether they belong to a feature or to the background. Figure 2.12 demonstrates a difficult segmentation situation where multidimensional thresholding may help. A histology section of a rat aorta taken under phase-contrast microscopy is shown in Figure 2.12A and one taken under fluorescent microscopy is shown in Figure 2.12B. Before histology, the lumen of the ex vivo aorta was filled with a green fluorescent dye that stains cell membranes[10]; therefore, the inner part of the aorta wall is more strongly fluorescent than the outer part (Figure 2.12B). Based on pure intensity thresholds, the aorta wall cannot be segmented from the lumen in Figure 2.12A because the intensities are very similar (mean image value in lumen, 85; mean value of wall, 89).

Nonetheless, the aorta wall and the lumen can be separated, because the standard deviations are very different ($\sigma_{\text{lumen}} = 19$ and $\sigma_{\text{wall}} = 38$). A suitable filter would be the local variance or the range operator, followed by strong Gaussian blurring. The local variance operator replaces each pixel by the variance of the pixel values inside an $n \times n$ neighborhood. The range operator replaces a pixel by the difference between maximum and minimum value in an $n \times n$ neighborhood. By using the result of this

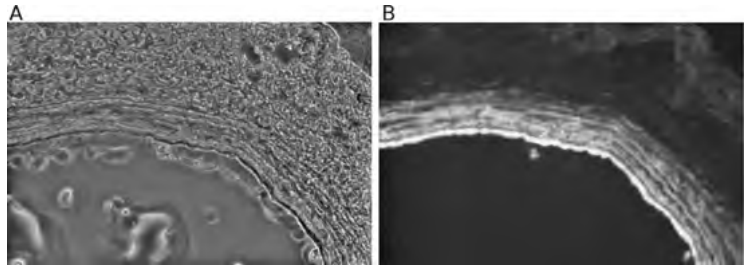


FIGURE 2.12 Example of difficult intensity-based segmentation. A phase-contrast microscopy image of a histology section of the rat aorta is shown (A). Image values between the blood vessel wall and the aorta are very similar, but the standard deviation (irregularity) of the wall, where individual cells become visible, is higher. Furthermore, the inner side of the vessel wall was stained fluorescently and can be identified easily in a fluorescent microscopy image (B).

filter for red colors and the fluorescent image for green, and by using Figure 2.12A for the luminance information, a composite image can be created (Figure 2.13A) where strong green or blue colors indicate the inner wall, red and orange hues indicate the outer wall, and darker areas indicate the lumen. A thresholding strategy can be seen in Figure 2.13B, which shows a two-dimensional histogram: the pixel counts of the filtered version of Figure 2.12A in one dimension combined with Figure 2.12B in the other dimension.

The two-dimensional histogram reveals a very prominent peak where the local variance operator has low values (the homogeneous area of the lumen) and where at the same time there is no fluorescence. Low fluorescence, in combination with high local variance (the cell layer of the outer aorta wall), creates the peaks labeled "ow" in Figure 2.13B. Finally, a comparatively small number of pixels have high fluorescence in Figure 2.12B in combination with some irregularity in Figure 2.12A. This combination creates a third peak ("iw"), which indicates the inner wall, in Figure 2.13B.

Higher-dimensional feature vectors can be devised. In Chapter 8 we describe how local pixel behavior (texture) can be quantified. Often, it is sufficient to separate the pixel classes with simple rectangular or elliptical thresholds (e.g., a square from 0,0 to 60,60 in Figure 2.13B can separate the lumen pixels). However, unsupervised methods exist to assign the points in a multidimensional histogram to classes: k-means clustering and fuzzy c-means clustering. For both clustering methods, it is necessary to know the number of clusters K in advance. In the context of segmentation, K is identical to the number of classes into which the image is to be subdivided. The examples of Figures 2.12 and 2.13 have three classes (bk, iw, and ow); therefore, $K = 3$ in this example. The goal of k-means clustering is to assign each relevant point $P_i = (x_i,y_i)$ (e.g., each nonzero point in the histogram) to one cluster in such a manner that the sum of the squared distances of each point to its assigned cluster centroid $C_c = (x_c,y_c)$ is minimized. In the general case, the points and centroids

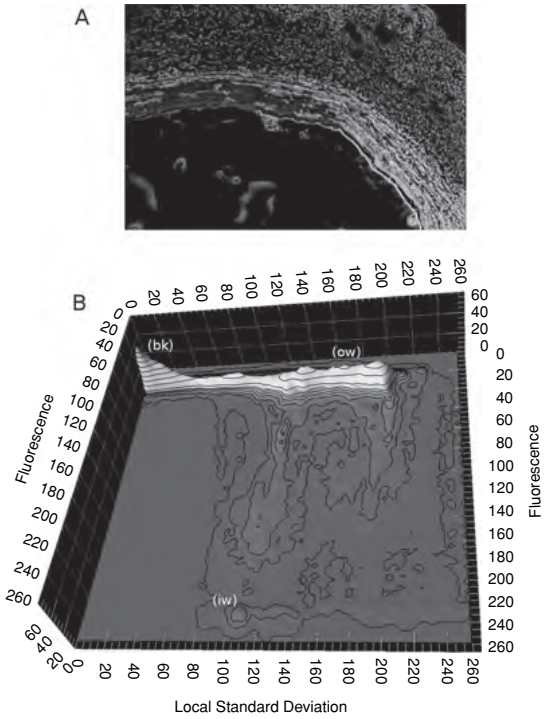


FIGURE 2.13 Multichannel thresholding. The upper image shows a composite created from Figure 2.12A (luminance), Figure 2.12B (green–blue hues), and the local range operator applied to Figure 2.12A (red–orange hues). A two-dimensional histogram of the last two images (B) shows peaks for the lumen (bk), the outer wall (ow) with high local irregularity but low fluorescence, and the inner wall (iw) with intermediate irregularity and high fluorescence. (*See insert for color representation of the figure*.)

are n-dimensional, and the distance metric $D_{i,c}$ between point P_i and centroid C_c is usually the Euclidean distance. K-means clustering requires the following steps:

Step 1. Initialize all centroids C_c for $1 \leq c \leq K$. Often, local maxima or random initial locations are used. Alternatively, centroids may be placed manually.

Step 2. Populate a $N \times K$ assignment matrix $\mathbf{U}$, where N is the number of points to be assigned and $\mathbf{U}(i,c)$ contains the value 1 when point P_i is assigned to cluster C_c, and the value 0 otherwise. The cluster assignment is determined for each point P_i by computing its distance to all cluster centroids C_c and choosing the centroid with the smallest distance. Each point can only be assigned to one cluster.

Step 3. Recompute the centroid locations by using the equation

$$C_c = \frac{\sum_{i=1}^{N} \mathbf{U}(i,c) P_i}{\sum_{i=1}^{N} \mathbf{U}(i,c)} \qquad \text{for } 1 \leq c \leq K \tag{2.38}$$

Step 4. Repeat steps 2 and 3 until the clustering process has converged. Convergence is achieved when either the matrix **U** does not change between two iterations, or the centroids move less than a predetermined distance ϵ.

For a multidimensional histogram, the cluster assignment of all locations in the histogram needs to be translated back into the original image. Each location of P_i corresponds to a vector of values in the various channels of the original image (irregularity and fluorescence in the examples of Figures 2.12 and 2.13). In the simplest form, each pixel of the original image channels would be examined: Its associated feature vector is used to look up the assigned cluster in the matrix **U**, and the cluster number is the new image value of the segmented image.

Fuzzy *c*-means clustering follows a similar principle, but the membership of a point P_i in cluster C_c is continuous; that is, the values in the membership matrix **U** may assume any value between 0 and 1, and each point P_i is to some degree a member of all clusters. The membership function can be very flexible, but a suitable membership function is $D_{i,c}^{-1}$, the reciprocal of the Euclidean distance between point P_i and centroid C_c under the assumption that neighboring pixels have the distance 1. Fuzzy *c*-means clustering follows the same algorithm as *k*-means clustering. The distinction is that the matrix **U** in step 2 is computed through the membership function rather than by determining the shortest distance. The final cluster assignment for P_i takes place after the algorithm has converged by choosing the cluster c for which $\mathbf{U}(i,c)$ has the highest value.

Finally, region splitting and merging deserve to be mentioned as unsupervised segmentation methods. From the pixel feature vector, a similarity measure can be defined. For example, when two feature vectors have a Euclidean distance below ϵ, they are considered similar and the associated pixels are considered to belong to the same class (i.e., feature). Connected pixels of the same class form a region. The following conditions hold true for all regions: (1) regions are composed of connected pixels; (2) pixels may belong to one region only, that is, regions are nonoverlapping; and (3) all regions, joined together, fill the original image. *Region splitting* is a top-down segmentation approach where the image is originally considered to be one region. If pixels within the region fail the similarity criterion, the region is split into four equal-sized rectangular subregions. Each subregion is recursively analyzed and split in the same manner until all pixels within individual regions meet the similarity criterion and splitting is no longer possible. The disadvantage of region splitting is its tendency to produce distinctly rectangular regions, which can be perceived as an unnatural segmentation of the image.

Region merging is the complementary bottom-up approach. Adjacent pixels are joined to form regions if they meet the similarity criterion. Iteratively, the regions formed in such a manner are joined if they meet the similarity criterion until no more regions can be joined. Frequently, region splitting and merging are combined into the split-merge approach, where one iteration of splitting is followed by one iteration of merging until no more splitting or merging is possible. The segmentation results of the split-merge segmentation tend to follow much more the natural object outlines in the image than the region-split segmentation alone.

2.6. IMAGE CALCULATIONS

Image calculations, sometimes termed *image math*, refer to arithmetic or logic operations on a pixel-by-pixel basis. Image values can either be manipulated with the same constant value for all pixels or with the values of corresponding pixels in a second image. Any mathematical expression is possible, and some operations are very useful in practical applications. One example of addition and subtraction are CT images. CT image values can typically range from −1000 HU (e.g., air) to +3000 HU (e.g., contrast agents, metal objects). To improve compatibility, negative numbers are often avoided by adding an offset of 1024 (2^{10}) to the image values. Therefore, the image values of the stored image now range from +24 to +4095 ($2^{12} - 1$), and the image values can be stored in an unsigned 12-bit integer field. Compatibility with image processing software that can only handle 8-bit values can be achieved by dropping the least significant 4 bits of the 12-bit integer value. This operation corresponds to a division by $2^4 = 16$ and the resulting value range is now 0 to 255. Other image analysis software that can handle higher bit depths would read the CT image and subtract the offset of 1024 to restore the original values in Hounsfield units. This type of pixel manipulation can be described in its most general form by

$$I'(x,y) = f(I(x,y), C) \tag{2.39}$$

where f describes the operation (i.e., addition, subtraction, multiplication, etc.) and C is a constant value. The function f is performed independently on all pixels of the image. Without explicitly defining the operation as image math, Equation (2.5) (histogram stretching) is a linear pixel-by-pixel operation that can serve as an example for the model in Equation (2.39). The function f may also be nonlinear. For example, gamma correction can be applied to enhance contrast in dark image regions. Frequently, output devices (e.g., video screens; even more so, printers) tend to reduce contrast in dark regions. Light output (or, in the case of printed paper, light reflectance) L can be modeled as the normalized image value raised to the power of a constant γ such that $L(x,y) = I(x,y)^{\gamma}$, where $I(x,y)$ is the image value normalized to the range 0 to 1. γ typically lies between 1 and 2. The suitable correction is to raise the value of each pixel to the power of $1/\gamma$ before displaying or printing the image. The effect of this operation is shown in Figure 2.14. An even stronger contrast adjustment is to compute the logarithm of each pixel value and to rescale the results to fit the display range (usually, 0 to 255). A logarithmic display function is often chosen to visualize extreme image value ranges such as those obtained through the Fourier transform. In this case it must be ensured that image values of zero or negative values do not occur. To display the magnitude of the Fourier transform (no negative values), an offset of 1 can be added to the image and the function f in Equation (2.39) becomes $\log(I + 1)$. In cases where negative values exist, a suitable offset can be added, or negative values are clamped to 1 through thresholding.

The general equation to perform an operation on two images, I_1 and I_2, is

$$I'(x,y) = f(I_1(x,y), I_2(x,y)) \tag{2.40}$$

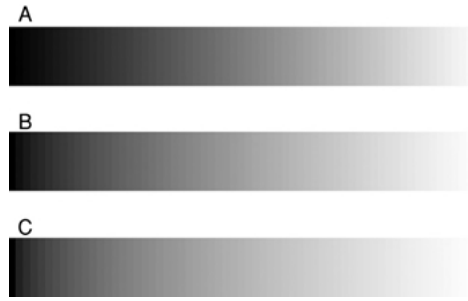


FIGURE 2.14 Gamma correction. A linear gradient with image values from 0 to 255 in steps of 4 (A) after contrast correction with $\gamma = 1.5$ (B) and $\gamma = 2.0$ (C).

Most commonly, the function f represents addition, subtraction, multiplication, or division. It is also assumed that I_1 and I_2 have the same size, although zero padding or tiling is possible in special cases. In addition, the same operation can be performed on an image stack for all stacked slices z:

$$I'(x,y,z) = f(I_1(x,y,z), I_2(x,y)) \tag{2.41}$$

Each of the four operations has typical applications in biomedical imaging. Image addition could be used for noise reduction. For example, N identical images can be acquired with a modality that has a strong noise component (such as magnetic resonance imaging or optical coherence tomography). By adding those N images and dividing the resulting value by N, the pixel-by-pixel average is computed, and additive Gaussian noise is reduced by a factor of $\sqrt{N}$. An example is given in Figure 2.15. Contrast is approximately 28 (in arbitrary OCT units), and the standard deviation is 5.6 (Figure 2.15A) and 1.46 (Figure 2.15B). The ratio of these two values can be used as a simplified metric of the signal/noise ratio (SNR) and results in SNR values of 5 and 20, respectively. This is an improvement in the SNR by a factor of 4, consistent with 20 averaged slices and $\sqrt{20} = 4.5$. Note that further noise reduction is possible by removing shot noise (median filter) before averaging the slices. The combined operation (median filtering of each individual slice followed by averaging) yields an SNR of 31.

Image subtraction and division are often used to remove an inhomogeneous background. Several examples are provided in Section 5.3. Subtraction needs to be used when the background is caused by an inhomogeneous bias (e.g., a bias field in magnetic resonance images), and division is a suitable operation for multiplicative inhomogeneities, such as inhomogeneous illumination. Consider, for example, visible-light illumination $L(x,y)$ that is brighter in the center than toward the edges of the image. A photograph is taken from a reflective object with a reflectance $R(x,y)$. The object is described by the reflectance, but the photograph contains the multiplicative image $I(x,y) = R(x,y)L(x,y)$. If $L(x,y)$ is known (e.g., by photographing a homogeneously gray surface under the same illumination), $R(x,y)$ can be recovered from $I(x,y)$ by dividing $I(x,y)$ by $L(x,y)$. Care must be taken when the image is divided by a second image with a large range of image values. First, a division-by-zero value

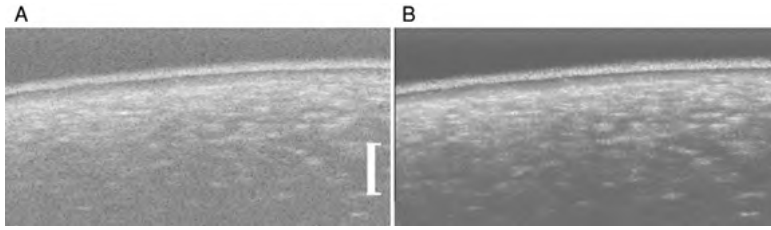


FIGURE 2.15 Example of image addition. Image A is an optical coherence tomography scan of an apple to a depth of approximately 0.5 mm. The white scale bar indicates 200 μm. The thin white layer is the apple's skin. Underneath is the actual plant tissue with a dark intermediate layer representing the transition of the skin into the tissue. The OCT slice exhibits considerable noise. Both Gaussian noise and shot noise (black pixels) exist. Image B was created by acquiring 20 successive slices of the same region and adding them. After addition, the values were rescaled to fit the range 0 to 255. Noise is markedly reduced and details become better visible. Further noise reduction could be achieved by applying a median filter operation on each individual slice before addition. (Courtesy of Jinjun Xia.)

needs to be avoided. This can be achieved by adding an offset to the denominator image. Similar considerations apply when the image in the denominator contains small values. One example for the division of two images is deconvolution in the frequency domain (see Section 3.2.3). Image subtraction is a popular method used to remove an inhomogeneous background when the illumination (or background) function is unknown. A semianalytical method for background removal is to model the background image $B(x,y)$ as a biparabolic plane or the section of a sphere, respectively:

$$B(x,y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 y^2 + a_5 xy \tag{2.42}$$

$$B(x,y) = \sqrt{a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 y^2} + a_5 \tag{2.43}$$

The unknown parameters a_0 through a_5 can be found by fitting the background function to the image. For this purpose, the image is divided into equal square tiles. When the features protrude as brighter objects from a dark background and the tile size is larger than the feature size, an assumption can be made that the darkest pixel in each tile is a background pixel. In each tile i, the darkest pixel $B_i\,(x_i,y_i)$ can be found, and the unknown parameters a_0 through a_5 are determined by nonlinear least-squares fitting. Finally, the background image $B(x,y)$ is subtracted from the image pixel by pixel. Alternatively, the background function can be found by extremely strong blurring of the image (e.g., by convolution with a large Gaussian kernel or by application of a Fourier lowpass filter). This background function can then be subtracted directly from the image. This operation was introduced in Section 2.3 as unsharp masking.

Subtraction and division are used when the normalized difference between two images I_1 and I_2 needs to be computed. The normalized difference image with pixel values in percent is given by

$$D(x,y) = 100 \cdot \frac{I_1(x,y) - I_2(x,y)}{I_1(x,y)} \tag{2.44}$$

Multiplication of images is a useful tool in masking. The thresholding operation as defined, for example, in Equation (2.29) provides a binary mask where feature pixels are represented by the value 1 and background pixels by the value 0. When the original image is multiplied with the mask pixel by pixel, all background pixels are assigned the value 0, while the feature pixels retain their original value. This combined operation, known as *hard thresholding*, can be described as

$$I_T(x,y) = \begin{cases} I(x,y) & \text{for } I(x,y) \geq T \\ 0 & \text{for } I(x,y) < T \end{cases} \tag{2.45}$$

The additional subtraction of the threshold value T (to be precise, the pixel-by-pixel subtraction of the mask multiplied by T) leads to an equation for *soft thresholding*:

$$I_T(x,y) = \begin{cases} I(x,y) - T & \text{for } I(x,y) > T \\ 0 & \text{for } I(x,y) \leq T \end{cases} \tag{2.46}$$

In both hard and soft thresholded images, further analysis can take place while zero-valued pixels are ignored. If image measurements (e.g., pixel count, image value average, histogram) are limited to nonzero values, the measurements reflect the masked features only. Another example where image multiplication plays a key role is image filtering in the frequency domain (Section 3.2). A filter function is the equivalent of an image in frequency space, described by its transfer function $H(u,v)$. The coordinates u and v are spatial frequencies and are treated in the same manner as the spatial coordinates x and y. The main difference is the widely used convention that the origin of the (x,y) coordinate system coincides with the top left corner of an image, while the origin of the (u,v) coordinate system coincides with the center of the image. The Fourier transform of an image has low-frequency components (intensity gradients that continue over the entire image) near the origin, and high-frequency components (e.g., abrupt changes, texture, noise) toward higher values of $|u|$ and $|v|$ closer to the edge of the frequency-domain image. A highpass filter is a filter that attenuates values near the center of the (u,v) coordinate system, with values of $H(u,v) < 1$ for small $|u|$ and $|v|$ and values of $H(u,v) \approx 1$ for large $|u|$ and $|v|$. Conversely, a lowpass filter has values of $H(u,v) \approx 1$ for small $|u|$ and $|v|$ and values of $H(u,v) < 1$ for large $|u|$ and $|v|$. The filter consists of three steps: computation of the Fourier transform, pixel-by-pixel multiplication of the transformed image $F(u,v)$ with the filter $H(u,v)$, and the inverse Fourier transform of the result.

For all image math operations, the image value range needs to be observed. Image math operations may lead to negative values, to values that are larger than the range of the original image (e.g., multiplication or addition of many images), or to very small values [e.g., division, most prominently in Equation (2.44)]. In image processing software that holds a limited value range (e.g., 8-bit images), the range may be exceeded or rounding errors may cause loss of detail. In many cases the image values are either rescaled after the operation to fit the full range or are clamped (e.g., pixels that exceed 255 in an 8-bit image are kept at 255). If this behavior is not acceptable, the result of the operation needs to be stored in a different format, for example, as a floating-point value for each pixel.

2.7. BINARY IMAGE PROCESSING

The segmentation results discussed in this chapter generally lead to binary images, considered to be masks for the segmented features. The information contained in the image values is lost, but the spatial relationship between the mask and the features is retained. Both mask and masked image can be subjected to an analysis of texture (Chapter 8) or shape (Chapter 9). Some additional processing of the segmented images is possible to further improve the segmentation results. These steps could be referred to as *postsegmentation processing* and include binary noise reduction and shape modification.

2.7.1. Morphological Image Processing

One important group of operators are called *morphological operators*. The concept of morphology is a comprehensive subject,[23] and for our purposes in this chapter, basic morphological operators are introduced more in the spirit of providing the necessary tools in a versatile toolbox than in the sense of introducing the concepts of morphology. The two basic operators are the erosion (thinning) and dilation (thickening) of shapes. A general form of the erosion and dilation operators, defined as vector additions of elements from the image and a pixel mask (called the *structuring element*) exist.[12] However, the most common implementation of the erosion and dilation is to replace the central pixel by the minimum or maximum of its neighborhood, respectively. The neighborhood may either be the 4-neighborhood, which includes the horizontal and vertical neighbors, or the 8-neighborhood, which also includes the four diagonal neighbors. Under this definition, erosion and dilation are rank filters related to the median filter that was discussed in Section 2.2 and are defined for gray-scale images. However, erosion and dilation applied to gray-scale images are not intuitive and are rarely used. In binary images, the definition of erosion and dilation above reduces to two simple rules: If a white (foreground) pixel touches a black (background) neighbor, it turns itself into a background pixel under the *erosion* operation. Conversely, if a black (background) pixel touches a white (foreground) neighbor, it becomes a foreground pixel under the *dilation* operation. Each application of the erosion operation therefore removes a single-pixel-wide strip from the boundary of any feature and thins the feature in the process. Furthermore, individual pixels and small pixel groups (e.g., noise) are removed from the image completely. Conversely, each application of the dilation operation adds one single-pixel-wide strip to the boundary of the feature, thickening the feature in the process. Small holes inside the feature and along the boundary get filled in the process. Erosion and dilation are complementary; that is, an erosion acting on white pixels is the same as a dilation acting on the black pixels, and vice versa.

When operating on binary images, the median filter can be seen as a majority filter. If the majority of the neighborhood pixels are white, the central pixel itself becomes white, and if the majority of the neighborhood pixels are black, the central pixel turns black. Therefore, the median filter acts as a noise filter on binary images, where noise removal not only includes the removal of isolated white pixels and the filling

of isolated black pixels (holes) in a feature, but also the smoothing of the boundary. Contrary to erosion and dilation, the rank filter leaves the boundary shape widely unchanged.

A possible implementation of the erosion and dilation operators is shown in Algorithm 2.6. Because of their similarity, both are implemented in the same operator, and a flag determines which operation is performed. Algorithm 2.6 contains the general form for gray-value images, but can be used without modification for binary masks. Algorithm 2.6 is designed to consider an 8-neighborhood.

Derived from erosion and dilation are the opening and closing operators. *Opening* is a single erosion followed by a single dilation, and *closing* is a single dilation followed by a single erosion. The opening operation removes individual white pixels or small pixel groups, and it separates weakly connected structures, for example, two parts of a feature that are connected by only a single pixel. The closing operation

```
allocate IM2[xmax,ymax];                    // Reserve space for the output image
erode=1;                                    // Set erode=1 for erosion or erode=0 for dilation

for (y=0 while y<ymax increment y=y+1)
   for (x=0 while x<xmax increment x=x+1)

      // Start inner loop to examine pixel neighborhood and determine min and max values

      min = IM(x,y); max=min;
      for (y1=y-1 while y1<=y+1 increment y1=y1+1)
         for (x1=x-1 while x1<=x+1 increment x1=x1+1)
            if ((x1>=0) and (y1>=0) and (x1<xmax) and (y1<ymax)) then
               if (min>IM(x1,y1)) then min=IM(x1,y1); endif;
               if (max<IM(x1,y1)) then max=IM(x1,y1); endif;
            endif;
         endfor;
      endfor;

      // Here, we know the minimum and maximum of the neighborhood
      // Use them according to the erode flag

      if (erode==0) then
         IM2(x,y) = max;                    // Dilate
      else
         IM2(x,y) = min;                    // Erode
      endif;
   endfor;
endfor;
```

Algorithm 2.6 Erosion and dilation operators. The input image is `IM` with size `xmax` and `ymax`. The algorithm stores the result in `IM2`. Depending on the value of the variable `erode`, this algorithm can perform either morphological erosion or morphological dilation.

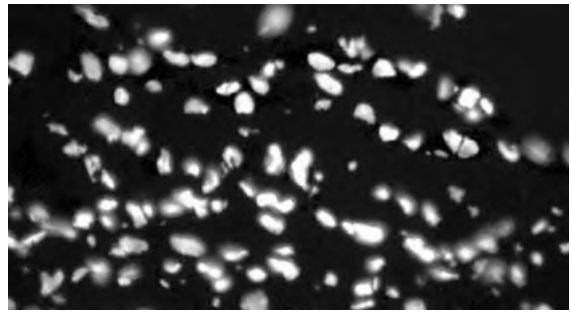


FIGURE 2.16 Cell nuclei made visible with the fluorescent DNA marker DAPI in an epifluorescent image.

fills small interior holes and strengthens weak connections. Both operations smooth the outlines and thus simplify the shapes, whereby the size of the shape is changed only minimally. An example is given in Figures 2.16 and 2.17. Figure 2.16 shows an epifluorescent image of cell nuclei stained with DAPI, a fluorescent marker that specifically targets DNA. The image was thresholded (Figure 2.17A) and several morphological operators were applied (Figure 2.17B and C). In all cases, two iterations of the operator were applied to make the effect more visible. Whereas the effect of the erosion and dilation is plainly visible, opening and closing have a more subtle effect. The smoothing of the shape boundary can be seen clearly.

Outlining is one possible application of this class of operators, in combination with image math. The outline is created by subtracting the original image from its dilated version. A thicker and more symmetric outline with respect to the shape boundary can be obtained by subtracting the eroded version of the image from the dilated version. In this case, the outline is 2 pixels wide. Figure 2.18 shows an example of outlining where the mask (Figure 2.17A) was dilated twice and subtracted from the undilated mask. The outlines were then superimposed over the original gray-scale image. The image outlined shows clearly which cells were included in the hysteresis threshold scheme. Outlining can therefore serve as a visualization tool. Labeling of gray-scale images with text is another example where an operation similar to outlining is helpful. Images with strong local variations of the gray values make superimposed labels difficult to read. To improve readability, the letters (assumed to be white) can be dilated several times. The original letters are then subtracted from the dilated letters, creating a black label over a white background that surrounds the letters. This combination, superimposed over the image, shows high contrast and is easy to read.

In the context of processing binary masks, two more operators are important: skeletonization and watershed segmentation. *Skeletonization* refers to iterative erosion with two additional constraints: A pixel may not be deleted in the erosion process if it is an endpoint or if its removal would break a feature in two.[26] Skeletonization acts best on long, thin objects and extracts their medial axis, which can be thought of as the central ridge. Skeletonization is a fundamental operation for shape classification and is discussed in detail in Section 9.6.

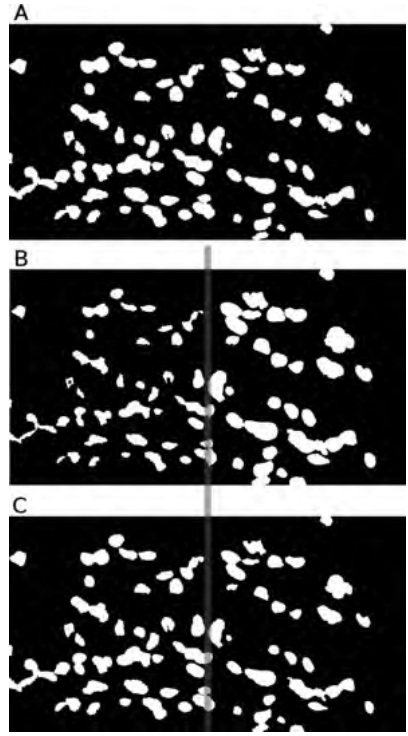


FIGURE 2.17 Examples of morphological operators. Hysteresis thresholding of Figure 2.16 leads to the binary mask (A). The individual operations are thinning by erosion (B, left of the dividing line), thickening by dilation (B, right of the line), opening (C, left of the line), and closing (C, right of the line).

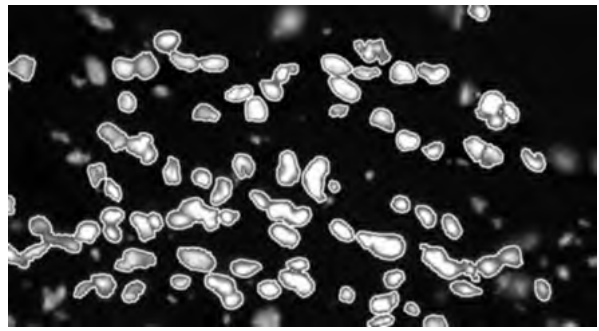


FIGURE 2.18 Outlining by computing the difference between the original mask and the dilated mask. In this example, the outline is superimposed over the unthresholded gray-scale image.

2.7.2. Watershed Segmentation

Watershed segmentation (often incorrectly termed the watershed transform—it is not a transform because no inverse transform to restore the original image exists) is a method to separate in a binary mask two convex shapes that partly overlap. The cells in Figure 2.16 contain typical examples. It can be seen that in the top left region of Figure 2.17A that a group of two cells and another group of three cells have not been separated by thresholding. Postprocessing of the binary mask with the watershed operator can separate those connected features. The watershed operator relies on iterative erosion and iterative dilation, both with certain constraints. If the mask is iteratively eroded, starting with the ring of pixels closest to the background, it is possible to label these rings with an image value identical to the erosion iteration number and to construct an auxiliary image from the eroded points labeled in such a manner. This auxiliary image is a gray-scale image, and the image values get higher with increasing distance of the pixels from the background. In fact, the image value is approximately the Euclidean distance of the pixel to the nearest background pixel and is therefore called the *Euclidean distance map*. Continuing with the iterative erosions, shapes like the cells mentioned above eventually get separated. The erosion process needs to be constrained such that isolated pixels (i.e., pixels surrounded by eight background neighbors in each and any erosion iteration) may not be eroded. These pixels coincide with local maxima in the Euclidean distance map and are called *ultimate eroded points*, because further constrained erosions do not change the image of ultimate eroded points. The iterative erosion process ends when only ultimate eroded points are left over. For convenience, the ultimate eroded points can now be numbered consecutively. The ultimate eroded points are now iteratively dilated. Each dilation process, according to its definition, adds a ring of pixels with the same image value as that of the originating ultimate eroded point. Dilations are constrained as well, with two constraints: First, no pixel that was a background pixel in the original mask may be added to the set (i.e., dilation is constrained to the original mask), and second, no pixel may be added to the set if it connects a feature to another feature with a different image value. In this way, merging of two features that grow from different ultimate eroded points is prevented, which is the key purpose of watershed segmentation. The relationship between the Euclidean distance map, ultimate eroded points, and watershed-segmented image is shown in Figure 2.19. Figure 2.19A is a surface rendering of the Euclidean distance map where the elevation of each pixel corresponds to its distance to the background. The peaks of the hills (i.e., points with the greatest distance to the background) coincide with the ultimate eroded points, each of which is the seed point for iterative dilation. The constrained iterative dilation leads eventually to Figure 2.19B, where each ultimate eroded point is now contained in a separate feature that is separated (black lines) from features grown from neighboring ultimate eroded points.

Watershed segmentation can be influenced by the definition of a local maximum in the Euclidean distance map. To improve the segmentation, we need to employ an alternative algorithm to generate the ultimate eroded points, which is not based on iterative erosion. Rather, the Euclidean distance of each feature pixel to the nearest

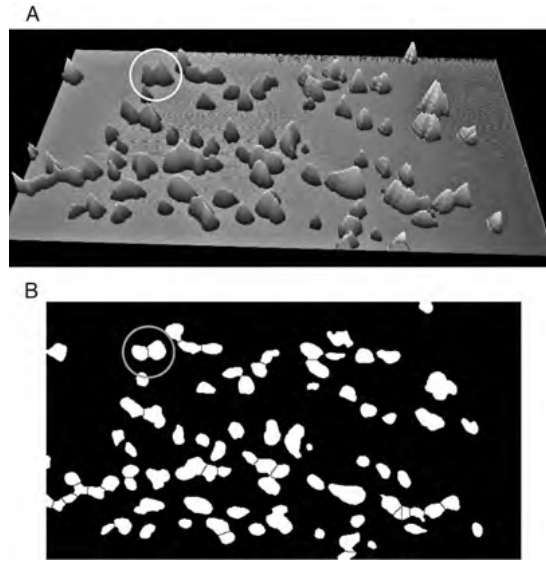


FIGURE 2.19 Watershed segmentation of the binary mask in Figure 2.17C. The first step is the creation of the Euclidean distance map (A), shown here as a three-dimensional surface rendering. Higher elevations indicate a larger distance to background, and the slopes of the hills are 45°. The peaks of the hills coincide with the ultimate eroded points. A typical example is highlighted (circle). It appears as a single feature in the mask, but the original image (Figure 2.16) clearly shows this to be two different cells. Correspondingly, the Euclidean distance map exhibits two local peaks. Iterative dilation from the ultimate eroded points with the constraint that no two different features may become connected leads to the segmented image (B), where thin black (background) lines separate regions that contain its individual ultimate eroded point.

background pixel is determined by an exhaustive search. The Euclidean distance map generated in this fashion can now be subjected to Gaussian smoothing to suppress small local maxima (i.e., noise). Further a priori knowledge may be included by defining the area in which a maximum needs to be unique to be recognized as a local maximum. With these two options, the final size of the segmented features, and therefore the sensitivity of the watershed segmentation, can be influenced.

2.8. BIOMEDICAL EXAMPLES

In this chapter, basic tools for a complete image analysis chain are introduced. These include, in order, image preprocessing (image enhancement or restoration, usually by filtering), image segmentation, some postsegmentation enhancement, and finally, quantitative analysis. One very typical example was presented by de Reuille et al.,[5] who acquired confocal sections of the shoot apical meristem in *Arabidopsis thaliana*. The meristem is plant tissue consisting of undifferentiated cells and is found in areas of

the plant where growth takes place. Image contrast was gained by fluorescent labeling of the cell membranes. Confocal sections form *z-stacks*, two-dimensional images of thin sections that can be stacked to form a three-dimensional volume image. In this study, preprocessing consisted of Gaussian blurring to smooth images, in particular to reduce the impact of isolated bright or dark dots (it should be noted that isolated bright or dark dots can better be removed by median filtering). Subsequently, a threshold was applied to remove all the pixels with an image value of less than 10. This step was followed by linear histogram stretching. The segmentation step posed a challenge because the tissue area contained dark regions that were similar to the background outside the tissue region. To overcome this challenge, the segmentation process involved a combination of intensity-based thresholding followed by morphological closing (to fill the smaller dark regions inside the tissue region), which was then followed by watershed segmentation. With the fluorescently labeled cell walls now segmented, further analysis of cell geometry and topology (see Chapter 9) was possible, and plant development over time was quantitatively analyzed.

Related methods were used for image preprocessing and segmentation of electron microscopy images of collagen fibrils in the presence or absence of gold nanoparticles.[9] The preprocessing steps were four iterations of the median filter followed by unsharp masking to remove background inhomogeneities. A binary mask was created by applying a threshold that was determined in an unsupervised manner by using Otsu's method. Postprocessing of the mask was required to remove the nanoparticles, which was performed by identifying and removing connected regions with fewer than 150 pixels. Finally, morphological closing was applied to remove small interior holes or fissures. The steps that led to the final mask are shown in Figure 2.20. The resulting mask was now suitable for quantitative analysis using shape analysis methods (Chapter 9): namely, run-length analysis and topological analysis.

An example of an area in which image math becomes important is in the processing of microscope images to determine fluorescence anisotropy. In one example,[8] fluorescent microscope images were used to map calmodulin binding over the projected cell area. A fluorescent calmodulin analog was imaged, and polarized microscope images were taken. Fluorescence anisotropy is based on the principle that some fluorescent molecules can be excited only in a specific polarization plane, and the fluorescent emission occurs in the same plane if the molecule is immobile. However, if the molecule can rotate during the excited lifetime, the fluorescence emission is randomized, and the degree of randomization is based on the rotational freedom. The degree of randomization can be determined from two images: one polarized parallel to the excitation polarization plane and one polarized perpendicular to it. Let those two images be designated $I_{\parallel}$ and $I_{\perp}$, respectively. When the rotational freedom is low, most emission is still parallel to the original excitation plane, and $I_{\parallel}$ is high. Conversely, when the polarization is randomized with high rotational freedom, $I_{\parallel}$ and $I_{\perp}$ are about equally high. Polarization anisotropy can be defined through[16]

$$R(x,y) = \frac{I_{\parallel}(x,y)/I_{\perp}(x,y) - 1}{I_{\parallel}(x,y)/I_{\perp}(x,y) + 2} \tag{2.47}$$

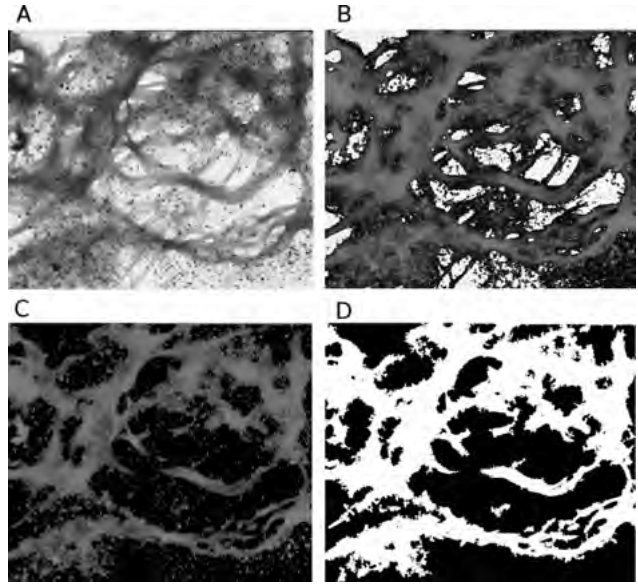


FIGURE 2.20 Preprocessing and segmentation of collagen fibrils in the presence of gold nanoparticles. (A) is the original electron-microscope image. The fibrils are the long gray features, and the nanoparticles are the small black dots. After denoising (median filter) and background removal, (B) is obtained. The application of a hard threshold [Equation (2.45)] with Otsu's method leads to image (C). A final mask (D) is created by removing all features that are smaller than 150 pixels. (From ref. 9.)

where $R(x,y)$ is the spatially resolved anisotropy. When the rotational mobility is low, the ratio of $I_{\parallel}$ to $I_{\perp}$ assumes a very large value and R approaches unity. Conversely, when rotational mobility is high, the ratio $I_{\parallel}$ over $I_{\perp}$ is close to unity, and R is close to zero. When the fluorescent calmodulin interacts with myosin II light chain kinase,[8] its rotational mobility is reduced and R is elevated. By computing R on a pixel-by-pixel basis using Equation (2.47) from microscope images of the cell, the locations of elevated calmodulin binding can be mapped.

To measure the water content (more precisely, the polarity of the environment) of a cell membrane, the fluorescent probe Laurdan is used. In a polar environment, the fluorescent emission of Laurdan experiences a red shift. Two fluorescent images can be taken, I_B at the maximum emission of Laurdan in a nonpolar environment (440 nm) and I_R at the maximum emission in a polar environment (490 nm). A ratiometric value of polarity, known as general polarization GP,[18] can be computed by

$$\mathrm{GP}(x,y) = \frac{I_B(x,y) - I_R(x,y)}{I_B(x,y) + I_R(x,y)} \tag{2.48}$$

Recently, the GP value was applied by Zhu et al.[27] to examine the influence of reactive oxygen species on the cell membrane, a process hypothesized to be linked to

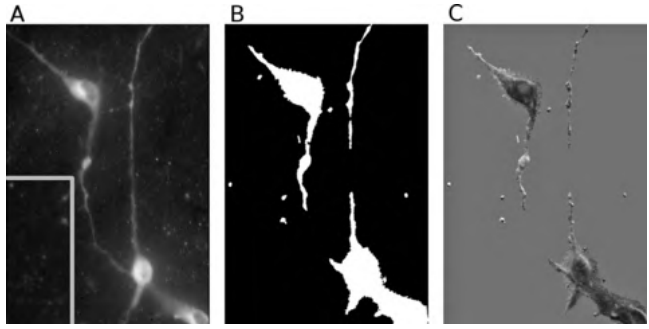


FIGURE 2.21 Steps to compute the general polarization of Laurdan. One of the two unprocessed images (I_R) is shown (A); after application of the smoothing operation, noise is reduced (inset in A, delimited by gray lines). After denoising and addition of I_R and I_B, a cell area mask (B) can be generated by thresholding with Otsu's method. The final GP value [Equation (2.48)] is computed only inside the cell area, while the background is kept at zero. Since the GP image contains negative values, histogram stretching to the printable value range of 0 to 255 causes the zero-valued background to be gray, with negative values being darker and positive values being lighter (C). (From ref. 27.)

cell damage and aging. To compute the GP map over the cell area, the two fluorescent images I_B and I_R were obtained. Equation (2.48) contains a division, which is a noise-sensitive operation. For this reason, both images first underwent Gaussian blurring; then the denominator was computed. Otsu's threshold method was used to create a mask of the cell area. The final GP value was computed only within regions identified as cell regions, and the background was kept at zero. The individual steps are shown in Figure 2.21. The final map underwent histogram stretching to be printable, and negative values (dominant I_R), which indicate regions of low polarity, are printed in dark shades and positive values (dominant I_B, high polarity) are printed in lighter shades. The main advantage of ratiometric methods is the normalization. The GP value, for example, is independent of the overall exposure. For this reason, ratiometric methods are particularly well suited for quantitative image analysis.

The lung was used repeatedly as an example in this chapter. In fact, an image analysis chain similar to the examples used in this chapter was presented by Chen et al.[4] CT slices of the lung were first filtered for noise removal, followed by a flood-filling process of the tissues surrounding the lung. Flood filling is functionally similar to region growing. The resulting mask was further enhanced by erosion. Results were similar to those shown in Figure 2.10.

Another example where the filtering techniques described in this chapter play a key role is in the analysis of angiograms of the retina.[25] The purpose is to detect aneurysms in the blood vessels of the retina, and contrast enhancement was performed with fluorescein. Fluorescein leads to a bright fluorescence of the blood vessels over the darker background of the surrounding tissue. Illumination inhomogeneities were first removed by dividing by a "flood image", that is, an image that reflects the illumination brightness. This operation was followed by unsharp masking to enhance

the contrast of thin blood vessels. Bright pixels were then extracted using a top-hat filter,[1] a morphological segmentation method that compares the brightest values in two concentric neighborhoods. If the brightest pixel in the smaller neighborhood is brighter by a threshold T than the brightest pixel in the larger neighborhood, the pixel is retained; otherwise, it is set to zero. In this formulation, the top-hat filter isolates local maxima. Those were eroded further to retain ultimate eroded points, which were used as seed points in the original fluorescent image for a region-growing process. In the final result, only small isolated and circular features, the suspected microaneurysms, were left in the image for visual inspection.

As a final example, a method is summarized to extract the steps of an aluminum step wedge from an x-ray image for the purpose of densitometry calibration.[11] The purpose of using the step wedge was to eliminate fluctuations of film exposure and aging of film developer in quantitative x-ray images of mouse bones. A logarithmic curve fit of the film density over the aluminum thickness yielded the apparent x-ray attenuation coefficient for aluminum, which can then be compared with known attenuation coefficients. In addition, any density could be calibrated in equivalent units of aluminum thickness. A key element for the segmentation of the wedge steps was edge detection, because the edges from one step to the next could be used to delimit the flat areas of the step. Since the scanned x-ray image was very noisy, the edge detector (in this case, the Sobel operator) was preceded by a median filter and followed by a one-dimensional Gaussian smoothing filter that blurred the image in the horizontal direction only. The intensity average, computed line by line, made it possible to reliably identify a sufficient number of edges (Figure 2.22). The area between the edges was averaged and used to represent density under the corresponding step.

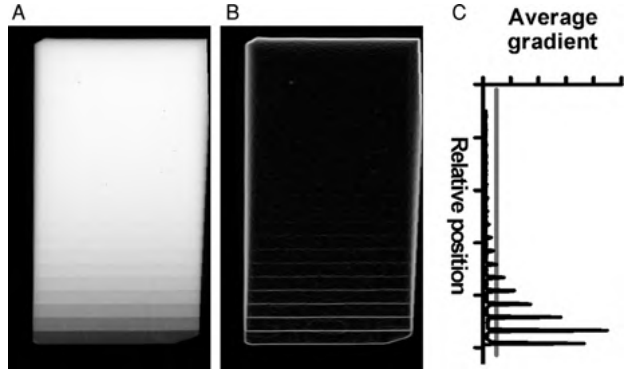


FIGURE 2.22 Segmentation of the steps of an aluminum step wedge in an x-ray projection image (A). The segmentation process makes use of the horizontal orientation of the steps. A median filter was used before the application of the Sobel operator for noise reduction, and the edge image was further filtered by convolving with a one-dimensional smoothing filter (B). Averaging of the intensity values line by line causes strong maxima at the edge locations (C), which can be used to detect edges. In this case, any maximum that exceeded the gray line (C) was recognized as an edge, and the algorithm allowed reliably to detect seven or eight steps. [(A) From ref. 11.]

REFERENCES

1. Bright DS, Steel EB. Two-dimensional top hat filter for extracting spots and spheres from digital images. J Microsc 1987; 146:191–200.
2. Canny JF. A computational approach to edge detection. IEEE Trans Pattern Anal Mach Intell 1986; 8:679–698.
3. Canny JF. Finding Edges and Lines in Images. Technical Report AI-TR-720. Cambridge, MA: Artificial Intelligence Laboratory, 1983.
4. Chen Z, Sun X, Nie S. An efficient method of automatic pulmonary parenchyma segmentation in CT images. Conf Proc IEEE Eng Med Biol Soc 2007; 2007:5540–5542.
5. de Reuille PB, Bohn-Courseau I, Godin C, Traas J. A protocol to analyse cellular dynamics during plant development. Plant J 2005; 44(6):1045–1053.
6. Deriche R. Using Canny's criteria to derive a recursively implemented optimal edge detector. Int J Comput Vis 1987; 1:167–187.
7. Frei W, Chen C-C. Fast boundary detection: a generalization and a new algorithm. IEEE Trans Comput 1977; 26:988–998.
8. Gough AH, Taylor DL. Fluorescence anisotropy imaging microscopy maps calmodulin binding during cellular contraction and locomotion. J Cell Biol 1993; 121(5):1095–1107.
9. Haidekker MA, Boettcher LW, Suter JD, Rone R, Grant SA. Influence of gold nanoparticles on collagen fibril morphology quantified using transmission electron microscopy and image analysis. BMC Med Imaging 2006; 6:4.
10. Haidekker MA, Ling T, Anglo M, Stevens HY, Frangos JA, Theodorakis EA. New fluorescent probes for the measurement of cell membrane viscosity. Chem Biol 2001; 8(2):123–131.
11. Haidekker MA, Stevens HY, Frangos JA. Computerized methods for x-ray-based small bone densitometry. Comput Methods Prog Biomed 2004; 73(1):35–42.
12. Haralick RM, Shapiro LG. Computer and Robot Vision, Vol. I. Reading, MA: Addison-Wesley, 1992.
13. Kapur JN, Sahoo PK, Wong AKC. A new method for gray-level picture thresholding using the entropy of the histogram. Comput Vis, Graph, Image Process 1985; 29:273–285.
14. Kirsch R. Computer determination of the constituent structure of biological images. Comput Biomed Res 1971; 4:315–328.
15. Ko S-J, Lee YH. Center weighted median filters and their applications to image enhancement. IEEE Trans Circuits Syst 1991; 38(9):984–993.
16. Lakowicz JR. Introduction to Fluorescence: Principles of Fluorescence Spectroscopy. 3rd edition. New York: Springer Verlag, 2006; 1–26.
17. Otsu N. A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 1979; 9(1):62–66.
18. Parasassi T, Di Stefano M, Loiero M, Ravagnan G, Gratton E. Influence of cholesterol on phospholipid bilayers phase domains as detected by Laurdan fluorescence. Biophys J 1994; 66:120–132.
19. Prewitt JMS. Object enhancement and extraction. In: Rosenfeld A, editor. Picture Processing and Psychopictorics. New York: Academic Press, 1970.
20. Ridler TW, Calvard S. Picture thresholding using an iterative selection method. IEEE Trans Syst Man Cybern 1978; 8:630–632.

21. Roberts LG. Machine perception of three-dimensional solids. In: Tippet JT, editor. Optical and Electro-optical Information Processing. Cambridge, MA: MIT Press, 1965:159–197.
22. Russ JC. Image enhancement. In: The Image Processing Handbook. Boca Raton, FL: CRC Press, 1999:227–304.
23. Serra J. Image Analysis and Mathematical Morphology. London: Academic Press, 1982.
24. Sobel I. Camera Models and Machine Perception. AIM-21. Palo Alto; CA: Stanford Artificial Intelligence Laboratory, 1970.
25. Spencer T, Olson JA, McHardy KC, Sharp PF, Forrester JV. An image-processing strategy for the segmentation and quantification of microaneurysms in fluorescein angiograms of the ocular fundus. Comput Biomed Res 1996; 29(4):284–302.
26. Zhang T, Suen C. A fast parallel algorithm for thinning digital patterns. Commun ACM 1984; 27:236–239.
27. Zhu D, Hu C, Sheng W, Tan KS, Haidekker MA, Sun AY, Sun GY, Lee JC. NADPH oxidase-mediated reactive oxygen species alter astrocyte membrane molecular order via phospholipase A2. Biochem J 2009; 421:201–210.

3

IMAGE PROCESSING IN THE FREQUENCY DOMAIN

In previous chapters the image was introduced as a spatial arrangement of discrete values: image values that represent a physical metric. Neighboring pixels relate to each other in a defined spatial relationship. The human eye is very adept at recognizing spatial relationships, such as repeat patterns, irregularities (noise), edges, or contiguous features. For image processing software, this task is more difficult, because the analysis of large neighborhoods is usually very time consuming. In Section 2.3, filters were introduced that modified an image in a specific manner: for example, by enhancing discontinuities (sharpening) or attenuating detail (blurring). These filters act differently on different spatial frequencies. The term *spatial frequency* refers to the rate of change of the image values. An intensity trend that continues over most of the image consists of low spatial frequencies; conversely, an edge (a rapid change of intensity over a few pixels) contains high-frequency components. The frequency domain is a different representation of the same image data where the strength of periodic components (such as a pattern that repeats every 20 pixels) is extracted. The Fourier transform is a tool that converts the image with spatially arranged data into the same data arranged by periodicity, and thus, frequency. Any transform is characterized by the existence of an inverse transform that allows us to restore the original arrangement of the data. By using the Fourier transform, image data can be presented in a way that makes a number of image manipulations easier (or possible, in the first place): Images can be manipulated or filtered in the frequency domain. With the inverse Fourier transform, the filtered image is restored to its original spatial arrangement.

Advanced Biomedical Image Analysis, By Mark A. Haidekker

3.1. THE FOURIER TRANSFORM

French mathematician Joseph Fourier found that any 2π-periodic signal $f(t)$ can be represented by an infinite sum of sine and cosine terms according to

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos kt + b_k \sin kt \tag{3.1}$$

where a_k and b_k are called the *Fourier coefficients*. The term 2π-*periodic* indicates that the signal repeats after integer multiples of 2π, that is, $f(t) = f(t + 2\pi n)$, where n is an integer number. The periodicity is a consequence of the sine and cosine terms in Equation (3.1) and will become a fundamental assumption in the Fourier analysis of signals and images. The cosine and sine functions in Equation (3.1) form an orthogonal basis, since their inner product is zero:

$$\int_{-\pi}^{\pi} \sin\, t\, \cos\, t\, dt = 0 \tag{3.2}$$

When a signal $f(t)$ is given, the coefficients a_k and b_k can be determined using *Fourier analysis of the signal* $f(t)$:

$$\begin{aligned} a_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos kt\, dt \\ b_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin kt\, dt \end{aligned} \tag{3.3}$$

Equation (3.3) needs to be evaluated for $k = 0, 1, 2, \ldots, \infty$ to obtain the infinite number of coefficients that are needed to exactly represent the signal $f(t)$. The special case $k = 0$ yields $b_0 \equiv 0$, and a_0 represents the mean value of the signal over one period. In practice, a limited number of coefficients is sufficient to represent $f(t)$ with satisfactory accuracy. The coefficients a_k and b_k identify the contribution of the kth harmonic oscillation to the overall signal. The kth harmonic signifies an oscillation with k times the frequency of the base oscillation (i.e., the oscillation with a period of 2π). For this reason, Equation (3.3) is also called *harmonic analysis*.

Consider the piecewise linear, periodic oscillations in Figure 3.1A (square wave) and Figure 3.1C (triangular wave). The triangular wave closely approximates a sine wave (gray curve in Figure 3.1A). The sine wave has only one nonzero coefficient, $b_1 = 1$, as can be seen in Equation (3.1). Since both the square wave and the triangular wave can be described analytically, the integrals in Equation (3.3) can be solved to obtain the Fourier coefficients a_k and b_k. The average of the signal over one period is zero, and $a_0 = 0$ follows immediately. In fact, all a_k are zero because of the function's symmetries. Furthermore, solving the second integral in Equation (3.3) yields

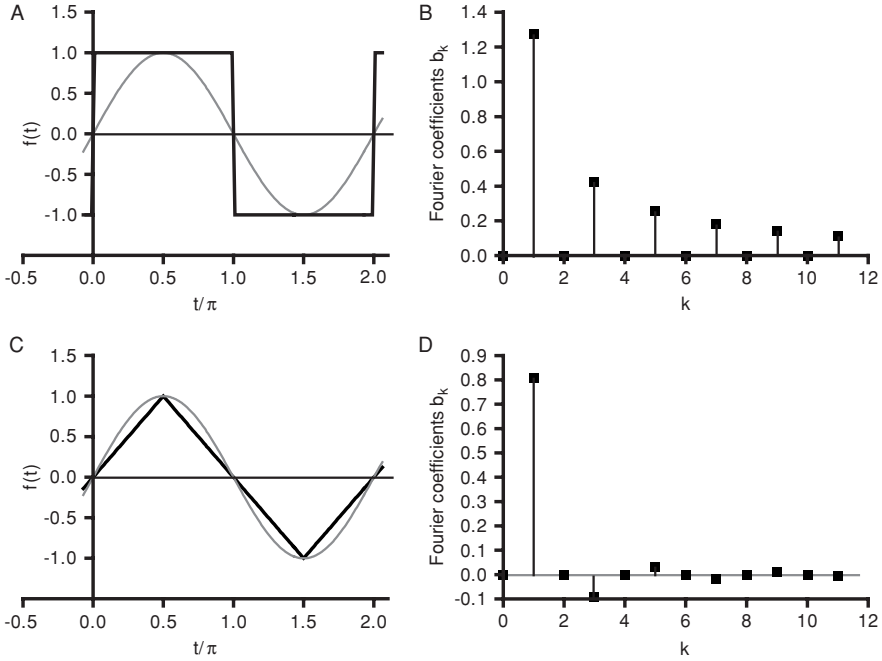


FIGURE 3.1 Fourier analysis of two periodic functions. A square function (A) and a triangular function (C) are shown next to a sine wave (gray curve in A and C). Solving the integrals in Equation (3.3) yields $a_k = 0$ for all k in both cases as a consequence of the symmetry. The b_k are plotted in B for the square wave and in D for the triangular function. It can be seen that the Fourier coefficients of the triangular function drop off more rapidly for higher k than those of the square wave, because the triangular function is more closely related to the sine wave than is the square wave.

$b_k = 0$ for all even values of k and nonzero coefficients b_k, as described for a square wave and a triangular wave, respectively:

$$b_k = \frac{4}{\pi k} \qquad k = 1, 3, 5, \ldots \tag{3.4}$$

$$b_k = \frac{8}{(\pi k)^2}(-1)^{(k-1)/2} \qquad k = 1, 3, 5, \ldots \tag{3.5}$$

Reconstruction of a signal from the Fourier coefficients is demonstrated in Figure 3.2. The individual harmonic oscillations for $k = 1$, 3, and 5 are shown in Figure 3.2A and C. According to Equation (3.1), the individual oscillations have integer multiples of the frequency and an amplitude decreasing with the reciprocal of the frequency. The summation of the oscillations (Figure 3.2B) shows convergence toward the final shape of the square-wave function. The harmonic coefficients of the triangle wave drop off more rapidly [according to Equation (3.5) with k^2], and convergence is much faster, as can be seen in Figure 3.2D.

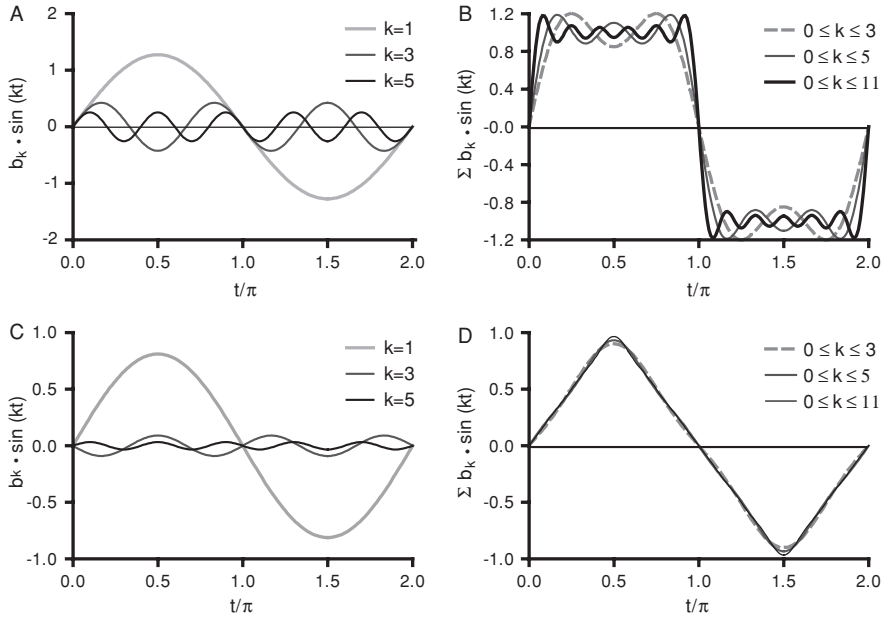


FIGURE 3.2 Fourier synthesis of the square-wave and triangular functions. The individual harmonic contributions (A and C, respectively), when added up, converge toward the original function as shown in B and D, respectively. For $k \to \infty$, the functions in Figure 3.1A and C emerge. Because the harmonic coefficients drop off toward higher frequencies faster in the triangular wave than in the square wave, the triangular function is better approximated with a lower number of coefficients.

The examples above show the transform of a continuous function into discrete Fourier coefficients. To apply the Fourier transform on discretely sampled signals and images, a discrete Fourier transform needs to be defined where the integration is replaced by a summation over the discrete samples of the signal. The one-dimensional discrete Fourier transform and the inverse transform are defined, respectively, by

$$\begin{aligned} F_r &= \frac{1}{N}\sum_{k=0}^{n-1} f_k \exp\left(-2\pi j \frac{rk}{N}\right) \\ f_k &= \sum_{r=0}^{N-1} F_r \exp\left(2\pi j \frac{rk}{N}\right) \end{aligned} \tag{3.6}$$

where the F_r are the complex Fourier coefficients of the signal $f_k = f(t_k)$ that is sampled at discrete time points $t_k = k\,\Delta t$. Equation (3.6) makes use of the Euler relationship (j is the imaginary unit)

$$\exp(j\varphi) := e^{j\varphi} = \cos\varphi + j\sin\varphi \tag{3.7}$$

and is therefore closely related to the discrete summation in Equation (3.1), with its orthogonal sine and cosine base. To obtain all possible Fourier coefficients, the summation in Equation (3.6) needs to be evaluated for $r = 0,1, \ldots, N-1$, where N is the number of data points in the time series.

Implicitly, periodic behavior of the time series f_k is assumed; that is, $f_k = f_{k+mN}$, where m may be any integer number. Due to the periodicity of the complex exponential, the same periodicity applies to the Fourier coefficients F_r: namely, $F_r = F_{r+mN}$. In fact, the Fourier coefficients are commonly interpreted such that the indices $r = 0, 1, 2, \ldots, N/2 - 1$ are considered coefficients for positive frequencies, and the indices $r = N - 1, N - 2, \ldots, N/2$ are considered negative frequencies and are shifted by subtracting N from r. The coefficient F_0, similar to a_0 in Equation (3.1), is the average value of the signal. F_0 is always real-valued. The indices r of the Fourier transform can be calibrated to relate to actual frequencies. When the signal has the period $T = N\,\Delta t$, the base frequency is $1/T$. Therefore, the frequency of the coefficient F_r for $r = \pm 1$ is $\pm 1/T$, and the frequency of each coefficient F_r is r/T. The highest possible frequency that the discrete Fourier transform provides is $N/2T$ or $1/2\Delta t$, in agreement with the Nyquist sampling theorem.

The same considerations apply to images. One horizontal line of pixels provides discrete values f_k, sampled with a constant spatial sampling interval Δx. Analogous to the time series, the indices r of the Fourier transform represent the spatial frequency $u_r = r/(M\Delta x)$. By convention, frequencies in the horizontal direction are given the index u, and frequencies in the vertical direction are given the index v. The two-dimensional Fourier transform of a two-dimensional image is computed by first computing the horizontal one-dimensional Fourier transform line by line, followed by the column-by-column Fourier transform of the horizontal Fourier transforms:

$$F(u,v) = \frac{1}{MN} \sum_{y=0}^{N-1} \left[\exp\left(-2\pi j \frac{vy}{N}\right) \sum_{x=0}^{M-1} I(x,y) \exp\left(-2\pi j \frac{ux}{M}\right) \right] \tag{3.8}$$

The second summation term represents the horizontal Fourier transform, and the first summation represents the vertical Fourier transform. Since the first exponential term is constant with respect to the second summation, the two exponential terms can be joined, leading to the most commonly used definition for the two-dimensional discrete Fourier transform:

$$F(u,v) = \frac{1}{MN} \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} I(x,y) \exp\left[-2\pi j \left(\frac{ux}{M} + \frac{vy}{N}\right)\right] \tag{3.9}$$

In Equations (3.8) and (3.9), $I(x,y)$ are the image values, M and N are the image size (number of pixels) in the horizontal and vertical directions, respectively, and $F(u,v)$ is the Fourier transform. Notably, the Fourier transform can be seen as an image with the same dimensions as the input image I, albeit with complex pixel values. Like the one-dimensional Fourier transform, those values of u that are larger than $M/2$

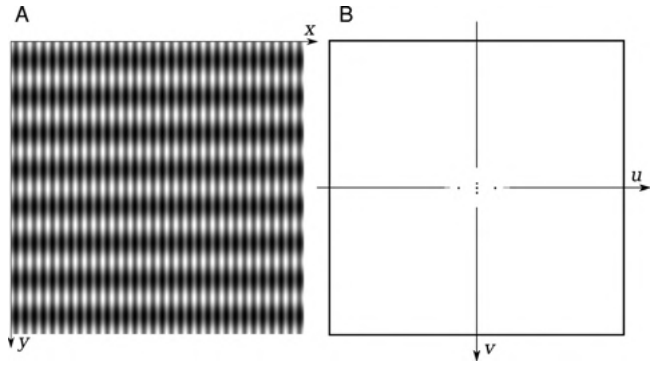


FIGURE 3.3 Two-dimensional Fourier transform. The spatial-domain image (A) contains periodic sinusoidal intensity variations; the image size is 512 × 512 pixels. Along the *y*-axis there are eight full waves, and along the *x*-axis, 32. Its Fourier transform (B, inverted with dark dots representing peak values for better visualization) shows five peaks. The central peak represents the image average. The two peak pairs along the *u* and *v* axes are the positive and negative frequency of the sinusoidal intensity variations along the *x* and *y* axes, respectively, of the original image. The spatial frequency along the *x*-axis is higher than that along the *y*-axis; correspondingly, the peaks on the *u*-axis are at higher *u* values (farther away from the origin) than are the peaks on the *v*-axis.

and those values of v that are larger than $N/2$ are converted to negative frequencies by subtracting M and N, respectively. The result is an image $F(u,v)$ with the origin ($u = 0$, $v = 0$) in the center, as illustrated in Figure 3.3. The value at the origin, $F(0,0)$, contains the image average as can be seen from Equation (3.9). A single row of pixels parallel to the *x*-axis would represent a sine wave with 32 full periods over $M = 512$ pixels or one period over 16 pixels. A one-dimensional Fourier transform of the single row would have three peaks: a peak in the center at $u = 0$ (the average value) and two symmetrical peaks 32 pixels right and left of the origin.

If the pixel size is known, the peak location translates into a spatial frequency. If we assume that this image has pixels of size 0.2 × 0.2 mm, one period along the *x*-axis is 3.2 mm. The highest spatial frequency (at $u = 256$) is 2.5 mm^{-1} and the spatial frequency at $u = 32$ is 2.5 $mm^{-1} \cdot 32/256$, or 0.3125 mm^{-1}, the reciprocal of 3.2 mm. Correspondingly, each column in Figure 3.3A shows sinusoidal oscillations of 64 pixels, or 12.8 mm along the *y*-axis. The peaks on the *v*-axis can be expected at $v = \pm 8$, which corresponds to 2.5 $mm^{-1} \cdot 8/256$ or 0.078125 mm^{-1}, the reciprocal of 12.8 mm. The farther away a pixel in the Fourier transform is from the origin, the higher its associated spatial frequency. In fact, all points with the same distance from the origin (i.e., on concentric circles) have the same spatial frequency. Therefore, any point (u,v) in the frequency-domain plane has the spatial frequency ω:

$$\omega = \sqrt{u^2 + v^2} \tag{3.10}$$

Since the Fourier transform is generally complex-valued, the visual representation is difficult. Most frequently, the magnitude of the frequency components $|F(u,v)|$ is

FIGURE 3.4 Demonstration of Fourier-based filtering. After removal of the peaks on the u-axis in Figure 3.3B and inverse Fourier transform, only the oscillations along the y-axis remain.

displayed and the phase information omitted. Furthermore, the dynamic range of the values of $F(u,v)$ is huge, and often contrast is improved by displaying $|F(u,v)|$ on a logarithmic scale, remapped to the displayable range from 0 to 255. Many times, the power spectrum, $|F(u,v)|^2$ is displayed. The power spectrum is a concept that originates in electrical engineering and relates to the distribution of the power of a signal over the frequency.

The basic idea behind Fourier-domain filters can be derived from the following experiment: If the two outer peaks on the u-axis in Figure 3.3B are removed, only the oscillation along the y-axis should remain. The experimental proof is given in Figure 3.4. The image in Figure 3.3A was Fourier-transformed, and the two symmetrical peaks on the u-axis were removed manually. After performing the inverse Fourier transform, only the oscillations along the y-axis remain. Filtering methods in the frequency domain and their applications are discussed in Section 3.2.

Abrupt changes of image intensity, such as lines or edges, have many frequency components. The Fourier transform of a step (such as the transition from 1 to 0) exhibits a broad spectrum with a (sin ω)/ω characteristic. If the intensity change is more gradual, its frequency components drop off more rapidly toward higher frequencies. Figure 3.5 demonstrates this effect. The Fourier transform of a white square (i.e., a step change of intensity in both the x and y directions) shows predominantly frequency components along the u and v axes (Figure 3.5A and B). A more gradual transition from black to white as in Figure 3.5C exhibits a much more rapid decay of the frequency components toward higher frequencies (Figure 3.5D). In the extreme, the gradual intensity change is sinusoidal (Figure 3.3A) and associated with a very narrow frequency peak. Notably, the Fourier transform of random pixel data (noise) also has a broad frequency spectrum. In other words, the spectral power of noise is spread over the entire frequency spectrum.

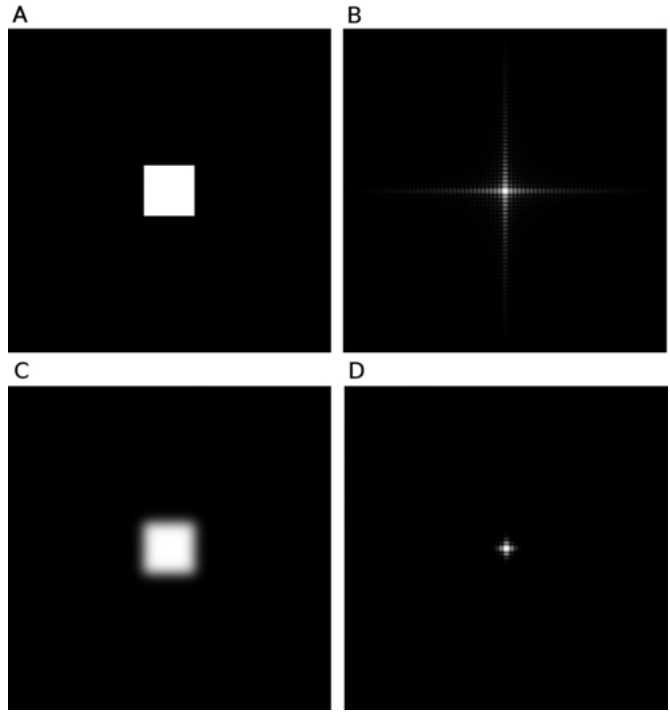


FIGURE 3.5 Frequency spectrum of abrupt intensity changes. A white square (step intensity changes in the x and y directions, A) has frequency components widely distributed along the u and v axes (B; shown is the logarithmic magnitude for better visualization). The more gradual, Gaussian-shaped transition (C) has frequency components that taper off much more rapidly toward higher frequencies (D).

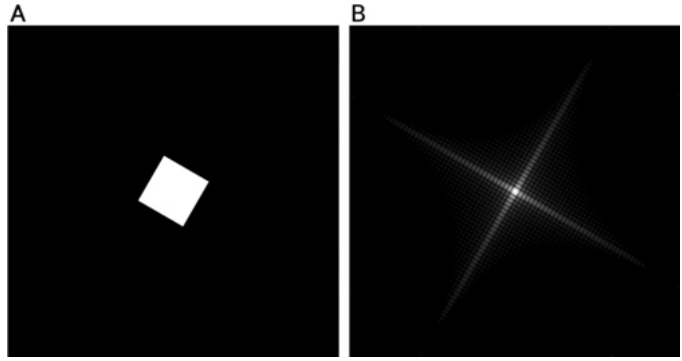


FIGURE 3.6 If an image is rotated, such as the white square from Figure 3.5A that is rotated clockwise by 30° in image A, its Fourier spectrum is that of the original image, but also rotated by the same angle (B).

If an image is rotated by a certain angle, the frequency spectrum of the rotated image is identical to the frequency spectrum of the original image but also rotated by the same angle (Figure 3.6). This is a consequence of the linearity of the Fourier transform, one of its most fundamental properties. In this context, linearity means that (1) a scaled image has a Fourier transform scaled by the same value, and (2) the Fourier transform of two images, added together pixel by pixel, is identical to the addition of the Fourier transforms of the individual images:

$$\begin{aligned} \mathcal{F}\{aI(x,y)\} &= a\mathcal{F}\{I(x,y)\} \\ \mathcal{F}\{I_1(x,y) + I_2(x,y)\} &= \mathcal{F}\{I_1(x,y)\} + \mathcal{F}\{I_1(x,y)\} \end{aligned} \tag{3.11}$$

where $\mathcal{F}$ denotes the Fourier transform; I, I_1, and I_2 are images; and a is a scalar.

The Fourier transform exhibits a number of symmetry properties. Generally, these are more important in one-dimensional data sets than in images, since images rarely exhibit relevant symmetries. If the original data are real valued, an even function $[f(t) = f(-t)]$ produces a real-valued, even Fourier transform, whereas an odd function ($[f(t) = -f(-t)]$) produces an imaginary, odd Fourier transform. Any asymmetrical function yields a complex Fourier transform. However, the Fourier transform always exhibits Hermitian symmetry, that is, $F(\omega) = F^*(-\omega)$, where F^* is the complex conjugate of F. In two dimensions, the Hermitian symmetry can be expressed as $F(u,v) = F^*(-u,-v)$.

Fast Fourier Transform The discrete Fourier transform (DFT) in Equation (3.9) can be implemented in a straightforward manner. Because of the double summation needed for every point of $F(u,v)$, the computational effort increases with the fourth power of the image size, specifically M^2N^2, if the image is $M \times N$ pixels in size. The extreme popularity of the Fourier transform can be explained primarily by the development of the fast Fourier transform (FFT). In one dimension the FFT reduces the computational effort from N^2 for the conventional discrete Fourier transform to $N \log_2 N$ for the FFT. To illustrate the efficiency gain, assume that a two-dimensional fast Fourier transform of a 1024×1024 pixel image takes 5 seconds on a personal computer. The conventional discrete Fourier transform would take 14 hours for the same image. It is easy to imagine how important this efficiency gain was in the early days of the digital computer, when computing power was only a fraction of what today's computers are capable of. This was the time when Danielson and Lanczos[8] created an artful recursive implementation of the DFT using the symmetries of the complex exponentials. The Danielson–Lanczos approach was refined further by Cooley and Tukey,[7] and the Cooley–Tukey algorithm is the basis of almost all FFT implementations today. The history of the discovery of fast Fourier algorithms is quite interesting, and a review paper by Cooley et al.[6] provides a historical outline. Additional aspects, particularly the impact of the FFT on research and science, as well as some speculation into future developments, are presented by Rockmore.[27]

To understand the FFT, one needs to recognize that the summation in Equation (3.6) can be split in odd and even halves:

$$\sum_{k=0}^{N-1} f_k \exp\left(-2\pi j \frac{rk}{N}\right) = \sum_{k=0}^{N/2-1} f_{2k} \exp\left(-2\pi j \frac{2rk}{N}\right) + \sum_{k=0}^{N/2-1} f_{2k+1} \exp\left(-2\pi j \frac{r(2k+1)}{N}\right) \tag{3.12}$$

In the second summation term, a constant phase factor of $\exp(-2\pi jr/N)$ appears, and the definition of a phase shift $W_N = \exp(-2\pi j/N)$ allows us to simplify Equation (3.12):

$$\cdots = \sum_{k=0}^{N/2-1} f_{2k} W_{N/2}^{rk} + W_N^r \sum_{k=0}^{N/2-1} f_{2k+1} W_{N/2}^{rk} \tag{3.13}$$

It can be seen from Equation (3.13) that the N-point DFT reduces to two $N/2$-point DFTs. The symmetry of the phase shifts comes into play when the number of summations for r is reduced from N to $N/2$ by using the following two equations for the lower and upper halves of the DFT:

$$\begin{aligned} F_r &= \sum_{k=0}^{N/2-1} f_{2k} W_{N/2}^{rk} + W_N^r \sum_{k=0}^{N/2-1} f_{2k+1} W_{N/2}^{rk} \\ F_{r+N/2} &= \sum_{k=0}^{N/2-1} f_{2k} W_{N/2}^{rk} - W_N^r \sum_{k=0}^{N/2-1} f_{2k+1} W_{N/2}^{rk} \end{aligned} \tag{3.14}$$

The upper and lower half-DFTs are identical, with the exception of the sign of the phase shift. This reduction of a N-point DFT into two $N/2$-point DFTs and some multiplications and additions is illustrated in the signal-flow graph in Figure 3.7.

In the same fashion, the two DFTs of $N/2$ points can be subdivided into four DFTs of $N/4$ points, and so on, until only two-point DFT operations remain. These operations are called *butterfly operations* and constitute the fundamental operation of the FFT. The phase shift W_N^r is often called a *twiddle factor*, and this factor differs only in the sign between the lower and upper halves of Equation (3.14). The final step in achieving the Cooley–Tukey FFT is to recognize that the butterfly operation can be applied to adjoining data points if the data are rearranged in a specific manner: namely, input data elements i and j are exchanged where j, interpreted as a binary value, has the mirrored bit pattern of i. This two-step process of first rearranging the data and then performing the recursive butterfly operations is given in Algorithm 3.1. Most notably, the FFT is not an approximation of the DFT but, rather, provides identical data. The major disadvantage of the split-by-two approach is the restriction

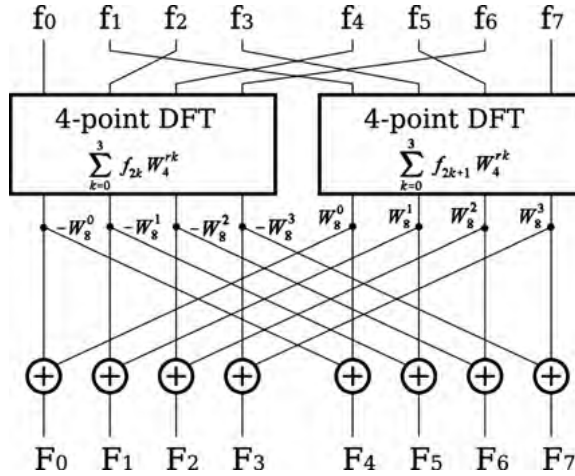


FIGURE 3.7 Reduction of an eight-point DFT into two four-point DFTs with eight subsequent multiplications and additions following Equation (3.14). Multiplication with the twiddle factors occurs along the diagonal lines. By using the same scheme, the four-point DFTs can be subdivided into two-point DFTs. The two-point DFT is the most fundamental operation and cannot be subdivided further.

of the FFT to data lengths N to powers of 2. If N is not a power of 2, the sequence needs to be padded with zeros to the next power of 2, which may offset some of the computational advantage (e.g., a 640×480 pixel image needs to be padded to form a 1024×512 pixel image for FFT).

The implementation of the FFT in Algorithm 3.1 is straightforward, although simple complex arithmetic needs to be provided with the FFT function. Some languages (e.g., FORTRAN) provide complex arithmetic. In C, a structure can be defined that contains two double-precision elements for the real and imaginary parts, respectively; simple complex arithmetic functions with two parameters that return the sum, difference, and product of the parameters need to be defined. In C++, the operators $+$, $-$, and * can be overloaded. A very efficient implementation of the FFT function that does not rely on external complex arithmetic is provided in *Numerical Recipes in C*.[23] In the same book, multidimensional FFT implementations are presented. Multidimensional FFTs can be made more efficient by using the symmetry of the exponential phase terms. Additional efficiency gains can be achieved when the input data are real-valued rather than complex-valued, which is almost always the case in image processing. Since many efficient implementations of the FFT exist, Algorithm 3.1 is provided for reference only. It is, of course, educational to implement the code in Algorithm 3.1 and monitor how the bit-reversal part and the Danielson–Lanczos algorithm operate.

One of the most advantageous options for performing an FFT is to use available FFT libraries. A freely available library can be accessed at http://www.fftw.org,[11] where fftw is an acronym for the "Fastest Fourier Transform in the West." Those

```
// Part 1: Bit-reversal reordering of the data: i and j are bit-mirrored
j=0;
for (i=0 while i<N-1 increment i=i+1)
   if (i<j) then
      swap (data[i], data[j]);      // Swap the bit-mirrored positions
   endif;
   k=N/2;
   while ((k<=j) and (k>=1)) do  // Prepare the bit mirrored j for the
                                 next i
      j = j-k;
      k = k/2;
   endwhile;
   j = j+k;
endfor;

// Part 2: Danielson–Lanczos recursive multiplication
k = 1;
while (k<N) do    // k is doubled every iteration, so this loop runs log2 N times
   theta = π/k;      // angle increment for this recursion level
   v = complex (-2*sin(0.5*theta)^2, -sin(theta));
   w = complex (1, 0);                    // w is the actual twiddle factor

   for (m=1 while m<=k increment m=m+1) do
      for (i=m-1 while i<N increment i=i+2*k) do
         j=i+k;
         c = w ⊗ data[j];                // These three lines constitute
                                         the butterfly operation
         data[j] = data[i] ⊖ c;
         data[i] = data[i] ⊕ c;
      endfor;
      w = w ⊗ v ⊕ w;
   endfor;
   k = k*2;
endwhile;
```

Algorithm 3.1 Fast Fourier transform in one dimension. This function depends on a complex input array `data` with N elements (0 to $N-1$), where N must be a power of 2. The FFT function needs an external function `swap` that exchanges the two arguments. The variables `v`, `w`, and `c` are complex numbers. Furthermore, the symbols $\otimes$, $\oplus$, and $\ominus$ denote complex multiplication, addition, and subtraction, respectively. The Fourier transform takes place within the `data` array, and the output needs to be normalized, that is, all elements of `data` divided by N. The inverse Fourier transform is almost identical except that `theta` in part 2 has a negative sign, and no normalization takes place.

familiar with the fftw library have little reason to doubt the veracity of the developer's claim. Not only does the fftw library contain several different algorithms optimized for different types of data in one or multiple dimensions, but a measurement function exists that determines the optimum algorithm for a given situation. Whereas the initial performance measurement constitutes computational overhead, the overall gain for the repeated use of the same FFT is considerable. For single FFT where the overhead is not justified, fftw estimates the optimum algorithm without additional computational effort. Another advantage of the fftw library is that input data are not restricted to power-of-2 sizes. Rather, fftw is capable of computing the FFT of arbitrary-sized data sets. Furthermore, fftw provides functions for real-valued input and contains code for the discrete sine and cosine transforms and the fast Hartley transform (see Section 3.3). With these capabilities, fftw is arguably the library of choice for most applications that call for discrete Fourier transforms and related integral transforms.

3.2. FOURIER-BASED FILTERING

In Section 3.1, the possibility of using the Fourier transform to remove periodic components from images was introduced briefly. However, frequency-domain filtering is far more comprehensive. The foundation of frequency-domain filtering is the convolution theorem, which stipulates that a convolution in the spatial domain corresponds to a multiplication in the Fourier domain. Computational effort of a convolution increases with K^2N^2 on $N \times N$ images and $K \times K$ kernels. With the fast Fourier transform, computational expense is reduced to $2\ N \log_2 N$ and $N \times N$ additional multiplications. On large images and large convolution kernels, Fourier filtering is considerably faster. Furthermore, due to the defined spatial frequencies, the action of the Fourier-domain filter can be dramatically better adjusted than that of convolution filters (which often are merely approximations of filter functions in a small neighborhood, such as a 3×3 kernel). A Fourier-domain filtering process includes three steps: computation of the Fourier transform of the image; multiplication with the filter function and computation of the inverse Fourier transform (Figure 3.8). A fourth and optional step, windowing, is discussed later in the chapter.

Two of the most commonly used filter types are lowpass filters (where the higher frequencies are attenuated) and highpass filters (where the low-frequency components are attenuated). Lowpass filters have noise-attenuating properties and blur the image. Lowpass filtering is associated with a loss of detail and with softening of sharp transitions. Highpass filtering enhances edges and may be used to remove inhomogeneous background components. Highpass filters also tend to increase the noise component. Bandpass filters and notch filters are used less frequently in image processing. A bandpass filter emphasizes specific periodic components, whereas a notch filter suppresses periodic components. The band of frequencies that are not attenuated is called the *passband*, and the band of frequencies that are attenuated is called the *stopband*. The frequency at which the transition between passband and stopband takes place is the *cutoff frequency*.

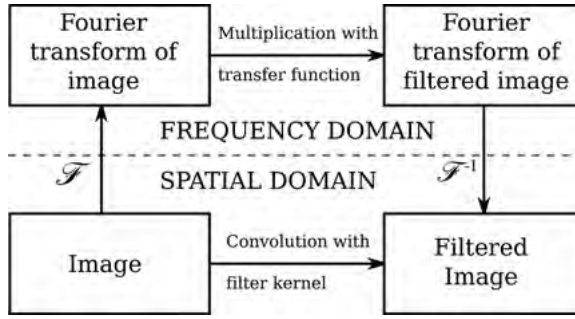


FIGURE 3.8 Filtering in the frequency domain. Instead of filtering the image in the spatial domain by convolving the image with the filter kernel, the image can be transformed into frequency space by using the FFT (indicated by the symbol $\mathscr{F}$). The Fourier transform of the image is then multiplied by the filter function (also called the transfer function, i.e., the Fourier transform of the convolution kernel), and the resulting frequency-space filtered image is subjected to the inverse Fourier transform ($\mathscr{F}^{-1}$). Although this "detour" into the frequency domain may appear to be additional effort, the overall computational expense is dramatically less than for a convolution with large kernel sizes.

Filters are usually designed to act equally on all spatial frequencies, making them rotationally symmetrical [see Equation (3.10)]. Furthermore, filter functions are designed to achieve a smooth transition between frequency components that are allowed to pass and those that are blocked. Steep transitions cause an artifact known as *ringing*. For this reason, a filter function such as a lowpass filter with a frequency response $H(\omega)$,

$$H(\omega) = \begin{cases} 1 & \text{for } \omega \leq \omega_0 \\ 0 & \text{for } \omega > \omega_0 \end{cases} \tag{3.15}$$

with the cutoff frequency ω_0 is not advantageous and not normally used.

3.2.1. Lowpass Filters

Lowpass filters are used primarily for noise reduction. They remove image detail and blur the image. Lowpass filters are often applied in preparation for segmentation to reduce the influence of noise. In the spatial domain, blurring is often performed by a convolution with an approximated Gaussian kernel. Interestingly, the Fourier transform of a Gaussian function is again a Gaussian function. Therefore, a popular lowpass filter is a filter with Gaussian-shaped frequency response

$$H(\omega) = \frac{A}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\omega^2}{2\sigma^2}\right) \tag{3.16}$$

where $\omega = \sqrt{u^1 + v^2}$ is the spatial frequency, A is the overall gain, and σ is the width (steepness).

An alternative implementation uses

$$H(\omega) = \frac{A}{\sigma\sqrt{2\pi}} \exp\left[-\frac{\text{clamp}(\omega - \omega_0)^2}{2\sigma^2}\right] \tag{3.17}$$

where the function clamp(x) is 0 for $x < 0$ and x for $x \geq 0$. With the help of the clamp function, the passband frequencies are extended such that all frequency components below ω_0, the cutoff frequency, remain unchanged.

Another frequently used lowpass filter function is the Butterworth filter. Its filter function has the steepest possible transition between passband and stopband without passband ripples (ringing). Its frequency response is

$$H(\omega) = \frac{A}{1 + C\,(\omega/\omega_0)^{2n}} \tag{3.18}$$

where A is the overall gain, ω_0 the cutoff frequency, C an asymmetry factor, and n the order.

Selection of the filter and the filter parameters is often empirical. Examples of the action of the filters are shown in Figure 3.9. Whereas similarly adjusted lowpass filters with Gaussian and Butterworth characteristics (Figure 3.9B and C) are visually similar, the edges are blurred less strongly with the Butterworth filter. The effect of a steep transition (ripples or ringing, seen as periodical oscillations near edges) is demonstrated in Figure 3.9D where a Butterworth filter of order $n = 25$ was used.

3.2.2. Highpass Filters

In general, highpass filters $H_{\text{HP}}(\omega)$ can be obtained from lowpass filters $H_{\text{LP}}(\omega)$ through

$$H_{\text{HP}}(\omega) = 1 - H_{\text{LP}}(\omega) \tag{3.19}$$

provided that the passband gain is normalized to unity. In the example of the Butterworth lowpass filter [Equation (3.18)] application of Equation (3.19) leads to the complementary Butterworth highpass filter,

$$H(\omega) = \frac{A}{1 + (1/C)\,(\omega_0/\omega)^{2n}} \tag{3.20}$$

The choice of a relatively steep filter with a low cutoff frequency provides a filter that partly removes an inhomogeneous background (an image component with a very low frequency), while a higher cutoff frequency leads to a distinctly edge-enhancing filter, similar to convolution-based edge detectors. Specifically for the removal of an inhomogeneous background, the filter

$$H(\omega) = A + \frac{1}{1 + \exp[-s(\omega - \omega_0)]} \tag{3.21}$$

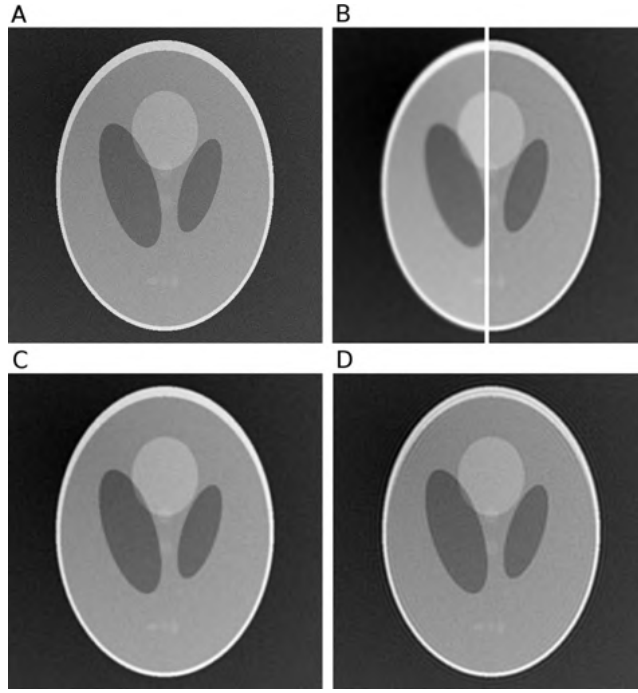


FIGURE 3.9 Effect of Fourier-domain lowpass filters. The original image is a high-contrast version of the Shepp–Logan head phantom with additive Gaussian noise and a gradual background gradient (A). A Gaussian lowpass filter with $\sigma = 30$ pixel^{-1} [B; left half: Equation (3.16), right half: Equation (3.17) with $\omega_0 = 20$ pixel^{-1}] reduces noise but also causes some degradation of the edges (blurring). A second-order Butterworth filter [Equation (3.18), $\omega_0 = 66$ pixel^{-1}, $C = 1$] produces a similar effect (C). If the transition from the passband to the stopband is very steep, for example, by using a high-order Butterworth filter (D, same parameters as C except that $n = 25$), edges are accompanied by spatial oscillations ("ringing").

is more suitable. In this filter, s is the steepness, ω_0 the cutoff frequency, and A an offset that can be used to adjust the strength of the filter. After application of the filter in Equation (3.21), the value range needs to be normalized if $A \neq 0$. Examples of the application of these filters are shown in Figure 3.10.

Very frequently, the filter in Equation (3.21) is applied in a multiplicative manner. In this case, the filter is known as a *homomorphic filter*. The underlying assumption is inhomogeneous illumination, where the image values $I(x,y)$ are the product of nonuniform illumination intensity $I_0(x,y)$ and the reflectance (or transmittance) of the imaged object, $O(x,y)$. In logarithmic form, the product becomes a sum, $\log I(x,y) = \log I_0(x,y) + \log O(x,y)$. The illumination intensity function $I_0(x,y)$ is assumed to have only low-frequency components. When the log-transformed data $\log I\,(x,y)$ are filtered with the filter in Equation (3.21), the low-frequency additive component $\log I_0(x,y)$ is strongly attenuated, and only the object component remains. By raising the

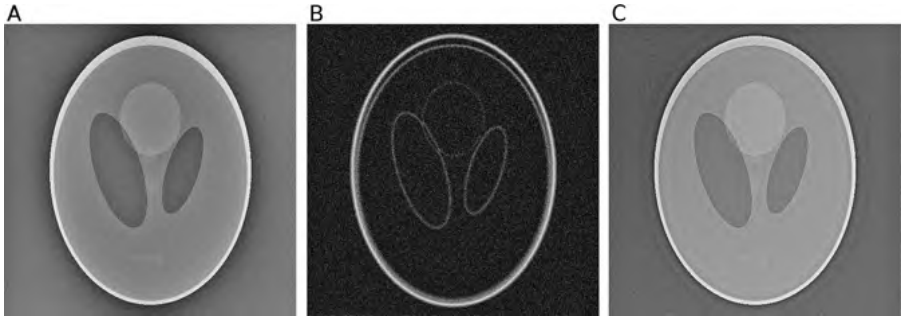


FIGURE 3.10 Highpass filtering of the phantom in Figure 3.9A. A Butterworth filter with low cutoff frequency ($\omega_0 = 2$ pixel^{-1}, $C = 1$, $n = 1$, $A = 1$) partly removes the background inhomogeneity, but it also introduces a new region of reduced intensity around the edge (A). With a higher cutoff frequency ($\omega_0 = 25$ pixel^{-1}), the filter assumes a strong edge-enhancing characteristic, particularly when the absolute values are taken (B). The filter in Equation (3.21) ($\omega_0 = 20$, $A = 2$, $s = 0.5$) effectively removes the background inhomogeneity (C) while exhibiting less of the reduced-intensity area around the phantom than does the Butterworth highpass in part A.

filtered log-transformed image to the power of 10, the original image is restored with the multiplicative background inhomogeneity removed.

3.2.3. Wiener Filters

Frequency-domain filters play an important role in image restoration. Any imaging device has a certain point-spread function (PSF), that is, the image function of an idealized point source. For an optical system, the PSF could be measured by acquiring the image of a back-illuminated pinhole. For a CT system, the PSF could be determined by imaging a thin axial wire. Most frequently, the PSF has a Gaussian shape. If the point-spread function $g(x,y)$ of an imaging device is known, any image $I(x,y)$ obtained by this device can be seen as the convolution of an idealized image $I'(x,y)$ with $g(x,y)$. Let $G(u,v) = \mathcal{F}\{g(x,y)\}$ be the Fourier transform of $g(x,y)$; then degradation by the PSF can be described in the frequency domain by

$$\mathcal{F}\{I(x,y)\} = \mathcal{F}\{I'(x,y)\}G(u,v) \tag{3.22}$$

The idealized but inaccessible image $I'(x,y)$ can therefore be restored by deconvolution, that is, division by the PSF in the frequency domain, described by

$$I'(x,y) = \mathcal{F}^{-1}\left\{\frac{\mathcal{F}\{I(x,y)\}}{G(u,v)}\right\} \tag{3.23}$$

The problem with this approach is that the point-spread function $G(u,v)$ has a low-pass character, which means that the values of $G(u,v)$ become very small for large

u and v. Since the measured image data $I(x,y)$ usually contain noise, the Fourier transform of $I(x,y)$ also contains noise. This noise component is broadly distributed and a considerable amount of noise exists in the Fourier transform of the image for large u and v, where the division by small values of $G(u,v)$ causes a massive noise amplification. Therefore, if an image is degraded by a lowpass point-spread function and additive noise, it is not possible to reconstruct the undegraded image. However, a filter function can be designed that provides an optimum estimate that minimizes the squared distance between I and I'. This best-estimate filter is called a *Wiener filter*. Often, the noise component can be measured or estimated, for example, by analyzing a flat region of the image. A common model is additive noise, where each pixel contains a noise component $\epsilon(x,y)$ with zero mean and a standard deviation of σ. It can be shown that the best-estimate filter is the function $W(u,v)$:

$$W(u,v) = \frac{G^*(u,v)}{|G(u,v)|^2 + |N(u,v)|^2/|I(u,v)|^2} \tag{3.24}$$

where $G^*(u,v)$ is the conjugate complex of the frequency-domain PSF, $N(u,v)$ is the estimated Fourier spectrum of the noise, and $I(u,v)$ is the Fourier spectrum of the original image and approximately similar to the Fourier spectrum of the degraded image, provided that the degradation is minor. To obtain the best-estimate approximation of the ideal image, the degraded image needs to be multiplied in the Fourier domain with $W(u,v)$, and Wiener filtering becomes the process

$$I'(x,y) \approx \mathscr{F}^{-1}\{\mathscr{F}\{I(x,y)\}W(u,v)\} \tag{3.25}$$

Sometimes the Wiener filter is implemented with an adjustable parameter k:

$$W(u,v) = \frac{G^*(u,v)}{|G(u,v)|^2 + k} \tag{3.26}$$

In this case, k can be adjusted manually to provide an acceptable approximation of the restored image. The main effect of the real-valued scalar k is to prevent the filter function $1/G(u,v)$ from assuming very large values for large u and v and thus amplifying the noise component. The effect of the Wiener filter and the choice of k are illustrated in Figure 3.11.

3.2.4. Cross-Correlation and Autocorrelation

Cross-correlation is a method for template matching. A template, $g(x,y)$, is slid over the image $I(x,y)$ and overlapping pixels are summed up. This is, in fact, equivalent to a convolution operation

$$C(x,y) = \frac{\sum_{j=0}^{N-1}\sum_{i=0}^{M-1} I(i,j)g(x+i,\, y+j)}{\left[\sum_{j=0}^{N-1}\sum_{i=0}^{M-1} I(i,j)^2 \sum_{j=0}^{N-1}\sum_{i=0}^{M-1} g(i,j)^2\right]^{1/2}} \tag{3.27}$$

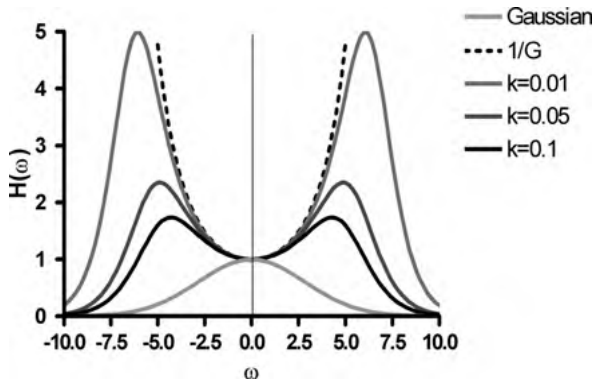


FIGURE 3.11 Frequency response of a Wiener filter as defined in Equation (3.26). Assume that the degradation function $G(\omega)$ is a Gaussian function (light gray line); then its reciprocal (dashed line) assumes very large values at moderate to high spatial frequencies. Multiplication of a degraded image by the reciprocal would strongly amplify noise. Instead, a Wiener filter can be used to attenuate very high frequencies. The frequency response of the filter in Equation (3.26) is shown for three different choices of k.

The template $g(x,y)$ and similar shapes can be found at peak locations of $C(x,y)$. By using the convolution theorem, the convolution in Equation (3.27) can be written as a multiplication in the frequency domain:

$$C(x,y) = \mathcal{F}^{-1}\{\mathcal{F}\{I(x,y)\}G^*(u,v)\} \tag{3.28}$$

where $G^*(u,v)$ is the conjugate complex of the Fourier transform of $g(x,y)$. The autocorrelation is the cross-correlation of the image with itself, and Equation (3.28) applies in the same manner.

3.2.5. Filter Implementation and Windowing

For the implementation of symmetrical lowpass and highpass filters, a filter function can be provided that returns $H(\omega)$ for any given ω. To filter an image with this filter function, the image needs to be subjected to a FFT, and the FFT image data need to be rearranged so that the zero-frequency element $F(0,0)$ is in the center. In a loop over all pixels (u,v) of the FFT image, the frequency response is obtained by computing ω as the Euclidean distance of (u,v) from the center and calling the filter function to obtain H for this frequency. Both the real and imaginary parts of the FFT values are then multiplied by H. Once all pixels are processed, the resulting filtered FFT image is subjected to the inverse Fourier transform. Since the result of the inverse FFT is usually complex-valued, the imaginary component can be discarded (it should contain values very close to zero, anyway) and the resulting spatial-domain real-valued pixel values rounded to fit the original image depth.

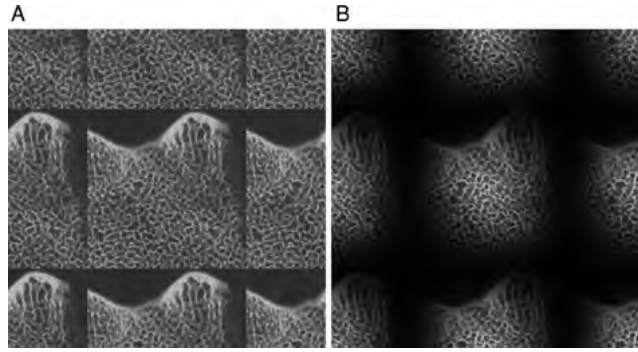


FIGURE 3.12 Discontinuities at the edges of images that are assumed to be periodic. The example shows a magnified section of an x-ray image of bovine bone. The Fourier transform assumes that images are 2π-periodic, that is, infinitely tiled, as illustrated in part A. This creates discontinuities at the image edges with associated high-frequency components. By multiplying the image by a window function, the image values at the edges are attenuated and a forced continuity at the edges is created (B).

An important consideration for the implementation of Fourier-domain filters, even for the estimation of spectral components, is the assumed periodicity of the image. In Section 3.1 the Fourier transform was introduced under the assumption that any signal is 2π-periodic, and any image consequently is 2π-periodic in both directions. It was also explained that edges create high-frequency components. Discontinuities between image values at the left and right border and at the upper and lower border also count as discontinuities, as can be seen in Figure 3.12. These discontinuities create additional high-frequency components in the Fourier transform. To prevent these high-frequency components, it is recommended that the image be multiplied by a window function in the spatial domain, a process called *apodization*. The window function attenuates image values toward the edges and therefore reduces the discontinuity (Figure 3.12B). After filtering and inverse Fourier transform, the original image values are restored by dividing the filtered image by the window function.

A window function has its own frequency response. For this reason, smooth window functions are preferred that contribute only minimally to the frequency spectrum. One example window function is

$$w(x, y) = \left[a - b\cos\left(2\pi\frac{x}{M+1}\right)\right]\left[a - b\cos\left(2\pi\frac{y}{N+1}\right)\right] \tag{3.29}$$

where M and N are the image dimensions in the x and y directions, respectively. With the choice of $a = 0.5$ and $b = 0.5$, this window is known as the *Hann window*, and with $a = 0.54$ and $b = 0.46$, the window function becomes the *Hamming window*. The Hamming window is one specific form of raised cosine window where the window values do not drop to zero at the edges. This is important for recovery of the image values after the inverse Fourier transform, because edge pixels where

$w(x,y)$ drops to zero cannot be recovered by dividing by $w(x,y)$. Other windows of linear (Bartlett window), parabolic (Welch window), or Gaussian shape exist. In the practical application the exact choice of the window function will not have a measurable impact on the filtered image as long as the window function approaches zero smoothly toward the edges of the image.

The summary of these steps is presented in Algorithm 3.2. The algorithm relies on two external functions, FFT and IFFT, which perform the forward and inverse Fourier transforms, respectively, in the image IM and its real and imaginary parts of the Fourier transform, IFR and IFI.

```
set a=1.0; c=1.0; n=2; omg0=25;   // Specify Butterworth filter parameters [Equation (3.18)]
allocate (IFR(M,N), IFI(M,N));    // Create space for the Fourier transform

// Part 1: Apply a window function (Hamming window), then Fourier-transform the image
for (y=0 while y<N-1 increment y=y+1)
   for (x=0 while x<M-1 increment x=x+1)
      w=(0.54-0.46*cos(2*π*x/(M+1))*(0.54-0.46*cos(2*π*y/(N+1));
      IM(x,y) = IM(x,y)*w;
   endfor;
endfor;
call FFT (IM, IFR, IFI, M, N);                  // the forward Fourier transform

// Part 2: Apply the filter function in the frequency domain. Note that IFR and IFI are M × N images.
for (v=0 while v<N-1 increment v=v+1)
   for (u=0 while u<M-1 increment u=u+1)
      omega=sqrt(sqr(u-N/2)+sqr(v-M/2));        // frequency of (u,v)
      H = a/(1+c*power((omega/omg0), 2*n));     // Butterworth lowpass
      IFR(u,v) = IFR(u,v)*H;                    // Multiply both real and ...
      IFI(u,v) = IFI(u,v)*H;                    // ... imaginary components by H
   endfor;
endfor;

// Part 3: Inverse Fourier-transform followed by reversal of the window function
call IFFT (IFR, IFI, IM, M, N);                 // the inverse Fourier transform
for (y=0 while y<N-1 increment y=y+1)
   for (x=0 while x<M-1 increment x=x+1)
      w=(0.54-0.46*cos(2*π*x/(M+1))*(0.54-0.46*cos(2*π*y/(N+1));
      IM(x,y) = IM(x,y)/w;
   endfor;
endfor;

delete (IFR(M,N), IFI(M,N));      // free space used for the Fourier transform
```

Algorithm 3.2 Fourier-based filtering using the example of a Butterworth lowpass filter. This algorithm relies on external functions to perform the Fourier transform. `FFT` transforms image `IM` and generates the real part `IFR` and the imaginary part `IFI` of the Fourier transform. The inverse transform function `IFFT` takes `IFR` and `IFI` and returns `IM`. This algorithm demonstrates how a window function is applied in the spatial domain and how a filter function is applied in the frequency domain.

3.3. OTHER INTEGRAL TRANSFORMS: THE DISCRETE COSINE TRANSFORM AND THE HARTLEY TRANSFORM

Other integral transforms exist that have properties similar to those of the Fourier transform but are real-valued. Most notably, these are the discrete cosine transform[26] and the discrete Hartley transform.[4,16] For both of them, a fast recursive formulation similar to the FFT exists.[5] Like the Fourier transform, both transforms decompose the image into its frequency components. The discrete cosine transform finds widespread use in lossy image compression (see Section 12.3), and the discrete Hartley transform is sometimes used as a real-valued substitute for the Fourier transform in frequency-domain filter operations. In fact, some image processing programs, such as NIH Image and ImageJ, really compute the discrete Hartley transform, although the menu indicates the Fourier transform. The advantage of the Hartley transform is that it does not require dealing with complex numbers. However, both the discrete cosine transform and the discrete Hartley transform have fundamentally different periodicity and symmetry properties than those of the Fourier transform. Consequently, some theorems, including the convolution theorem, are different from, and notably more complex than, the equivalent theorems of the Fourier transform. In addition, patent restrictions that were held up until 1995 may have hampered a more widespread adoption of the fast Hartley transform.

The two-dimensional discrete cosine transform and its inverse transform are given, respectively, by*

$$C(u,v) = \frac{s(u)s(v)}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x,y) \cos\left[\frac{\pi}{2} \frac{u(2x+1)}{M}\right] \cos\left[\frac{\pi}{2} \frac{v(2y+1)}{N}\right] \tag{3.30}$$

$$I(x,y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} S(u)S(v)C(u,v) \cos\left[\frac{\pi}{2} \frac{u(2x+1)}{M}\right] \cos\left[\frac{\pi}{2} \frac{v(2y+1)}{N}\right] \tag{3.31}$$

where the scaling factor $s(k)$ assumes the values $s(0) = \sqrt{2}$ and $s(k) = 2$ for all other k. Whereas the Fourier transform assumes 2π-periodicity of the input signal, the discrete cosine transform assumes the input data to be only one-half of the 2π period. The other half is mirrored. Consider the one-dimensional input sequence to be $x_0, x_1, \ldots, x_{N-1}$; then the sequence is assumed to continue with $x_{-1} = x_0$, $x_{-2} = x_1, \ldots$ to the left, and $x_N = x_{N-1}, x_{N+1} = x_{N-2}, \ldots$ to the right. This behavior is identical to a discrete Fourier transform when the input data are real-valued with even symmetry and all even-indexed elements are zero. Furthermore, the discrete cosine transform does not yield negative frequency components as the Fourier transform does. For this reason, the origin with zero frequency cannot be shifted into the center as shown in Figure 3.3. Rather, the $M \times N$ data points of the discrete cosine transform

*In fact, four different definitions of the DCT, often called DCI-I through DCT-IV, are defined by the symmetry conditions. These behave in a similar manner. In this chapter we use DCT-II as the most widespread form of DCT in image processing.

represent the positive part of the spectrum only. To achieve a spectrum similar to the one obtained from the Fourier transform, the $M \times N$ point spectrum needs to be mirrored along the u-axis, then the new spectrum mirrored again along the v-axis, leading to a $2M \times 2N$ spectrum. This mirroring process generates four zero-frequency points and is therefore rarely performed. However, even the one-quadrant spectrum of positive frequencies provides the ability to design filters based on the discrete cosine transform.

The discrete Hartley transform is more closely related to the Fourier transform than the discrete cosine transform. In fact, Hartley envisioned his integral transform as a real-valued replacement of the Fourier transform for real-valued data.[3] Whereas the Fourier transform uses the harmonic complex oscillation as its basis function, the Hartley transform uses the *cas function*. The term *cas* is an acronym for cosine and sine, and the *cas* function is defined as

$$\operatorname{cas} t = \cos t + \sin t = \sqrt{2} \cos\left(t - \frac{\pi}{4}\right) \tag{3.32}$$

The one-dimensional discrete Hartley transform is given by

$$H(\omega) = \frac{1}{N} \sum_{i=0}^{N-1} f_i \operatorname{cas}\left(2\pi \frac{\omega i}{N}\right) \tag{3.33}$$

The discrete Hartley transform is its own inverse; in other words, the inverse Hartley transform is identical to the forward Hartley transform. In Equation (3.38) the inverse transform is obtained by exchanging $I(x,y)$ and $H(u,v)$ and performing the summation over u and v. The Fourier and Hartley transforms are closely related. The Hartley transform can be computed from the real and imaginary components of the Fourier transform through the equation[3]

$$H(\omega) = \operatorname{Re}\{F(\omega)\} - \operatorname{Im}\{F(\omega)\} \tag{3.34}$$

and the Fourier transform can be obtained by splitting the Hartley transform into its even and odd components, which then constitute the real and imaginary parts of the Fourier transform[4]:

$$F(\omega) = \frac{H(\omega) + H(N-\omega)}{2} - j\frac{H(\omega) - H(N-\omega)}{2} \tag{3.35}$$

Similar to the Fourier transform, the signal is assumed to be periodic, and its Hartley transform is also periodic in the sense of $H(k) = H(k + mN)$, where N is the length of the data set and m is an integer number. The discrete Hartley transform yields the transform components for negative frequencies in the upper half of the transformed data. For this reason, the shift $H(-k) = H(N-k)$ is usually performed to move the zero-frequency point into the center of the data stream or, in the two-dimensional case, into the center of the image. The magnitude of the Hartley transform $|H(u,v)|$ and the

magnitude of the Fourier transform $|F(u,v)|$ are very similar in appearance. Frequency-domain filtering is possible with the Hartley transform, but the convolution theorem differs from that of the Fourier transform. If two signals g_1 and g_2 are convolved into a signal g, the convolution theorem for the discrete Hartley transform requires that we split one of the transforms into its even and odd components such that $G_1 = \mathcal{H}\{g_1\}$ is the Hartley transform of g_1, G_{1e} is the even part of G_1 with $G_{1e}(\omega) = \frac{1}{2}[G_1(\omega) + G_1(N-\omega)]$, and G_{1o} is the odd part of G_1 with $G_{1o}(\omega) = \frac{1}{2}[G_1(\omega) - G_1(N-\omega)]$, in analogy to Equation (3.35). With these definitions, the Hartley transform $G = \mathcal{H}\{g\}$ of the convolved signal can be computed through[4]

$$G(\omega) = G_1(\omega)G_{2e}(\omega) + G_1(-\omega)G_{2o}(\omega) \tag{3.36}$$

When one of the functions (e.g., a Gaussian filter kernel) is even, $G_{2o}(\omega) = 0$ follows, and the second summation term in the convolution theorem [Equation (3.36)] can be dropped.

In two dimensions, additional considerations arise, as there are two possible extensions of the one-dimensional discrete Hartley transform[34] with mutually exclusive properties. The following equation describes a two-dimensional Hartley transform that maintains the relationship between the Fourier and Hartley transform in Equations (3.34) and (3.35):

$$H(u,v) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x,y) \operatorname{cas}\left[2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right] \tag{3.37}$$

The normalization factor $M^{-1}N^{-1}$ can either be applied in the forward Hartley transform as proposed by Bracewell,[4] in the inverse Hartley transform as proposed by Watson and Poirson,[34] or in both the forward and inverse transform, as in Equation (3.37) with the advantage that the Hartley transform is still its inverse under this definition. The following equation defines a separable discrete Hartley transform[34] that is separable in analogy to the Fourier transform [Equation (3.8)]:

$$H(u,v) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x,y) \operatorname{cas}\left(2\pi\frac{ux}{M}\right) \operatorname{cas}\left(2\pi\frac{vy}{N}\right) \tag{3.38}$$

The separable Hartley transform in Equation (3.38) was proposed by Bracewell as the two-dimensional extension of the one-dimensional discrete Hartley transform.[3] However, the separable discrete Hartley transform no longer obeys the relationship to the Fourier transform in Equations (3.34) and (3.35). This implies that the convolution theorem, Equation (3.36), is not valid in two dimensions under Equation (3.38). Instead, Watson and Poirson derived the modified convolution theorem[34]

$$\begin{aligned} G(u,v) = {} & G_{1e}(u,v)G_{2e}(u,v) - G_{1o}(u,-v)G_{2o}(u,-v) \\ & + G_{1e}(u,-v)G_{2_o}(u,v) + G_{1o}(u,v)G_{2e}(u,-v) \end{aligned} \tag{3.39}$$

where the indices e and o indicate the even (cosine) part and the odd (sine) part of the discrete Hartley transform, respectively.

To view and examine a frequency spectrum, the discrete Hartley transform is an attractive alternative to the Fourier transform, since the transform is real-valued and no imaginary component or phase component needs to be considered. For the same reasons, the memory requirement is lower, and the actual transform is faster, since the cas function can be computed with a single trigonometric operation, where two such operations are needed for the complex exponential of the Fourier transform. However, when the Hartley transform is used for image filtering, the complexity of the convolution theorem offsets some of that advantage.

3.4. BIOMEDICAL EXAMPLES

Image processing in the Fourier domain is a fundamental tool not only in the process of manipulating and enhancing images but also in image formation. Most computerized imaging modalities rely on the Fourier transform at some point of the image formation process. The following examples of computed tomography and magnetic resonance image formation highlight the importance of the Fourier transform in biomedical imaging.

In 1917, J. Radon presented his work "On the determination of functions through line integrals along certain manifolds."[24] X-ray attenuation through inhomogeneous media can be described by

$$I = I_0 \exp\left(-\int_s \mu(\vec{r})d\vec{r}\right) \tag{3.40}$$

where I_0 is the incident x-ray intensity, I the x-ray intensity after the x-rays passed through the object, $\mu(\vec{r})$ the x-ray absorption at location $\vec{r}$, and integration takes place along a straight path s through the object. The argument of the exponential function is a line integral in the sense of Radon's work, and a set of line integrals along parallel paths that completely covers the object describes an x-ray projection. The set of projections at different angles are referred to as the *Radon transform*. Radon showed that this is indeed a transform, as an inverse transform exists which allows to recover the original object from the Radon transform. A relationship exists between the Radon and Fourier transforms called the *Fourier slice theorem*. It stipulates that the Fourier transform of a parallel projection of an image $\mu(x,y)$, taken at an angle θ, gives a one-dimensional slice of the two-dimensional Fourier transform of $\mu(x,y)$, subtending an angle θ with the u-axis.[18] The Fourier slice theorem is illustrated in Figure 3.13.

On the basis of the Fourier slice theorem, image reconstruction in computed tomography becomes possible by first collecting one-dimensional projections at many angles. Their one-dimensional Fourier transforms are then entered into a two-dimensional frequency-domain matrix spaceholder. Since it is practically impossible to fill all matrix elements of the frequency-domain matrix, missing elements need to be interpolated. Once the matrix is completely filled and interpolated, the object is

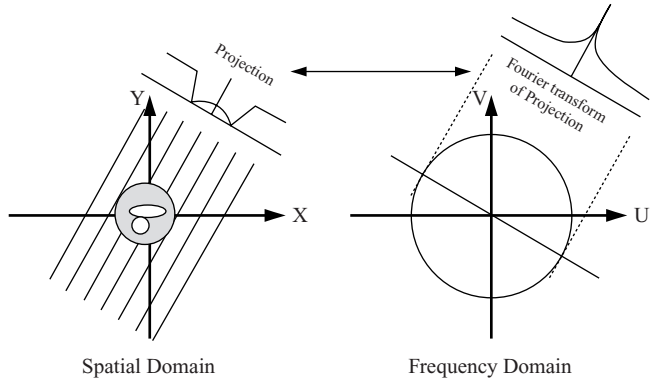


FIGURE 3.13 The Fourier slice theorem relates the one-dimensional Fourier transform of a projection to a slice through the two-dimensional Fourier transform of the object along a line through the origin at the same angle as the projection.

obtained by two-dimensional inverse fast Fourier transform of the matrix. An alternative approach for CT reconstruction is filtered backprojection. In the backprojection process, the projection is distributed over the image along the line of projection, and the contributions of the projections at different angles are added up. It can be shown that the point-spread function of the backprojection process is a $1/r$ function such that a point source blurs into an image with an intensity that drops off with the reciprocal of the distance r to the point center. A suitable filter to correct for the $1/r$ point-spread function has a frequency response of $H(\omega) = |\omega|$,[15] and a convolution-based filter kernel to achieve this frequency response was proposed by Ramachandran and Lakshminarayanan.[25] The filtered backprojection thus becomes a two-step process, where the projection is first convolved with the Ramachandran–Lakshminarayanan kernel and then backprojected over the image space for reconstruction. Since the Fourier transform of this kernel is proportional to $|\omega|$, its frequency-domain implementation is much more efficient, and the first step of the filtered backprojection becomes a frequency-domain filtering step where the projection is Fourier-transformed, the transform multiplied with $|\omega|$, and the result subjected to the inverse Fourier transform. This operation has a close relationship to computation of the first derivative:

$$\mathcal{F}\left\{\frac{df(t)}{dt}\right\} = j\omega\mathcal{F}\{f(t)\} \tag{3.41}$$

Consequently, it has a strong noise-amplifying effect. Shepp and Logan[28] therefore suggested a different filter with a frequency response $H(\omega)$:

$$H(\omega/N) = \left|\frac{1}{\pi}\sin\frac{\pi\omega}{2N}\right| \tag{3.42}$$

which has a lower amplification of high spatial frequencies than that of the Ramachandran–Lakshminarayanan filter. This behavior can be seen in Figure 3.14.

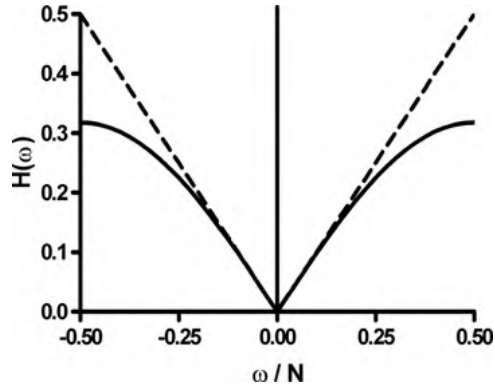


FIGURE 3.14 Frequency responses of the filters used in the filtered backprojection for computed tomography reconstruction. The dashed line represents the filter by Ramachandran and Lakshminarayanan, and the solid line represents the filter by Shepp and Logan. The Shepp–Logan filter exhibits less amplification of the higher frequencies, which reduces image noise at the expense of some sharpness.

As a result, the Shepp-Logan filter will cause the reconstructed images to have a lower noise component at the expense of some edge sharpness. Reconstruction in Fourier space and filters for reconstruction are presented in a review paper by Zubal and Wisniewski.[36]

The Fourier transform is of similar importance in the reconstruction of magnetic resonance images. Hydrogen protons possess a magnetic moment. In a strong external magnetic field, the magnetic moments assume an orientation parallel and antiparallel to the magnetic field with a small net magnetization in favor of the parallel orientation. In addition, the magnetic moment shows a small precession with an angular frequency (Larmor frequency) $\omega = \gamma B_0$, where B_0 is the external magnetic field and γ is the gyromagnetic ratio, which is a material constant. In a typical clinical scanner with $B_0 = 1.5$ T, the precession frequency is $f = \omega/2\pi = 63.87$ MHz. A radio-frequency pulse can be used to expand the precession cone and to bring the precessing moments into coherence. Once the radio-frequency pulse stops, precession coherence is lost, and the precession cone narrows as the magnetic moments return to their original orientation parallel to B_0. In the process, the protons emit a decaying radio-frequency signal with their precession frequency that is received by a suitable antenna. To obtain an image, the radio-frequency signal needs to be encoded in three dimensions. Encoding is performed by three orthogonal magnetic gradients that are superimposed over the main magnetic field B_0. These gradients change the precession frequency along the axis of the gradient. Analysis of the frequency components by means of the Fourier transform allows spatial separation of the signal components. More precisely, the first gradient is applied in the axial (z) direction during radio-frequency injection. This ensures that precession coherence is achieved in only a thin slice of the tissue. Upon echo acquisition, only this slice contributes to the radio-frequency signal. Between radio-frequency injection and echo acquisition, a brief gradient along the x-axis

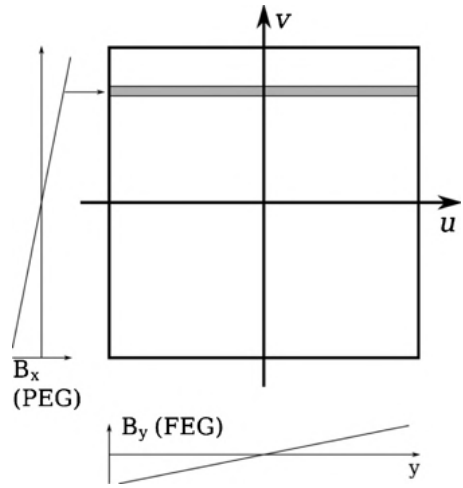


FIGURE 3.15 Filling of the *k*-space matrix in MRI. The frequency encode gradient (FEG) causes different precession frequencies along the *y*-axis, and Fourier analysis of one single echo signal allows us to fill one line of pixels in the *k*-space matrix (gray shaded box). The phase encode gradient (PEG) is a gradient along the *x*-axis that is applied briefly before echo acquisition. Phase encode gradients of different strength can be seen as vectors that position the echo line parallel to the *v*-axis. Repeated echos, taken with different PEG strengths, are used to fill the *k*-space matrix line by line. The inverse Fourier transform of the filled *k*-space matrix yields the cross-sectional image.

[B_x, called the *phase-encode gradient* (PEG)] creates a position-dependent phase shift of the precessing moments along the *x*-axis. During acquisition, the third gradient is applied, which modulates the precession frequencies along the *y*-axis [B_y, called the *frequency-encode gradient* (FEG)]. Phase shift and echo frequency are orthogonal and can be seen as a point in a two-dimensional frequency-domain matrix called the *k-space matrix*. Each echo acquisition fills one line parallel to the *u*-axis of the *k*-space matrix, and application of the *x* gradient determines the position of this line on the *v*-axis. The *k*-space matrix is filled with repeated acquisitions. An inverse Fourier transform yields the cross-sectional image (Figure 3.15).

Ultrasound imaging is not as fundamentally dependent on the Fourier transform as are CT and MRI, but spectral analysis of the echo signals is a key step in Doppler ultrasound to determine flow velocity. In one example, high transducer frequencies and harmonic analysis in combination with contrast agents provide high-quality images of the liver.[33] Since one of the primary advantages of ultrasound imaging is its near-real-time data acquisition, fast execution of the Fourier transform is particularly important. One option to accelerate the FFT further is the implementation in hardware. Hu et al.[17] propose a field-programmable gate array (FPGA). An FPGA consists of semiconductor circuits which allow the realization of flexibly programmable sequential logic operations, somewhat analogous to a graphics processor chip, which can be tuned to execute a very limited instruction set at very high speed.

The flexibility of FPGA programming allows us to implement a complete image processing chain (i.e., data collection, demodulation, Fourier transform) in one single chip with an execution time of less than 50 μs per spectrum.[17] A frequently used alternative to accelerate the FFT is to use fast integer arithmetic operations instead of the slower floating-point operations.[29,35]

In image processing, Fourier filtering is often an essential step that has, in the overall scheme, little visibility. Optical imaging methods—optical coherence tomography and scattering confocal imaging—were compared to determine their ability to measure the thickness and homogeneity of tissue-engineered sheets.[19] In a one-dimensional axial confocal scan, the flask–tissue interface and the tissue–media interface appear as prominent intensity peaks. However, the scan contains noise. A robust method to detect the peaks is the removal of both the noise component and some of the high-frequency detail of the scan itself with a strong Butterworth lowpass filter [Equation (3.18)]. In a scan of approximately 3000 data points, a second-order Butterworth filter with $\omega_0 = 10$ pixels^{-1} was used, followed by baseline removal. A sample scan before and after filtering can be seen in Figure 3.16. The same filter could be implemented with a convolution operation, but strong blurring requires a large convolution kernel, and Fourier-domain filtering is more efficient in this case. After filtering, robust peak detection can easily be implemented by finding any pixels whose n neighbors on both sides have a lower intensity.

Image restoration, that is, the application of a filter that reverses a known or partially known image degradation process, almost always calls for filtering in the frequency domain. Most predominantly, the Wiener filter is used. Out of a very large body of literature, three typical examples of Wiener filter application should be considered. Araki and Nashimoto used near-infrared light and a CCD camera to obtain projections from a human forearm.[1] Tissue is highly scattering; a point source such as a laser illuminating soft tissue creates a circular area where scattered light reemerges at the surface and the light intensity in this area has a Gaussian profile.

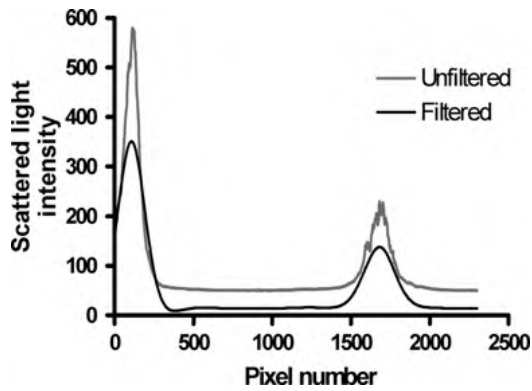


FIGURE 3.16 Noise and detail removal with a Butterworth lowpass. The filtered signal allows more accurate determination of the intensity peaks.

This Gaussian profile can be interpreted as the point-spread function, and Araki and Nashimoto corrected the image degradation by Wiener filtering. A second application is Wiener-filter-based denoising. Special radiochromic film (EBT film) exists which shows an almost linear increase in optical density with radiation exposure. For a computerized analysis of radiation exposure, EBT film is frequently digitized with a scanner. Ferreira et al. examined the influence of various scan parameters and image enhancement steps on the dose reported.[10] One step was the application of a Wiener filter for noise reduction, and the finding of this study was that the Wiener filter did not significantly change the calibration curve but reduced calibration uncertainty. The third example involves high-magnetic-strength MR scanners. In strong magnetic fields of 3 T and more, inhomogeneities of the radio-frequency field cause image nonuniformities. Hadjidemetriou et a.l[13] developed a spatially adaptive Wiener filter that restored the local cooccurrence statistics of the MR image. In this case, the highpass characteristic of the Wiener filter was used to eliminate low-frequency inhomogeneities. A less typical application of a Wiener filter was presented by Thurman and Fienup.[31] It was mentioned that the digitization process limits the frequency components to one-half of the sampling rate (or sampling distance). If the signal contains higher-frequency components, these get mirrored into the low-frequency part of the spectrum and therefore distort the spectrum, a process called *aliasing*. Thurman and Fienup present the mathematical framework to eliminate the spectral component introduced by aliasing through use of a Wiener filter. Finally, Galatsanos et al. propose an algorithm to optimize a Wiener filter if the point-spread function of the degrading imaging system is known only partially,[12] and Bonmassar et al. present a Wiener filter-based method to eliminate motion blur.[2]

The fast Hartley transform, although not as mainstream as the fast Fourier transform, has been used in medical imaging as well. Its main use is as a real-valued substitute for the complex-valued Fourier transform. For example, Paik and Fox used the fast Hartley transform as a direct substitute for the Fourier transform to study spectra of liver ultrasound images.[22] A more general review paper by Oberholzer et al. features the Hartley transform as the method of choice for frequency-domain filtering.[21] Villasenor shows that the discrete cosine transform, used universally in image compression, may not be the optimum choice in special cases.[32] These special cases include magnetic resonance imaging and positron emission tomography, which are characterized by a circular region of high frequencies where the transform coefficients are very close to zero. Instead of performing a tiled discrete cosine transform and setting high-frequency coefficients to zero (quantization; see Section 12.3), Villasenor proposes the global application of the FFT or FHT and global quantization by exploiting the fact that most high-frequency coefficients are close to zero, thus minimizing quality loss and local discontinuities. These low-valued high-frequency coefficients may also be the reason why Shyam Sunder et al. found image compression with the DHT superior in terms of image quality to DCT-based image compression in brain MR images with high bit depths,[30] while the DCT remained superior in other cases, such as angiography images.

The Fourier transform is not only useful for filtering. The coefficients of the Fourier transform can provide quantitative information about images, both in one

and multiple dimensions, as will be shown in two examples. In one example, the micromotion of the cell membrane was examined using the light-scattering properties of the cell membrane of cultured cells.[14] By applying a refinement over a method described earlier,[20] the intensity fluctuations over time of light scattered from a low-incident-angle laser beam were subjected to the Fourier transform and its frequency components were analyzed. These fluctuations exhibit a relatively low dominant frequency of 0.14 Hz, but both peak frequency and overall amplitude are lowered significantly when the cells are treated with paraformaldehyde, a toxin that cross-links the proteins in the membrane and therefore makes the membrane more rigid. In the second example, spectral properties were found to be related to the perception of images.[9] Observer ratings of aesthetic appeal and discomfort of artistic images were compared to the frequency-domain spectra of the image's luminance. Images often show a strong decay of the amplitude of the frequency components at higher frequencies. Images perceived as "uncomfortable" have unnaturally strong spectral components at spatial frequencies to which the visual system is most sensitive, a fact that the artists apparently exploited.

The Fourier transform is a relevant image processing operation in several chapters in this book. In Section 5.2, some adaptive frequency-domain filtering techniques are explained in detail. In Section 10.4, frequency-domain methods to determine fractal or self-similar properties in images are explored, and frequency-domain texture classification (Section 8.3) and shape classification methods (Section 9.5) are presented. Finally, in Section 12.3, the discrete cosine transform is introduced as a fundamental operation in image compression.

REFERENCES

1. Araki R, Nashimoto I. Near-infrared imaging in vivo (I): image restoration technique applicable to the NIR projection images. Adv Exp Med Biol 1992; 316:155–161.
2. Bonmassar G, Schwartz EL, Harvard M, Chadestown MA. Real-time restoration of images degraded by uniform motion blur infoveal active vision systems. IEEE Trans Image Process 1999; 8(12):1838–1842.
3. Bracewell RN. Aspects of the Hartley transform. Proc IEEE 1994; 82(3):381–387.
4. Bracewell RN. Discrete Hartley transform. J Opt Soc Am 1983; 73(12):1832–1835.
5. Bracewell RN. The fast Hartley transform. Proc IEEE 1984; 72(8):1010–1018.
6. Cooley JW, Lewis PAW, Welch PD. Historical notes on the fast Fourier transform. IEEE Trans Audio Electroacoust 1967; 15:76–79.
7. Cooley JW, Tukey JW. An algorithm for the machine calculation of complex Fourier series. Math Comput 1966; 19:297–301.
8. Danielson GC, Lanczos C. Some improvements in practical Fourier analysis and their applications to x-ray scattering from liquids. J Franklin Inst 1942; 233:365–380.
9. Fernandez D, Wilkins AJ. Uncomfortable images in art and nature. Perception 2008; 37(7):1098–1113.
10. Ferreira BC, Lopes MC, Capela M. Evaluation of an Epson flatbed scanner to read Gafchromic EBT films for radiation dosimetry. Phys Med Biol 2009; 54(4):1073–1085.

11. Frigo M, Johnson SG. The design and implementation of FFTW3. Proc IEEE 2005; 93(2):216–231.
12. Galatsanos NP, Mesarovic VZ, Molina R, Katsaggelos AK. Hierarchical Bayesian image restoration from partially known blurs. IEEE Trans Image Process 2000; 9(10):1784–1797.
13. Hadjidemetriou S, Studholme C, Mueller S, Weiner M, Schuff N. Restoration of MRI data for intensity non-uniformities using local high order intensity statistics. Med Image Anal 2009; 13(1):36–48.
14. Haidekker MA, Stevens HY, Frangos JA. Cell membrane fluidity changes and membrane undulations observed using a laser scattering technique. Ann Biomed Eng 2004; 32(4):531–536.
15. Härer W. [Algorithms for image reconstruction and systems theory]. In: Morneburg H, editor. [Imaging Systems for Medical Diagnostics]. Erlangen, Dermary: Publicis MCD, 1995:44–67.
16. Hartley RVL. A more symmetrical fourier analysis applied to transmission problems. Proc IRE 1942; 30(3):144–150.
17. Hu CH, Zhou Q, Shung KK. Design and implementation of high frequency ultrasound pulsed-wave Doppler using FPGA. IEEE Trans Ultrason, Ferroelectr Freq Control 2008; 55(9):2109–2111.
18. Kak AC, Slaney M. Principles of Computerized Tomographic Imaging. New York: IEEE Press, 1988. Electronic edition, 1999.
19. LaCroix JT, Xia J, Haidekker MA. A fully automated approach to quantitatively determine thickness of tissue-engineered cell sheets. Ann Biomed Eng 2009; 37:1348–1357.
20. Nishio I, Peetermans J, Tanaka T. Microscope laser light scattering spectroscopy of single biological cells. Cell Biophys 1985; 7(2):91–105.
21. Oberholzer M, Ostreicher M, Christen H, Bruhlmann M. Methods in quantitative image analysis. Histochem Cell Biol 1996; 105(5):333–355.
22. Paik CH, Fox MD. Fast Hartley transforms for image processing. IEEE Trans Med Imaging 1988; 7(2):149–153.
23. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C, 2nd ed. Cambridge, UK: Cambridge University Press, 2002.
24. Radon J. On the determination of functions through line integrals along certain manifolds. Berlin, Saxon Acad Sci 1917; 29:262–279.
25. Ramachandran GN, Lakshminarayanan AV. Three-dimensional reconstruction from radiographs and electron micrographs: application of convolution instead of Fourier-transform. Proc Natl Acad Sci U S A 1971; 68(9):2236–2240.
26. Rao KR, Yip P. Discrete Cosine Transform: Algorithms, Advantages, Applications. San Diego; CA: Academic Press, 1990.
27. Rockmore DN. The FFT: an algorithm the whole family can use. Comput Sci Eng 2000; 2(1):60–64.
28. Shepp LA, Logan BF. Reconstructng interior head tissue from x-ray transmissions. IEEE Trans Nucl Sci 1974; 21(3):228–236.
29. Sheridan P. A method to perform a fast Fourier transform with primitive image transformations. IEEE Trans Image Process 2007; 16(5):1355–1369.
30. Shyam SR, Eswaran C, Sriraam N. Medical image compression using 3-D Hartley transform. Comput Biol Med 2006; 36(9):958–973.

31. Thurman ST, Fienup JR. Wiener reconstruction of undersampled imagery. J Opt Soc Am A 2009; 26(2):283–288.
32. Villasenor JD. Alternatives to the discrete cosine transform for irreversible tomographic image compression. IEEE Trans Med Imaging 1993; 12(4):803–811.
33. Vogt W. [Value of ultrasound and Doppler sonography in chronic hepatitis and liver cirrhosis]. Praxis (Bern 1994) 2005; 94(16):639–643.
34. Watson AB, Poirson A. Separable two-dimensional discrete Hartley transform. J Opt Soc Am 1986; 3(12):2001–2004.
35. Yan Y, Su G, Wang C, Shi Q. Invertible integer FFT applied on lossless image compression. Proc IEEE Int Conf Robot, Intell Syst Signal Process 2003; 2:1219–1223.
36. Zubal IG, Wisniewski G. Understanding Fourier space and filter selection. J Nucl Cardiol 1997; 4(3):234–243.

4

THE WAVELET TRANSFORM AND WAVELET-BASED FILTERING

The wavelet transform belongs to the class of integral transforms, such as the Fourier transform. Whereas the Fourier transform uses sine and cosine functions as basis functions, the wavelet transform uses special functions with finite support, termed *wavelets*. The most fundamental difference between wavelets and the Fourier transform is the scalability of wavelets. Whereas the Fourier transform allows us to analyze the frequency components only globally, shifting and scaling operations used in the wavelet transform allow us to analyze local properties of a signal or image. The wavelet transform is therefore useful in analyzing signals and images with large discontinuities and nonperiodic signals (the Fourier transform assumes all signals to be periodic). The wavelet transform gives rise to powerful and flexible filters, to the analysis of images at different scales (multiscale analysis), and to lossy image compression methods.

To understand the wavelet transform, let us recall the definition of convolution (Section 2.3) of a time function $f(t)$ with a kernel $g(t)$:

$$(f \circledast g)(\tau) = \int_{-\infty}^{\infty} f(t)g(t-\tau)dt \tag{4.1}$$

The integral in Equation (4.1) needs to be evaluated for all possible values of τ. If the kernel has finite support [e.g., $g(t)$ is completely zero for all $t < -T$ and for all $t > T$], the integration boundaries may be finite, too. In this example, integration would take place from $\tau - T$ to $T - \tau$. For each value of τ for which Equation (4.1) is

Advanced Biomedical Image Analysis, By Mark A. Haidekker

evaluated, the integral returns one scalar value. Let us call this value $W(\tau)$. Also, let us choose a wavelet function $\psi(t)$ for the kernel $g(t)$. Wavelet functions are a specific class of functions that are explained below. With these definitions, Equation (4.1) can be rewritten as

$$W(\tau)\{f\} = \int_{-\infty}^{\infty} f(t)\ \psi(t-\tau)dt \tag{4.2}$$

We can read Equation (4.2) as computing one convolution value $W(\tau)$ of the function to be transformed, f, by evaluating the integral in Equation (4.2) at the value τ, which causes the wavelet to be centered on $f(\tau)$. Now let us introduce another parameter, s, which allows us to stretch ($s > 1$) or compress ($0 < s < 1$) the wavelet function. We now have two selectable parameters, τ and s, and Equation (4.2) extends into

$$W(s,\tau)\{f\} = \int_{-\infty}^{\infty} f(t)\ \psi\left(\frac{t-\tau}{s}\right)dt \tag{4.3}$$

With Equation (4.3) we transform $f(t)$ into a two-dimensional space (s,τ). The parameter τ selects the focus where the wavelet function is centered on $f(t)$, and the parameter s determines the sharpness of the focus, since for small s, only small sections of f are included in the integration, and for large s, longer sections of f are included in the integration. With an additional normalization factor, we arrive at the definition of the one-dimensional, continuous wavelet transform $W(s,\tau)\{f\}$ of a function $f(t)$:

$$W(s,\tau)\{f\} = \frac{1}{c_\psi}|s|^{-1/2}\int_{-\infty}^{\infty} f(t)\ \psi\left(\frac{t-\tau}{s}\right)dt \tag{4.4}$$

where $\psi(t)$ is the wavelet function, s and τ are the scaling and shift parameters, respectively, and c_ψ is defined as

$$c_\psi = 2\pi\int_{-\infty}^{\infty}\frac{|\Psi(\omega)|^2}{|\omega|}d\omega \tag{4.5}$$

with $\Psi(\omega)$ being the Fourier transform of $\psi(t)$. For a function $\psi(t)$ to be a wavelet, it is necessary that c_ψ be finite (i.e., $0 < c_\psi < \infty$) and that the mean value of $\psi(t)$ vanishes [i.e., $\Psi(0) = 0$]. Furthermore, $\psi(t)$ must have finite support, that is, beyond a certain value of $\psi(t) = 0$ for $t < -t_1$ and for $t > t_2$ ($0 \le t_1, t_2 < \infty$). The simplest example of a wavelet function is the Haar wavelet (Figure 4.1), defined by

$$\psi(t) = \begin{cases} 1 & \text{for } 0 \le t < 0.5 \\ -1 & \text{for } 0.5 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The wavelet properties, namely, the zero mean value and the finite support (0 to 1), can be seen immediately. The finite value for c_ψ can be proven by computing the Fourier transform of the Haar wavelet and applying Equation (4.5). For the Haar wavelet, $c_\psi = 2 \cdot \ln 2$. Intuitively, application of the Haar wavelet in the wavelet transform [Equation (4.4)] can be interpreted as the computation of the finite difference of two range averages: from 0 to 0.5 and from 0.5 to 1. This intuitive interpretation of the wavelet transform as the computation of a weighted finite difference will become important when we introduce the bandpass and lowpass components of the wavelet analysis filter.

In wavelet analysis, the unscaled wavelet $\psi(t)$ is referred to as the *mother wavelet*, and the scaled and translated wavelets $\psi_{s,\tau}(t)$ that are obtained from the mother wavelet,

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \tag{4.7}$$

are called the *wavelet basis*. From the wavelet properties described above, it can be concluded that the mother wavelet has a frequency spectrum that exhibits bandpasslike characteristics, rapidly dropping off toward $\omega = 0$ and gradually dropping off toward high frequencies (Figure 4.1B). Translation in time (parameter τ) only changes the phase of the Fourier transform, but changing the scale s changes the bandwidth of the Fourier transform in the opposite direction:

$$\mathcal{F}\{f(t)\} = F(\omega) \rightarrow \mathcal{F}\{f(st)\} = \frac{1}{|s|}F\left(\frac{\omega}{s}\right) \tag{4.8}$$

With suitable scaling steps (e.g., doubling s for each wavelet), the wavelet basis acts like a bandpass filter bank, each expanded wavelet providing a lower (and compressed) frequency band.

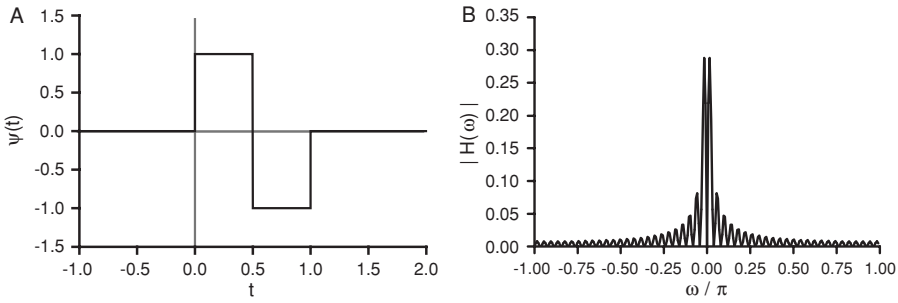


FIGURE 4.1 The Haar wavelet (A) and the magnitude of its Fourier transform (B).

4.1. ONE-DIMENSIONAL DISCRETE WAVELET TRANSFORM

For applications in signal and image processing, that is, applications on discrete data, a discrete wavelet transform must exist. For a discrete wavelet transform it is not necessary to translate and scale the mother wavelet continuously. Rather, it is possible to rewrite Equation (4.7) to reflect discrete translation and scaling steps j and k:

$$\psi_{j,k}(t) = \frac{1}{\sqrt{s^j}}\psi\left(\frac{t - k\tau s^j}{s^j}\right) \tag{4.9}$$

where s and τ are now constant scaling and translating increments. Typically, $s = 2$ and $\tau = 1$ are chosen in discrete wavelet implementations. The localization of discrete wavelets as a function of the scaling and translating parameters j and k is shown in Figure 4.2, where each circle represents the location of one wavelet. Figure 4.2 not only shows the doubling of the scale with increasing j, but also the doubling of the time step size at larger scales.

The discrete wavelet transform is realized by using the principles of subband coding. The discrete convolution of the signal with a wavelet function can be interpreted as bandpass filtering of a signal $x(k)$ with discrete filter coefficients $g(r)$ according to

$$y(k) = \sum_{r=-\infty}^{\infty} g(r)x(r - k) \tag{4.10}$$

Equation (4.10) represents the discrete convolution, which was introduced in Section 2.3. The discrete filter coefficients $g(r)$ are related to the wavelet function ψ and are responsible for the matching frequency behavior between the discrete filter [Equation (4.10)] and the continuous filter [Equation (4.4)]. In the context of digital signal processing, the filter coefficients $g(r)$ constitute the impulse response function of the filter, and a filter with a finite number of filter coefficients is referred to as a

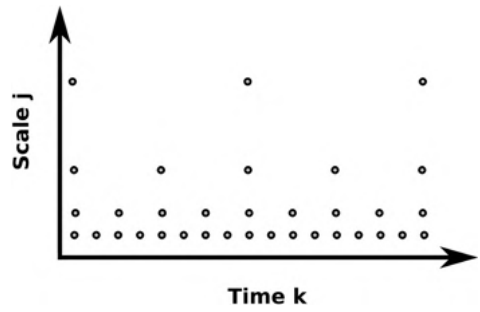


FIGURE 4.2 Localization of discrete wavelets as a function of the translation parameter k and the scaling parameter j.

finite impulse response filter. To complete the subband coding filter, a function $\varphi_{s,\tau}(t)$ needs to be defined that is complementary to $\psi_{s,\tau}(t)$, and filtering of the signal $f(t)$ with the function $\varphi_{s,\tau}(t)$ leads to a lowpass filtered output. In the same manner that the discrete wavelet filter uses the filter coefficients $g(r)$, the discrete filter with the filter function $\varphi_{s,\tau}(t)$ uses the filter coefficients $h(r)$. The two sets of filter coefficients $g(r)$ and $h(r)$ are related as

$$g(L-n-1) = (-1)^n h(n) \tag{4.11}$$

where L is the total number of filter coefficients and n runs from 0 to $L-1$. The function $\varphi(t)$ is called a *scaling function*, and the name is derived from a special property, namely, that $\varphi(t)$ obeys the scaling equation:

$$\varphi(t) = \sqrt{2}\sum_{k\epsilon Z} h_k \varphi(2t-k) \tag{4.12}$$

In addition, being a lowpass filter, any scaling function $\varphi(t)$ cannot have a vanishing zeroth moment; more specifically,

$$\int_{-\infty}^{\infty} \varphi(t)dt = 1 \tag{4.13}$$

which is contrary to the requirement for a wavelet function $\psi(t)$. The frequency response $\Phi(\omega)$ of the finite impulse response filter defined by the filter coefficients $h(k)$ is

$$\Phi(\omega) = \sum_{k=-\infty}^{\infty} h(k)e^{-j\omega k} \tag{4.14}$$

where $\Phi(0) \neq 0$ for a lowpass filter. While each wavelet function–scaling function pair $\psi(t)$ and $\varphi(t)$ have discrete filter coefficients, $g(k)$ and $h(k)$, respectively, the relationship between the continuous functions and the discrete filter coefficients is not trivial. The derivation of the equations that lead to the design of the filter parameters $h(k)$ goes beyond the scope of this book. A detailed description of the mathematical foundations and necessary steps to design wavelets and to derive the filter coefficients $h(k)$ can be found in the book by Burrus et al.[4] Instead, the filter parameters $h(k)$ for several widely used wavelets are provided in Table 4.1. Note that $g(k)$ can be computed from $h(k)$ using Equation (4.11). The simplest case is the Haar wavelet with the wavelet parameters $g(0) = 1/\sqrt{2}$ and $g(1) = -1/\sqrt{2}$. It can be seen that the scaling filter is characterized by $h(0) = h(1) = 1/\sqrt{2}$. This is a simple averaging lowpass filter, whereas the wavelet filter is a first-order finite-difference filter. Ingrid Daubechies has pioneered the wavelet transform,[9,10] and the Daubechies series of filter coefficients is listed for filter orders 2 through 10 in her book *Ten Lectures on Wavelets*.[10] The filter functions of higher-order filters are more regular, smoother, and

TABLE 4.1 Filter Parameters for Several Widely Used Wavelets

Wavelet	Order N	Vanishing Moments	Parameters h_k
Haar	1	1	$h_0 = 1/\sqrt{2}; h_1 = 1/\sqrt{2}$
Daubechies-4	2	2	$h_0 = (1+\sqrt{3})/(4\sqrt{2})$ $h_1 = (3+\sqrt{3})/(4\sqrt{2})$ $h_2 = (3-\sqrt{3})/(4\sqrt{2})$ $h_3 = (1-\sqrt{3})/(4\sqrt{2})$
Daubechies-8	4	4	$h_0 = 0.2303778133089$ $h_1 = 0.71484657055292$ $h_2 = 0.63088076792986$ $h_3 = -0.02798376941686$ $h_4 = -0.18703481171909$ $h_5 = 0.03084138183556$ $h_6 = 0.03288301166689$ $h_7 = -0.01059740178507$
Coiflet-6	2	2	$h_0 = -0.07273261951285$ $h_1 = 0.33789766245781$ $h_2 = 0.85257202021226$ $h_3 = 0.38486484686420$ $h_4 = -0.07273261951285$ $h_5 = -0.01565572813546$
Coiflet-12	4	4	$h_0 = 0.016387336463$ $h_1 = -0.041464936781$ $h_2 = -0.067372554722$ $h_3 = 0.386110066823$ $h_4 = 0.812723635449$ $h_5 = 0.417005184423$ $h_6 = -0.076488599078$ $h_7 = -0.059434418646$ $h_8 = 0.023680171946$ $h_9 = 0.005611434819$ $h_{10} = -0.001823208870$ $h_{11} = -0.000720549446$

more often differentiable, as can be seen in Figure 4.3. This smoothness is represented by the number of vanishing moments of the wavelet. The kth moment μ_k of a wavelet is defined as

$$\mu_k = \int_{-\infty}^{\infty} t^k \psi(t) dt \tag{4.15}$$

and vanishing moments are moments where $\mu_k = 0$ for $k = 0, 1, 2, \ldots$. For example, the Haar wavelet has only one vanishing moment, $\mu_0 = 0$, which is the minimum requirement for a wavelet, as stated initially. A smooth wavelet has a smooth frequency response, which is generally a desired property.

FIGURE 4.3 Some wavelets (black) and their corresponding scaling functions (gray) of the Daubechies family.

At this point we have assembled all necessary tools to realize the discrete wavelet transform. The key elements are the discrete convolutions [Equation (4.10)] with finite impulse response filters characterized by the filter coefficients $g(k)$ and $h(k)$. With the scaling constant $s = 2$, the bandwidth of the filtered signal is half of the bandwidth of the original signal. For this reason, the Nyquist sampling theorem is still satisfied if the number of discrete samples in the filter output is reduced by a factor of 2 (by omitting every other sample). This step is called *subsampling* by a factor of 2. The elements of one subband filter stage are shown in Figure 4.4. Each subband

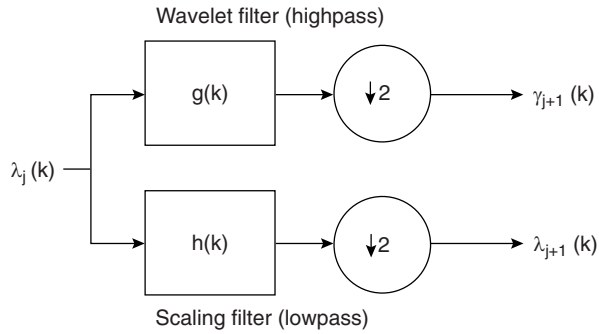


FIGURE 4.4 Sketch of one subband filter stage. The input signal $\lambda_j\,(k)$ is convolved with the highpass filter function $g(k)$ to provide the high-detail content $\gamma_{j+1}\,(k)$ and with the lowpass filter function $h(k)$ to provide the low-detail content $\lambda_{j+1}\,(k)$. The latter can be used as input to the next filter stage. The operations labeled with $\downarrow 2$ indicate subsampling by a factor of 2.

filter splits the signal into a lowpass and a complementary highpass component (both subsampled by 2). This filter pair in Figure 4.4 is known as a *quadrature mirror filter pair* in digital signal processing.

The wavelet filter bank can be implemented using a recursive scheme. Since the subsampled signal has twice the sampling interval of the input signal, a scaling factor $s = 2$ is inherent. Consequently, the lowpass-filtered component can be used as the input to the next (identical) filter stage, and multiple subband filter stages can be chained recursively to form a multiscale analysis filter, as shown in Figure 4.5. The output values of such a filter bank are called the *wavelet coefficients* of the signal $f(k)$.

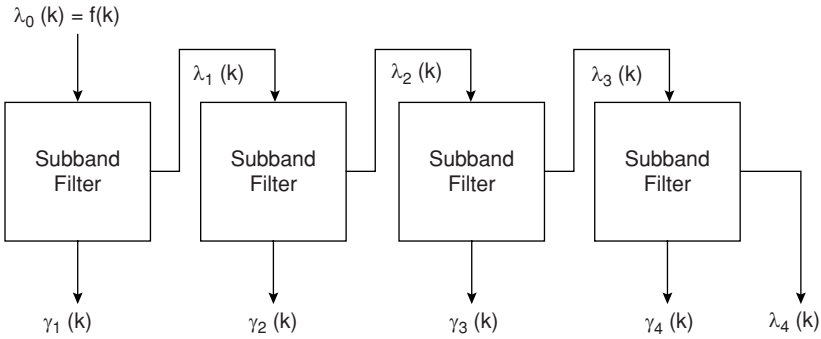


FIGURE 4.5 Multiscale analysis wavelet filter composed of four identical subband filter stages (shown in Figure 4.4). The lowpass output of each filter stage is used as input signal to the next filter stage. The wavelet filter outputs $\gamma_1(k)$, $\gamma_2(k)$, $\gamma_3(k)$, and $\gamma_4(k)$ contain successively less detail, and each signal has half the number of points of the preceding signal. The filter chain is ended by adding to the outputs the lowpass signal $\lambda_4(k)$, as the component with the lowest detail.

The two components of the subband filter (Figure 4.4) are described as

$$\lambda_{j+1}(k) = \sum_{m=0}^{N-1} h(m-2k)\lambda_j(m) \tag{4.16}$$
$$\gamma_{j+1}(k) = \sum_{m=0}^{N-1} g(m-2k)\gamma_j(m)$$

Each sequence of output coefficients, λ_{j+1} and γ_{j+1}, contains half the number of data values of the input sequence λ_j. The pyramidal decomposition scheme as laid out in Figure 4.5 with the subband filter in Equation (4.16) is explained in more detail in Figure 4.6. Starting with the initial signal (sequence of values f_k), the subband filter is applied. The result is an interleaved sequence of scaling and wavelet coefficients. These need to be rearranged (operation **R**) to separate the scaling coefficients from the wavelet coefficients. Only the scaling coefficients from one filter stage are propagated to the next filter stage, followed in turn by rearrangement. From stage to stage, therefore, a shorter sequence of wavelet coefficients with less detail is provided, indicated by a darker shade of gray. Each sequence of lower detail has half the length of the higher-detail sequence. For this reason, the input sequence must have a power-of-2 length.

The discrete wavelet transform is indeed a *transform* in the sense that it allows the exact reconstruction of the original data from the filter output. Equation

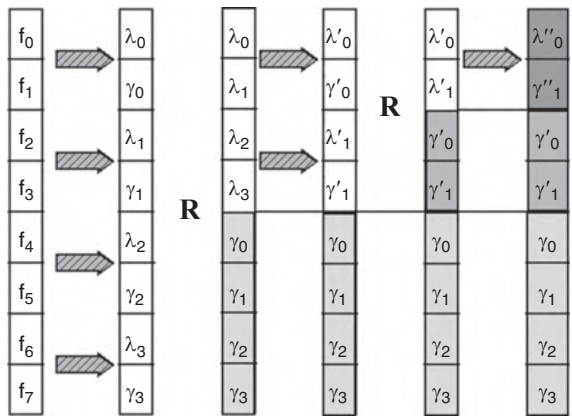


FIGURE 4.6 Schematic representation of the pyramidal decomposition scheme performed by the filter in Figure 4.5 in the example of the Haar wavelet (arrows). Data pairs of the initial data sequence f_k are subjected to the filter in Equation (4.16), resulting in pairs of scaling and wavelet coefficients, λ_k and γ_k. Rearranging the coefficients (**R**) provides the scaling coefficients λ_k followed by the wavelet coefficients γ_k. In the next step, the reduced number of scaling coefficients is again subjected to the subband filter and rearranged. After each filter step, a new sequence of coefficients with less detail (darker gray shade) is available.

(4.17) describes the reconstruction process in an manner analogous to Equation (4.16), that is, by using the recursive scheme in reverse. By reversing the signal flow through the filter in Figure 4.4, the lowpass and highpass components of each stage, $\lambda_{j+1}(k)$ and $\gamma_{j+1}(k)$, respectively, are combined until the original signal is restored for $j = 0$:

$$\lambda_j(k) = \sum_{m=0}^{N-1} h(2k - m)\lambda_{j+1}(m) + \sum_{m=0}^{N-1} g(2k - m)\lambda_{j+1}(m) \tag{4.17}$$

4.2. TWO-DIMENSIONAL DISCRETE WAVELET TRANSFORM

Up to this point, we have covered the one-dimensional discrete wavelet transform. In image processing, a multidimensional wavelet transform is desired. Fortunately, the wavelet transform is a linear operation. Therefore, analogous to the Fourier transform, the two-dimensional wavelet transform can be performed by first computing the row-by-row one-dimensional wavelet transform in the horizontal direction, followed by the column-by-column one-dimensional wavelet transform in the vertical direction. This principle can be extended toward any number of dimensions. As a consequence of the decomposition scheme, however, each subband filter generates four image regions with half of the side length. By convention, the image regions are arranged as depicted in Figure 4.7, and regions with reduced detail in either the x or y direction (L/L, L/H, H/L) are subdivided further in the subsequent filter stage (dashed lines).

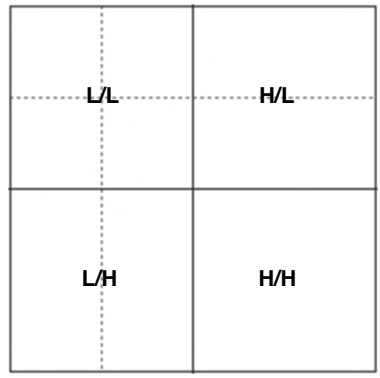


FIGURE 4.7 Output of a two-dimensional subband decomposition stage. The image is subdivided into four regions: a region with high detail (H/H), a region with high detail in the x-direction (H/L) but low detail in the y-direction, a region with high detail in the y-direction (L/H), and a fourth region with low detail (L/L).

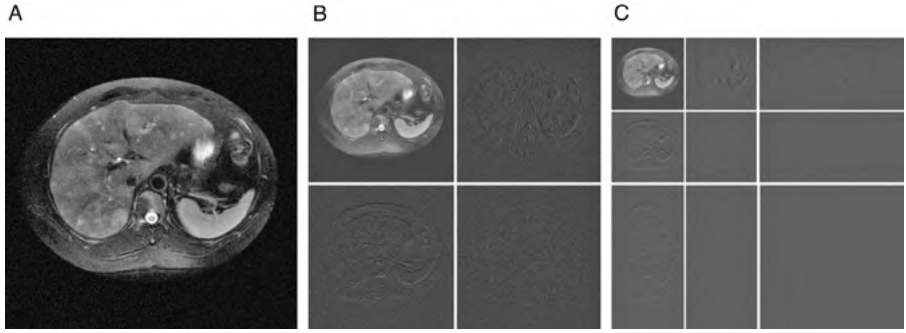


FIGURE 4.8 Example of the pyramidal decomposition of an image. The original image (A) is a noisy MR slice of the abdomen. The first-stage decomposition is shown in (B), where a subdivision into the four regions shown in Figure 4.7 has taken place. In subsequent decomposition steps (C), all regions except region H/H are subdivided further. Note that for visualization purposes, regions H/L, H/H, and L/H have been contrast-enhanced and white divider lines have been added.

Figure 4.8 shows two steps in the decomposition of a medical image, in this example a MRI slice of the abdomen with artificially added Gaussian noise. In the first decomposition step, the image is subdivided into four regions as described in Figure 4.7. In the next subdivision step, regions H/L and L/H are subdivided into two regions, and region L/L is subdivided into four. Further subdivisions of region L/L are possible in subsequent decomposition steps. From this image, the potential of the wavelet decomposition for filtering purposes becomes clear. Region H/H contains predominantly noise, whereas region L/L appears to contain the major parts of the image information. Wavelet-based filter functions are covered in Section 4.3. In addition, the wavelet transform gives rise to lossy image compression: It would appear as if region L/L in Figure 4.8B represents the original image quite well, although it contains only one-fourth of the image area. In fact, wavelet image compression is now being used widely. Wavelet-based image compression is covered in detail in Section 12.3.

Computer Implementation of the Two-Dimensional Wavelet Transform We now focus on implementation of the discrete wavelet transform. For this purpose, an algorithmic representation of Equations (4.16) and (4.17) needs to be formulated. A suitable approach is to implement a one-dimensional subband coding filter and use this filter for higher-level decomposition functions. The convolution described in Equation (4.16) can be viewed as a matrix multiplication of the data vector $\mathbf{x}$ with a convolution matrix $\mathbf{A}$, resulting in the output data vector $\mathbf{y}$, where $\mathbf{x}$ corresponds to $\lambda(m)$ in Equation (4.16) and $\mathbf{y}$ corresponds to the vector containing the next stage's $\lambda(k)$ and $\gamma(k)$:

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{4.18}$$

The matrix **A** contains, in alternating rows, the parameters of the scaling and wavelet filters:

$$\mathbf{A} = \begin{bmatrix}
g_0 & g_1 & g_2 & g_3 & \cdots & g_{l-1} & 0 & 0 & \cdots & 0 & \\
h_0 & h_1 & h_2 & h_3 & \cdots & h_{l-1} & 0 & 0 & \cdots & 0 & \\
0 & 0 & g_0 & g_1 & g_2 & g_3 & \cdots & & & & \\
0 & 0 & h_0 & h_1 & h_2 & h_3 & \cdots & & & & \\
0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & \cdots & & \\
0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & \cdots & & \\
\cdots & & & & & & & & & & \\
0 & 0 & 0 & 0 & \cdots & g_0 & g_1 & g_2 & g_3 & \cdots & g_{l-1} \\
0 & 0 & 0 & 0 & \cdots & h_0 & h_1 & h_2 & h_3 & \cdots & h_{l-1} \\
\cdots & & & & & & & & & & \\
g_2 & g_3 & \cdots & g_{l-1} & 0 & 0 & 0 & 0 & \cdots & g_0 & g_1 \\
h_2 & h_3 & \cdots & h_{l-1} & 0 & 0 & 0 & 0 & \cdots & h_0 & h_1
\end{bmatrix} \tag{4.19}$$

Note that the matrix coefficients start to wrap around in the last lines, depending on the number of wavelet parameters. A possible implementation is provided in Algorithm 4.1. By using two different indices for input and output data, it becomes

```
set s=N/2;                          // Start index of the gamma coefficients in output array
allocate ydata[N];                  // Provide memory for the output data vector

for (i=0 while i<N increment i=i+1) do        // loop over input data
   ydata[i]=0;                               // Reset summation accumulators
endfor;

j=0;                                         // Index into output data array
for (i=0 while i<N increment i=i+2) do        // loop over input data
   for (k=0 while k<L increment k=k+1) do     // convolution loop
      didx = (i+k) mod N;                    // index into input data with wraparound
      ydata[j] = ydata[i] + G[k]*xdata[didx]; // scaling filter contribution
      ydata[j+s] = ydata[i+s] + H[k]*xdata[didx];// wavelet filter contribution
   endfor;
   j=j+1;                                    // Update position in output array
endfor;
```

Algorithm 4.1 Wavelet subband filter. Pseudocode for one stage of the subband decomposition filter described in Equation (4.16). The input data vector is stored in `xdata[]`, and the output will be returned in `ydata[]`. The length of both data vectors is `N`. It is assumed that the wavelet filter parameters `G[k]` and the scale filter parameters `H[k]` are provided prior to using this algorithm. The number of parameters is `L`. For this algorithm, `N` must be an even number.

possible to perform the convolution step and the rearrangement step (see Figure 4.6) in the same loop, and the output vector will contain the $\lambda(k)$ in the lower half and the $\gamma(k)$ in the upper half. From Algorithm 4.1 it is straightforward to derive the subband decoding filter that performs one step of the inverse wavelet transform. Algorithm 4.2 performs one such step according to Equation (4.17). The input data vector must be organized like the output vector of Algorithm 4.1: that is, the $\lambda(k)$ reside in the lower half and the $\gamma(k)$ reside in the upper half. The output vector is one contiguous data vector. Algorithms 4.1 and 4.2 build on a table of filter coefficients. Since the parameters $g(k)$ and $h(k)$ are mutually dependent on each other through Equation (4.11), it is generally sufficient to store one set of filter coefficients (usually, the scale filter coefficients) and precede the wavelet transform by a preparatory step, Algorithm 4.3, to populate both filter coefficient arrays.

At this point, these algorithms can be combined to form a pyramidal wavelet decomposition and reconstruction. We begin by providing a one-dimensional decomposition, because it helps us to understand the two-dimensional decomposition. In Algorithm 4.4, a one-dimensional discrete wavelet transform is performed building on Algorithm 4.3, here referred to as `dwt_prep`, and Algorithm 4.1, referred to as `dwt_decompose`. With Algorithm 4.4 performing a full one-dimensional

```
set s=N/2;                              // Start index of the gamma coefficients in intput array

for (i=0 while i<N increment i=i+1) do   // loop over input data
   ydata[i]=0;                          // Reset summation accumulators
endfor;

j=0;                                    // Index into output data array
for (i=0 while i<N increment i=i+2) do   // loop over input data
   for (k=0 while k<L increment k=k+1) do // convolution loop
     didx = (i+k) mod N;                // index into output data with
                                        wraparound
     ydata[didx] = ydata[didx] + H[k]*xdata[j];     // scale filter contribution
     ydata[didx] = ydata[didx] + G[k]*xdata[j+s]; // wavelet filter
                                                    contribution
   endfor;
   j=j+1;                               // Update position in input array
endfor;
```

Algorithm 4.2 Wavelet reconstruction subband filter. Pseudocode for one stage of the subband reconstruction filter described in Equation (4.17). The input data vector is stored in `xdata[]` and must contain the $\lambda(k)$ in the lower half and the $\gamma(k)$ in the upper half. The output will be returned in `ydata[]`. The length of both data vectors is `N`. It is assumed that the wavelet filter parameters `G[k]` and the scale filter parameters `H[k]` are provided to this algorithm in a fashion identical to Algorithm 4.1, with the number of parameters being `L`. For this algorithm, `N` must be an even number. The calling function is responsible for providing the `ydata` array.

```
set L=4;                          // Daubechies-4 has four parameters
allocate G[L], H[L];              // Provide memory for parameters G and H

H[0] = 0.482962913144534;         // These can be obtained from the literature
H[1] = 0.836516303737808;         // In some cases, these are the solutions of
                                  closed terms such as
H[2] = 0.836516303737808;         // (1 +/− cos(PI/3) +/− sin(PI/3))/(2*sqrt(2))
H[3] = -0.12940952255126;

// Now calculate the highpass filter form the lowpass filter above

set sgn = -1;                     // This becomes (−1)^k, the alternating sign
for (k=0 while k<L increment k=k+1)
      G[L-1-k]=sgn*H[k];
      sgn = -sgn;
endfor;
```

Algorithm 4.3 Wavelet filter preparation. Pseudocode for the preparation of the filter parameter arrays `G[k]` and `H[k]` for one specific example, the Daubechies-4 wavelet (see Table 4.1). This algorithm computes the wavelet filter parameters $g(k)$ from the scale filter parameters $h(k)$ and sets the filter length `L` that is required by Algorithms 4.1 and 4.2.

pyramidal decomposition, it is now possible to extend the wavelet transform toward two-dimensional data (i.e., images). Since the discrete wavelet transform is a linear operation, a two-dimensional discrete wavelet transform can be implemented as a one-dimensional wavelet transform in the x-direction followed by a one-dimensional wavelet transform in the y-direction. This strategy is reflected in Algorithm 4.5, which builds on the one-dimensional wavelet transform in Algorithm 4.4, here referred to as `dwt_1d`. When implementing Algorithms 4.4 and 4.5, care should be taken that `dwt_prep`, the preparation of the filter parameters, is executed only once, that is, in Algorithm 4.5 and not in Algorithm 4.4.

Algorithms 4.1 through 4.5 combined perform a two-dimensional wavelet transform on an image with a freely selectable wavelet and a selectable number of decomposition levels. The inverse wavelet transform is implemented in a straightforward manner by substituting the call to the wavelet decomposition (`dwt_decompose`, Algorithm 4.1) by a call to Algorithm 4.2.

4.3. WAVELET-BASED FILTERING

We discussed the decomposition of a signal (or image) into a lowpass- and highpass-filtered component. Wavelet-based filters can be envisioned as algorithms where wavelet decomposition takes place, followed by an attenuation of either the

```
call dwt_prep;                    // Algorithm 4.3, prepare L, H[k], and G[k]
allocate ydata[N];                // Provide memory for the output data vector

i=N; level=0;

// Outer decomposition loop. Drop out when there are fewer than four points to decompose
// or a preset number of decompositions has been reached.

while ((i>=4) and (level<maxlevel)) do
     call dwt_decompose (xdata, ydata, i);      // one lvl decomp. of xdata into
                                                   ydata,
                                                   // only the first i points
                                                   (Algorithm 4.1)
     i = i/2;                                   // Next decomposition limited to
                                                   lower half,
     level=level+1;                             // we proceed from fine to
                                                   coarse detail

     for (j=0 while j<N increment j=j+1)   // Finally, copy ydata back into xdata
           xdata[j] = ydata[j];            // for the next iteration
     endfor;

endwhile;
delete (ydata);                            // Discard the ydata array
```

Algorithm 4.4 Discrete wavelet transform. Pseudocode to perform a one-dimensional wavelet transform on the input data array `xdata[]` with a total length of `N` elements. The output vector is also in `xdata[]`. With the variable `maxlevel`, the number of decomposition steps can be restricted.

lowpass-filtered component or the highpass-filtered component. An inverse wavelet transform then restores the filtered image. The underlying principle can be seen in Figure 4.8. The H/H area of the decomposed image contains mostly noise, whereas the L/L area contains most of the lower-detail image information. A smoothing or noise-reducing filter would attenuate the wavelet coefficients at higher scales (i.e., wavelet coefficients containing image detail). Sharpening filters or background removal filters would attenuate the lowpass-filtered wavelet coefficients at a high level of decomposition, in analogy to lowpass filtering in the Fourier domain. Although some work has been done in detrending signals by using the wavelet transform, wavelet-based background removal has not entered the mainstream of image processing. In fact, the most widely used wavelet-based filtering technique is the noise-reducing filter analogous to the Fourier-domain lowpass filter. Contrary to the Fourier lowpass filter, however, wavelet-based noise-reduction filters do not blur the image edges significantly. The removal of noise from images using wavelet techniques is far superior to that using Fourier techniques, and the term *denoising* has become common in conjunction with wavelet filtering techniques to indicate almost complete noise removal, as opposed

```
allocate IM_DWT(xmax,ymax);     // Prepare output image
call dwt_prep;                  // Algorithm 4.3, prepare L, H[k], and G[k]
allocate xdata[N];              // Provide memory for the 1D data vector, first in the
                                X direction
set maxlevel=5;                 // May restrict the number of decomposition levels
                                // or use a very large number to allow full decomposition

for (y=0 while y<ymax increment y=y+1)      // Run over all rows of the image

   for (x=0 while x<xmax increment x=x+1)  // Copy the image row into xdata
      xdata[x] = IM(x,y);                  // in preparation for the 1D DWT
   endfor;

   call dwt_1d (xdata,xmax);               // Perform the 1D DWT

   for (x=0 while x<xmax increment x=x+1)  // Copy the row-transformed
                                           data into the output
      IM_DWT(x,y) = xdata[x];              // image which is used as placeholder

   endfor;

endfor;                                    // Row-by-row transform finished

delete (xdata);          // Discard the old xdata array because the y dimension may
                         be different
allocate xdata[ymax];    // and obtain memory for the column transform

for (x=0 while x<xmax increment x=x+1)     // Run over all columns of the image

   for (y=0 while y<ymax increment y=y+1)  // Copy the row-transformed
                                            data into xdata
      xdata[y] = IM_DWT(x,y);              // in preparation for the orthogonal
                                            1D DWT
   endfor;

   call dwt_1d (xdata,ymax);               // Perform the 1D DWT on the
                                            column data

   for (y=0 while y<ymax increment y=y+1)  // Copy the final transformed
                                           data into ...
      IM_DWT(x,y) = xdata[y];              // ... output image as final result
   endfor;

endfor;                                    // column-by-column transform finished

delete (xdata);                            // Discard the old xdata array
```

Algorithm 4.5 Discrete wavelet transform in two dimensions. Pseudocode to perform a two-dimensional wavelet transform on the input image `IM(x,y)`. It is assumed that the image dimensions `xmax` and `ymax` are powers of 2. The data are stored in the output image `IM_DWT(x,y)` with the same dimensions as `IM(x,y)` and which, upon exit, holds the discrete wavelet transform of `IM`.

to noise attenuation or noise reduction performed using Fourier techniques with lowpass filters or convolution filters.

4.3.1. Wavelet-Based Denoising

Denoising filters involve more complex operations than simply eliminating the wavelet coefficients in the H/H area before reconstruction. The effect is demonstrated in Figure 4.9, where high-frequency areas L/H, H/L, and H/H of the first-level decomposition were set to zero before reconstruction. The noise is hardly attenuated, but resembles more closely the output of a Gaussian convolution filter.

A robust method of wavelet-based denoising was proposed by Donoho.[12,13] Here, the wavelet coefficients are subjected to a hard or soft threshold as shown in Figure 4.10. This operation is termed *wavelet coefficient shrinking*. The result of applying a wavelet shrinking filter to the image in Figure 4.9A is shown in Figure 4.11. It can be seen that noise suppression is considerably stronger than in Figure 4.9B, whereas edge blurring, associated with a Gaussian blurring operation (Figure 4.9C), is less pronounced. As a generalization, hard thresholding leads to a smaller mean-squared error between the ideal noise-free image and the noisy image, whereas soft thresholding has a lower tendency to cause spurious oscillations in the reconstructed image. These oscillations may be related to Gibbs phenomena[7] and can be suppressed by the following averaging technique: Multiple denoising operations are performed where the original image is cyclically shifted by one or multiple pixels for each wavelet denoising operation. After reconstruction, the image is shifted back in the same manner. Since those oscillations occur where image discontinuities and wavelet discontinuities coincide, the average of the images filtered by the shift–denoise–unshift operation will exhibit attenuated oscillations.

Donoho and Johnstone propose $\tau = \sqrt{2 \log n}$ as a universal threshold,[13] but the optimum global threshold τ depends on the noise component, more specifically the

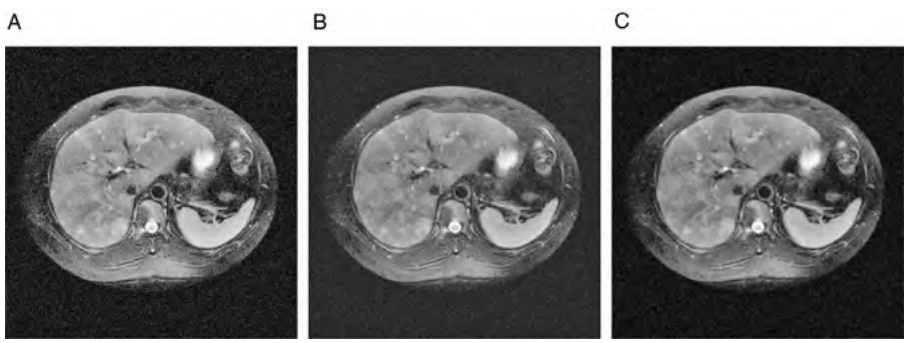


FIGURE 4.9 A first approach at attenuating noise in an image using the wavelet transform. The original image in Figure 4.8, a MRI slice with artificially added Gaussian noise (A), was decomposed as shown in Figure 4.8B, and the L/H, H/L, and H/H regions set to zero before reconstruction (B). The resulting image shows a considerable noise component, but on a coarser scale, very similar to the result of Gaussian blurring (C).

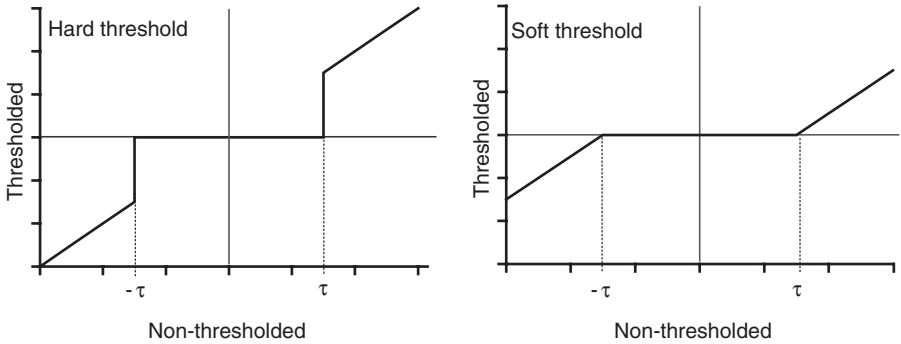


FIGURE 4.10 Thresholding schemes for wavelet coefficient shrinking. With a hard threshold, every wavelet coefficient γ is either clamped to zero if γ falls below the threshold (hard thresholding) or all coefficients are set to zero if $\gamma \leq \tau$ and all coefficients with $\gamma > \tau$ are reduced by the threshold τ (soft thresholding).

noise standard deviation σ. If σ is known, the following equation provides a better threshold value:

$$\tau = \Upsilon \sigma \sqrt{2 \log n} \tag{4.20}$$

Here n is the total number of wavelet coefficients, σ the standard deviation of the noise component, and Υ an empirical constant that determines filter strength. The effect of the threshold on the filtered image quality can be seen in Figure 4.12, which shows the root-mean-squared error (RMSE) of the wavelet filter output image and

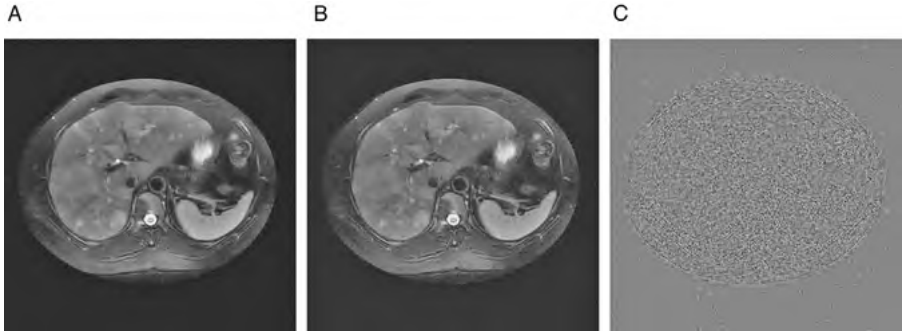


FIGURE 4.11 Denoising of the image in Figure 4.9A using the Donoho algorithm. Soft thresholding (A) leaves some visible noise, especially in the background region. Hard thresholding, although having approximately the same root-mean-squared error (B), shows less residual noise but some oscillation artifacts. These oscillations are emphasized in image C, which is the difference between filtered image B and the original, noise-free image. Image C has been slightly contrast-enhanced for better visualization.

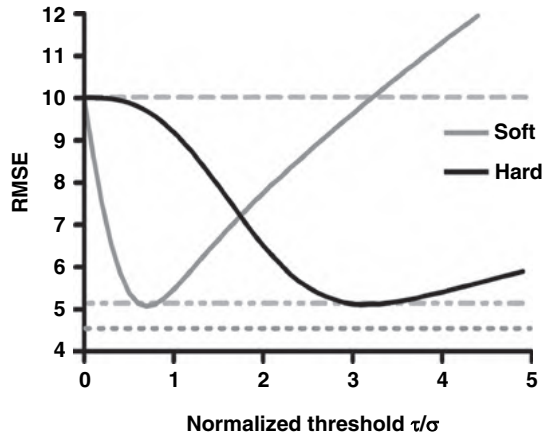


FIGURE 4.12 Root-mean-squared error (RMSE) between the denoised image and the original, noise-free image as a function of the threshold. The dashed lines indicate, from top to bottom, the RMSE of the unfiltered image, the RMSE of the image filtered with a median filter, and the image filtered by Gaussian blurring.

the original noise-free image. It can be seen that the selection of a suitable threshold value has a major influence on the filtered image. A low threshold value reduces the efficacy of the filter, and a significant noise component is left in the image. High threshold values lead to the removal of image detail and cause block artifacts in the reconstruction. For comparison, the RMSE values of a Gaussian blurring filter and a median filter are shown. Although both median filter and Gaussian blurring yield good RMSE values, the perceived quality of the wavelet-filtered image is superior because it retains its edge information. The choice of wavelet function for decomposition and reconstruction also has an influence on the filter efficacy. The example in Figure 4.12 was created by using the Daubechies-4 wavelet. With the Haar wavelet and soft thresholding, the minimum RMSE is 5.44, compared to 5.1 when using the Daubechies-4 wavelet. An even smoother wavelet, the Daubechies-12 wavelet, reduces the RMSE to 4.88, whereas no further reduction is seen with the Daubechies-20 wavelet. The influence of the wavelet choice on the reconstructed image can be seen in Figure 4.13. With the Haar wavelet, the reconstruction appears to have square patches, which are particularly visible in the liver area. With both Daubechies wavelets, the tissue seems to blur into the background (stronger with the Daubechies-20 wavelet).

Many wavelet denoising approaches are based on the wavelet shrinkage filter developed by Donoho and Johnstone.[12,13] The statistical modeling of the wavelet coefficients is an important step toward automated determination of the optimum shrinkage threshold.[27] Donoho and Johnstone[13] suggested using the median absolute wavelet coefficient, divided by 0.6745, as an estimator for the noise variance:

$$\hat{\sigma} = \frac{\text{median } |\gamma(k)|}{0.6745} \tag{4.21}$$

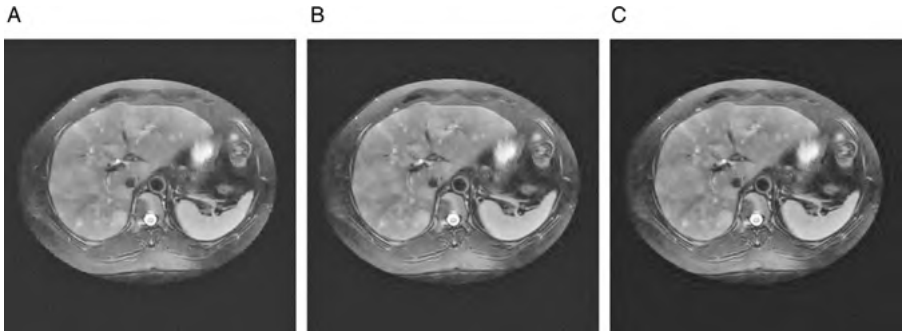


FIGURE 4.13 Influence of the wavelet choice on the reconstructed, denoised image. In all cases, a gamma contrast-enhancement function has been applied to make the background texture more visible. Image A was reconstructed using the Haar wavelet, for image B the Daubechies-12 wavelet was used, and for image C the Daubechies-20 wavelet was used.

This estimate can be used for σ in Equation (4.20). This approximation is valid only if the noise component is zero-mean, Gaussian, uncorrelated, additive noise. In other words, each wavelet coefficient $\gamma(k)$ is a superposition of the "clean" wavelet coefficient $\gamma_0(k)$ from the idealized, noise-free image and a contribution $\epsilon(k)$ from additive Gaussian white noise, that is, $\gamma(k) = \gamma_0(k) + \epsilon(k)$.

Another proposed model[23] that leads to a spatially adaptive filter also assumes that the wavelet coefficients contain an additive zero-mean, Gaussian-distributed random component but with a high local correlation and known global standard deviation σ_N. In this case, an estimate for $\gamma_0(k)$ can be given that uses a local minimum mean-squared-error method:

$$\hat{\gamma_0}(k) = \frac{\sigma^2(k)}{\sigma^2(k) + \sigma_n^2}\gamma(k) \tag{4.22}$$

The local variance $\sigma^2(k)$ can be estimated by using

$$\sigma^2(k) = \frac{N}{4\lambda}\left[\sqrt{1 + \frac{8\lambda}{N^2}\sum_{j\epsilon\Omega}\gamma^2(j)} - 1\right] - \sigma_n^2 \tag{4.23}$$

under the assumption that the wavelet coefficients are distributed with an exponential probability function:

$$P(\sigma^2) \propto \lambda e^{-\lambda\sigma^2} \tag{4.24}$$

In Equation (4.23), the local neighborhood around pixel k is Ω (e.g., a $n \times n$ square neighborhood) and contains N elements. The exponent λ can be determined by computing the histogram of local variances for the wavelet coefficients and by fitting

the function in Equation (4.24) to the histogram. In the actual denoising process, each highpass coefficient $\gamma(k)$ is replaced by the estimated noise-free coefficient $\hat{\gamma}_0(k)$, and the resulting wavelet coefficient image is subjected to the inverse wavelet transform.

Yet another technique used to find an optimum threshold for wavelet shrinkage is the generalized cross-validation (GCV) technique.[17] Generalized cross-validation entails finding a threshold τ so that the GCV estimator, $G(\tau)$, is minimized in the equation

$$G(\tau) = \frac{N_l \sum_{i=0}^{N_l-1} [\gamma_l(i) - \hat{\gamma}_l(i)]^2}{T_l^2} \tag{4.25}$$

where the $\gamma_l(i)$ are the original highpass wavelet coefficients at decomposition level l, the $\hat{\gamma}_l(i)$ are the wavelet coefficents after application of threshold τ, N_l is the total number of wavelet coefficients at decomposition level l, and T_l is the number of wavelet coefficients that have been set to zero as a consequence of the thresholding process. It is important that hard thresholding be used for the computation of the $\hat{\gamma}_l(i)$ and $G(\tau)$. Once the value for τ that minimizes $G(\tau)$ has been found, the $\hat{\gamma}_l(i)$ are used to reconstruct the image through the inverse wavelet transform. The estimator $G(\tau)$ is computed for, and applied to, each decomposition level independently.

The denoising performance of the three unsupervised threshold functions described above can be seen in Figure 4.14. The example is again based on the noisy MR slice in Figure 4.9A and a three-level decomposition using the Daubechies-12 wavelet. From Figure 4.14 it can be seen that the universal threshold underestimates the noise component in strongly noise-affected images. Multiplying the universal threshold with the estimated standard deviation, obtained through Equation (4.21), gives a very high threshold that is optimal with respect to RMSE and coincides with the threshold in Figure 4.12, where the hard-threshold minimum RMSE was found. The histogram shows a deviation from the Gaussian shape and therefore indicates that image information was removed in the filtering process. Also, the reconstruction shows block artifacts (compare to Figure 4.11B). The filter used for Figure 4.14D differs from Figure 4.14B and C insofar as individual threshold values were determined for each decomposition level. The optimization of the GCV [Equation (4.25)] is performed for each decomposition level. Applying an optimized threshold for each decomposition level provides noise removal close to the actual added noise component with no visible image structures removed.

The GCV algorithm is not difficult to implement. Although more efficient minimum-search methods exist, a simple brute-force approach provides a suitable starting point. A suggested implementation of GCV-based denoising is presented in Algorithm 4.6. This algorithm performs an in-place wavelet shrinkage on the wavelet-transformed image $W(x,y)$, and uses the function `thresh(coeff,t)` to perform hard thresholding on a coefficient `coeff` with threshold `t`. This function compares the absolute of `coeff` with `t` and returns the value of coeff if `|coeff|>t` and zero otherwise. Also, a reasonable upper bound for the threshold, `uthresh`, needs to be specified. In the simplest case, this variable may be set to the largest wavelet coefficient in the input image.

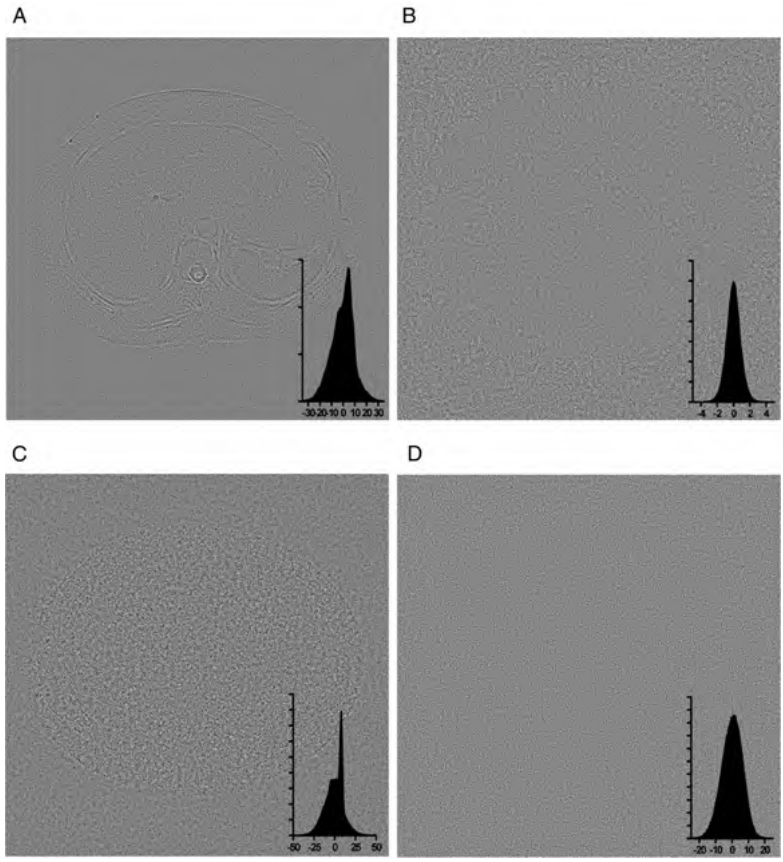


FIGURE 4.14 Comparison of the denoising efficacy of three automated threshold-finding algorithms with conventional Gaussian blurring. Shown is the noise component removed from Figure 4.9A, that is, difference between image 4.9A and the filtered image. The insets show the gray-value histograms of the various images. In image A, conventional Gaussian blurring was applied. The difference image clearly shows the noise removed in most areas, but it also contains image information: namely, the edges that get blurred in the smoothing process. Image B was produced using Donoho's universal threshold $\tau = \sqrt{2 \log n}$, image C used a threshold based on an estimate of σ through Equations (4.20) and (4.21), and image D was produced using general cross-validation, Equation (4.25). In the latter case, the noise component is almost identical to the artificially added noise, and no structural elements can be seen in the difference image.

4.3.2. Wavelet-Based Highpass Filtering

The complementary operation to wavelet-based lowpass filtering is wavelet-based highpass filtering, a process analogous to highpass filtering in the frequency domain. To create a wavelet-based highpass filter, the wavelet coefficients of the lowest detail level, that is, the lowpass output of the last subband stage (the λ_k), need to be

```
x1=xm; y1=ym;        // Start with the full-sized image

for (lvl=1 while lvl<=L increment lvl=lvl+1) // outer loop over decomp

   x2=x1/2; y2=y1/2;      // lower right corner of next decomp area

   // Part 1: Brute-force method to find gcv-minimizing threshold
   // Go over all possible thresholds from 0 to uthresh in small steps
   // and compute GCV(t). Determine the smallest GCV(t) and store in
   // gcvmin and optthresh. The latter is the threshold we will use
   // for wavelet shrinking at this level.

   gcvmin=1e20; optthresh=0;
   for (t=0 while t<uthresh increment t=t+uthresh/200)
      gcv=0.0; cnt=0;
      for (y=0 while y<ym increment y=y+1)
         for (x=0 while x<xm increment x=x+1)
            if (x>=x2 or y>=y2) then    // Exclude L/L area
               a = hthresh(W(x,y), t);
               gcv = gcv+(a-W(x,y))^2; // numerator of Eq. (4.25)
               if (a==0) then
                  cnt = cnt+1;          // denominator of Eq. (4.25)
               endif;
            endif;
         endfor;
      endfor;
      if (cnt>0) then
         gcv=gcv/(cnt*cnt);             // actual GCV sans constant multiplier
         if (gcv<gcvmin) then           // Store the minimum
            gcvmin=gcv;
            optthresh=t;
         endif;
      endif;
   endfor;

   // At this time, we have the optimum threshold optthresh for level lvl
   // Proceed to shrinking the wavelet coefficients
```

Algorithm 4.6 GCV-based denoising. Pseudocode to perform wavelet shrinking with a separate threshold per decomposition level determined through minimization of GCV. The algorithm expects a wavelet-transformed input image `W(x,y)` with multiple levels of decomposition, arranged as shown in Figure 4.8. The size of `W(x,y)` is `xm` and `ym`, both powers of 2. The number of decomposition levels is `L`. This algorithm relies on a function hthresh for hard thresholding. Wavelet coefficient shrinking takes place in `W(x,y)`, and upon termination of this algorithm, `W` may be subjected to the inverse wavelet transform to restore the denoised image. (*The algorithm is continued on the next page.*)

```
    // Part 2:
    // Run over L/H, H/H, and H/L areas of this decomposition level
    // and perform wavelet shrinking by hard thresholding

    for (y=0 while y<ym increment y=y+1)
       for (x=0 while x<xm increment x=x+1)
          if (x>=x2 or y>=y2) then          // Exclude L/L area
             W(x,y) = hthresh(W(x,y), t);    // actual wavelet shrinking
          endif;
       endfor;
    endfor;

    // Done for this level, now prepare for next decomposition level

    x1=x2; y1=y2;        // dyadic reduction of outer rectangle

endfor;                  // finished for all decomposition levels
```

Algorithm 4.6 GCV-based denoising (*continued*). This part performs the actual wavelet shrinking.

attenuated. Although this process is rarely found in image processing, it is conceptually easy to implement. Setting the L/L coefficients of the lowest detail decomposition to zero, or shrinking them strongly, effectively removes background trends, but it also causes block artifacts upon reconstruction. In analogy to the threshold functions shown in Figure 4.10, a soft attenuation of the lowpass coefficients λ_k can be achieved with a saturation function (Figure 4.15) that attenuates the lowpass coefficients λ_k if they exceed a certain threshold τ. Such a saturation function can be described as

$$\lambda'_k = \begin{cases} \lambda_k & \text{for } \lambda_k \leq \tau \\ \tau + \dfrac{\lambda_k - \tau}{k} & \text{for } \lambda_k > \tau \end{cases} \tag{4.26}$$

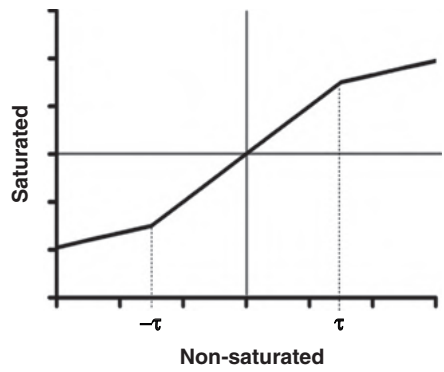


FIGURE 4.15 Saturation function to attenuate high-level lowpass subband coefficients.

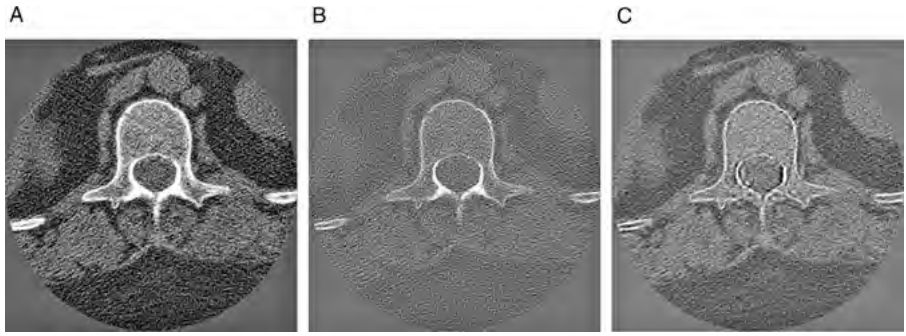


FIGURE 4.16 Comparison of convolution-based and wavelet-based sharpening. Image A is a cross-sectional CT image of a lumbar vertebra. Convolution-based sharpening (B) enhances the texture of the image, but it also strongly amplifies noise on the pixel level. With wavelet-based sharpening (C), perceived sharpness (texture enhancement) is comparable to convolution-based sharpening, but pixel noise (additive Gaussian noise) is not amplified.

where k is the attenuation factor for high-valued coefficients. The sharpening operation involves three steps: wavelet decomposition of the original image, application of the saturation function to the L/L coefficients of the lowest decomposition level, and wavelet reconstruction. The result is comparable to a sharpening operation (see Section 2.3), but it does not amplify additive noise on the pixel level. This effect can be explained with the frequency response of the filters. The sharpening operation is based on the Laplacian filter, which emphasizes frequency components proportional to the square of their spatial frequency. Therefore, noise on the pixel level is most strongly amplified. Conversely, wavelet sharpening can be seen as wavelet shrinking at the lowest frequency level. As a consequence, no noise amplification takes place. An example is shown in Figure 4.16, where the saturation operation has been applied to the L/L region of a three-level decomposition with the Daubechies-12 wavelet.

In a similar manner, very effective removal of background inhomogeneities is possible (background flattening). The conventional method is unsharp masking, that is, the subtraction of a strongly blurred version of the image from the original image. Depending on the window size, unsharp masking is affected adversely by strong edges. If the same operation, unsharp masking, is performed on the L/L region of the lowest decomposition level, image detail is not affected. The wavelet-based unsharp masking operation therefore involves three steps: wavelet decomposition of the original image, application of an unsharp masking operation to the L/L coefficients of the lowest decomposition level, and wavelet reconstruction. The result of this operation is shown in Figure 4.17. For this filter, the image was decomposed over three levels with the Daubechies-12 wavelet. Unsharp masking was performed on the lowest level, a square of 64×64 pixels in size, with a 32-pixel circular mask. Wavelet-based filtering removes the inhomogeneous background more effectively, while the background texture is retained and no burned-out regions appear.

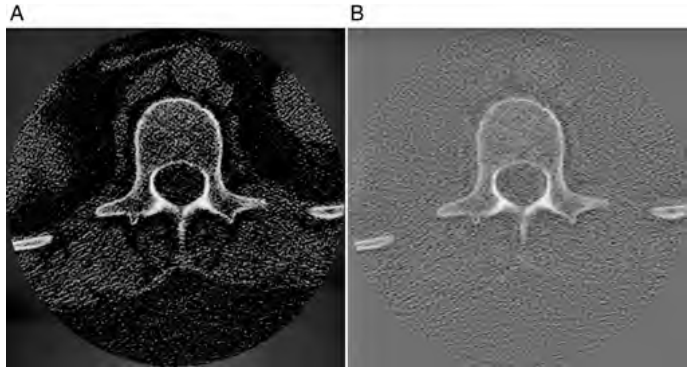


FIGURE 4.17 Comparison of conventional unsharp masking and unsharp masking on the lowest level of wavelet decomposition. The original image is the CT cross section in Figure 4.16. Image A shows the result of unsharp masking, and image B shows the result of a wavelet-based operation, where only the lowest decomposition level was subjected to unsharp masking. Both images were subjected to gamma contrast enhancement to make the background texture more visible. Conventional unsharp masking shows loss of texture detail and large burned-out regions, whereas wavelet-based background removal shows a much more detailed texture.

4.4. COMPARISON OF FREQUENCY-DOMAIN ANALYSIS TO WAVELET ANALYSIS

Both the Fourier transform and the wavelet transform provide linear decompositions of a signal $f(t)$ into coefficients a_k such that

$$f(t) = \sum_{k=-\infty}^{\infty} a_k \zeta_k(t) \tag{4.27}$$

and $\zeta_k(t)$ are the basis functions. In the case of the Fourier transform, the $\zeta_k(t)$ are complex oscillations $\exp(-2\pi jk\omega t)$. These oscillations are continuous from $-\infty < t < \infty$, which implies that the function $f(t)$ is a 2π-periodic signal that stretches from $-\infty$ to ∞. The wavelet transform, on the other hand, uses basis functions with compact support. Furthermore, the wavelet transform relies on a two-dimensional set of coefficients $a_{j,k}$, so that

$$f(t) = \sum_{k} \sum_{j} a_{j,k} \psi_{j,k}(t) \tag{4.28}$$

and the set of basis functions $\psi_{j,k}(t)$ reflects the scaled and shifted mother wavelet. As a consequence of the compact support of the wavelets, $f(t)$ itself may be nonperiodic or have compact support itself. The assumption of a periodic $f(t)$ is often violated in the discrete Fourier transform and leads to techniques such as multiplication with a window function to eliminate the discontinuity between the start and the end of a

discretely sampled signal or image. These considerations do not apply to the wavelet transform.

The most crucial difference between the Fourier transform and the wavelet transform is the latter's ability to retain spatial information. A Fourier-transformed image contains the summed-up contributions of all image areas toward a spatial frequency irrespective of their location. A wavelet-transformed image, on the other hand, separates the image by frequency bands while it retains the spatial information in each band. For this reason, nonlinear and adaptive filter techniques can be designed that would not be possible with the Fourier transform. Examples of nonlinear techniques are the filter functions shown in Figures 4.10 and 4.15. Examples of adaptive filters include a locally variable threshold in Figure 4.10, determined from the local noise variance. Furthermore, wavelet-based filters can include a priori spatial knowledge: for example, a restriction to segmented areas of the image.

A second fundamental difference between the Fourier transform and the wavelet transform is the ability of the wavelet transform to control the level of detail with the scaling variable, s in Equation (4.4). The Fourier transform, in comparison (see Figure 4.18), merely provides the global contribution of the frequency components toward the signal. The Fourier transform can be restricted in time, leading to the windowed Fourier transform, but as the window size decreases, less information about high frequencies will be provided. Whereas the Fourier transform is restricted to periodic sinusoidal oscillations, the wavelet transform allows the flexibility of choice of a suitable wavelet basis function for optimized smoothness and frequency response of the filter bank.

Another important property of the wavelet transform cannot readily be quantified. The application of filters to multiple scales retains self-similar properties. Many objects have some self-similar properties that are perceived as natural. Therefore, the results of wavelet-based filters often appear more natural than those of Fourier- or convolution-based filters. The most prominent example is wavelet-based image

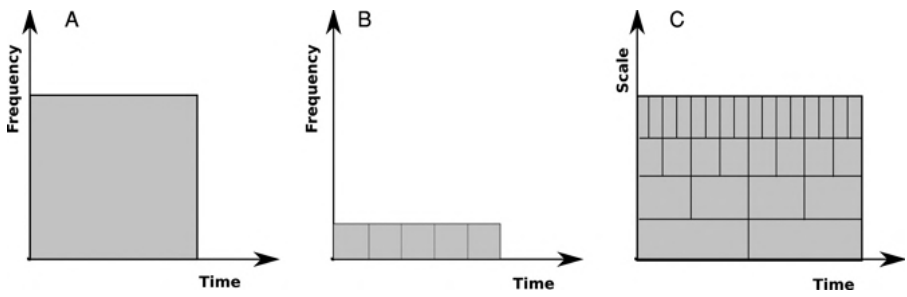


FIGURE 4.18 Comparison of the Fourier transform to the wavelet transform. The Fourier transform of a signal (A) provides the global frequency components without respect to their location in time. A windowed Fourier transform (B) allows some localization with respect to time, but as the windows become smaller in time, so does their ability to provide high-frequency information. The wavelet transform (C), by merit of the scalability of the basis functions, provides localized high-frequency information.

TABLE 4.2 Comparison of the Wavelet Transform to the Fourier Transform

Fourier Transform	Wavelet Transform
Linear, separable in *n* dimensions.	Linear, separable in *n* dimensions.
Basis functions with infinite support require assumption of periodicity.	Basis functions with finite support.
Because of the assumed periodicity, discontinuities at the edges need to be suppressed (windowing).	No assumption of periodicity; therefore, no windowing is needed.
Fixed basis functions (sin, cos).	Flexible basis functions; can be optimized for a specific application.
Basis functions are computationally expensive.	Dyadic wavelet transform is faster than FFT because only multiplications and additions are needed.
Complete loss of spatial information.	Spatial information is retained on all levels.
No multiscale decomposition possible.	Allows multiscale decomposition and multiscale analysis.
Filters may only use frequency information.	Filters can use spatial information to establish adaptive filter schemes, locally variable thresholds, or use a priori shape information.

compression. Lossy image compression implies that the original image cannot be reconstructed—a price to be paid for very high compression rates. A very common method of lossy compression is based on the discrete cosine transform (Section 3.3), whereby small square areas of the image are transformed and high-frequency coefficients are discarded. This compression method leads to typical block artifacts. Wavelet-based compression does not show these artifacts, and the perceived image quality of images highly compressed with a wavelet algorithm is much higher than that of images compressed with the discrete cosine transform. The advantages of wavelet-based compression are so high that this new compression method became part of the JPEG-2000 standard. Wavelet-based image compression is covered in detail in Section 12.3. The key differences between the Fourier transform and the wavelet transform, with its implication on filters, are summarized in Table 4.2.

4.5. BIOMEDICAL EXAMPLES

The importance of the wavelet transform for biomedical image processing was realized almost immediately after the key concepts became widely known. Two reviews[18,20] cover most of the fundamental concepts and applications. In addition, the application of wavelets in medical imaging has led to development of powerful wavelet toolboxes that can be downloaded freely from the Web. A list of some wavelet software applications is provided in Section 14.6.

The most dominant application is arguably wavelet-based denoising. Noise is an inescapable component of medical images, since noise is introduced in several steps of signal acquisition and image formation. A good example is magnetic resonance imaging, where very small signals (the magnetic echos of spinning protons) need to be amplified by large factors. Many fast MRI acquisition sequences gain acquisition speed at the expense of the signal/noise ratio (SNR). For this reason, the availability of detail-preserving noise-reduction schemes are crucial to optimizing MR image quality. A comprehensive survey of different wavelet-based denoising algorithms in MR images was conducted by Wink and Roerdink[33] together with a comparison to Gaussian blurring. Gaussian blurring is capable of improving the SNR by 10 to 15 dB in images with very low SNR. However, when the SNR of the original image exceeds 15 to 20 dB (i.e., has a relatively low noise component), Gaussian blurring does not provide further improvement of SNR. Most wavelet denoising schemes, on the other hand, show consistent improvement of the SNR by 15 to 20 dB even in images with a low initial noise component. Noise is particularly prevalent in MR images with small voxel sizes: for example, in high-resolution MR microscopy images. For those cases, a wavelet denoising scheme was introduced by Ghugre et al.[15] The three-dimensional algorithm is based on hard thresholding and averaging of multiple shifted transforms following the translation-invariant denoising method by Coifman and Donoho.[7] These shifts are particularly important, since Ghugre et al. subdivide the three-dimensional MRI volume into smaller $64 \times 64 \times 64$ voxel blocks and denoise each block individually. Without the shift-average operation, the subdivision of the volume would cause blocking effects. Three-dimensional denoising outperformed corresponding two-dimensional methods both in mean-squared-error metrics and in subjective perception. Finally, in the special case of magnetic resonance images, the denoising process can be applied to both the real and imaginary parts of the spatial image.[25,36] The k-space matrix was reconstructed through an inverse Fourier transform, resulting in a real and an imaginary spatial image. Soft-threshold wavelet shrinking with the threshold determined by generalized cross-validation [Equation (4.25)] was applied on both real and imaginary parts before computation of the spatial magnitude image, which was the final reconstruction step.

Plain x-ray imaging is much less susceptible to noise than is MRI. Yet some applications, predominantly mammography, require particularly low noise with excellent preservation of detail. The ability of wavelet-based methods to achieve these goals has been examined. Scharcanski and Jung[30] presented an x-ray denoising technique based on wavelet shrinkage (hard thresholding) as the second step in a two-step process. The first step is local contrast enhancement. The standard deviation of x-ray noise component increases with image intensity, and a contrast enhancement function was designed that acts similar to local histogram equalization but depends on the local noise component. Wavelet shrinkage was optimized under the assumption of a Gaussian distribution of the noise component, but a Laplacian distribution of noise-free wavelet coefficients. The end effect of local contrast enhancement and subsequent wavelet denoising is a mammography image where local features, such as microcalcifications, are strongly contrast enhanced and visually more dominant than in either the original image, the image subjected to local histogram equalization, or the image

subjected to noise-dependent contrast enhancement only. A similar idea of a two-step approach involving local contrast enhancement and adaptive denoising was followed by Sakellaropoulos et al.[29] In this study, a nondecimating wavelet transform (i.e., a wavelet transform that does not downsample the lowpass-filtered component) was used, and the wavelet coefficients were denoised by soft thresholding followed by applying nonlinear gain at each scale for adaptive contrast enhancement.

Ultrasound imaging is another biomedical imaging modality where noise is dominant and leads to poor image quality. Usually, skilled medical personnel are needed to interpret ultrasound images. Suitable image processing methods have been developed to improve image quality and help interpret the images. To remove noise and noise-related ultrasound speckles, wavelet-based methods have been developed. Contrary to MR and x-ray images, ultrasound speckle noise is frequently interpreted as multiplicative noise [i.e., the A-mode signal $f(z)$ consists of the echo intensity $e(z)$ as a function of depth z and noise n such that $f(z) = e(z)n$]. Consequently, taking the logarithm of $f(z)$ converts multiplicative noise into additive noise. A straightforward approach, such as proposed by Gupta et al.,[16] would be to convert the ultrasound image to its log-transformed form, perform a wavelet decomposition, use a soft-thresholding wavelet shrinkage for denoising, reconstruct the image through an inverse wavelet transform, and restore the denoised intensity values through a pixelwise exponential function. This idea can be taken even further when the wavelet-based denoising filter is built into the ultrasound machine at an early stage of the image formation process. Such a filter would act on the RF signal prior to demodulation. A wavelet filter that acts either on the RF signal before demodulation or on the actual image data was proposed by Michailovich and Tannenbaum.[22] However, this study indicated that the model of uncorrelated, Gaussian, multiplicative noise was overly simplified. In fact, Michailovich and Tannenbaum demonstrate a high autocorrelation of the noise component, depending on its position in the B-mode scan. By using a nonlinear filter, they propose to decorrelate the noise, after which soft thresholding can be applied on the logarithmic data. A completely different approach was taken by Yue et al.,[35] who combined wavelet decomposition with a nonlinear anisotropic diffusion filter (see Section 5.1). In this special case, wavelet decomposition of the image took place over three levels, and all wavelet coefficients except the lowest-detail region were subjected individually to anisotropic diffusion. Since the wavelet transform is a powerful method of separating signal and noise, the anisotropic diffusion filter can preserve edges even better than the same filter applied on the original image.

Single-photon emission computed tomography (SPECT) and positron emission tomography (PET) are modalities that rely on radioactive decay of radiopharmaceuticals. The decay is a random event, and with the required low doses of radioactive material, the signal is composed of a few integrated events. In other words, the image is made of noise. One model for this type of noise is the Poisson model, where the standard deviation is proportional to the signal intensity. For visualization purposes, the SPECT or PET signal is frequently severely blurred, false-colored, and superimposed over a higher-resolution MR or CT image. Wavelet-based denoising is a promising approach to improving SPECT and PET image quality. For example, Bronnikov[3]

suggested the use of a wavelet shrinkage filter in low-count cone-beam computed tomography, which is related to SPECT in its high Poisson noise content. For PET images, a three-dimensional extension of the wavelet denoising technique based on a threshold found through general cross-validation was proposed by Charnigo et al.,[6] who describe a multidimensional wavelet decomposition strategy that is applied to small nonoverlapping cubes (subspaces) of the original three-dimensional PET image. To find a suitable threshold for wavelet coefficient shrinking, Stein's unbiased risk estimator (SURE)[11,31] was proposed, but with an adjustable parameter to either oversmooth or undersmooth [similar to the parameter Υ in Equation (4.20)]. Charnigo et al. point out that undersmoothing (i.e., selecting a weaker than optimal filter) in medical images is preferable to oversmoothing. The efficacy of different denoising protocols in PET time-course images of the heart was analyzed by Lin et al.,[21] who propose a wavelet decomposition scheme that does not include subsampling. The resulting redundancy led to shift invariance of the wavelet filter. Some of the denoising protocols proposed allowed more accurate determination of myocaridal perfusion and better differentiating between normal and underperfused regions. Similar conclusions were found in a study by Su and et al.[32] on simulated PET images and small-animal PET images. Wavelet filtering improved nonlinear least-squares model fitting of time-course models to the data, particularly in images with high noise levels when compared to conventional smoothing filters.

Neither light microscopic nor electron microscopic images are typically known as being extremely noisy. However, there are conditions under which the signal-to-noise ratio is very low, particularly when imaging weak fluorescent signals or reducing exposure to reduce photobleaching. For this type of image, Moss et al.[24] developed a wavelet-based size filter that emphasizes structures of a specified size and made use of the property of the wavelet transform to provide the strength of correlation of the data with the wavelet.[24] This particular filter makes use of the continuous wavelet transform [Equation (4.4)] rather than the fast dyadic wavelet transform. Furthermore, Moss et al. propose designing a wavelet transform in three dimensions that uses separability to improve computational efficiency and symmetry to provide rotation invariance. In a different study, Boutet de Monvel et al.[2] present wavelet-based improvements to the deconvolution of confocal microscope images. They noted that the deconvolution process affects image detail differently depending on its size: namely, that small image features are more sensitive than large features to noise or mismatches of the optical point-spread function. Furthermore, the iterative nature of the deconvolution process exacerbates this effect. It was found that a wavelet-based denoising step introduced between the deconvolution iterations strongly improves the image quality of the final reconstructed images.

Optical coherence tomography (OCT) is a more recent imaging modality where light-scattering properties of tissue provide contrast. Broadband laser light (i.e., laser light with a short coherent length) is directed onto the tissue through an interferometer. Laser light that is scattered back from the tissue into the optical system provides a signal only from within the short coherent section. The coherent section and thus the axial resolution is several micrometers deep. Scanning the coherent section in the axial direction provides the optical equivalent of an ultrasound A-mode scan. Most

OCT devices generate a two-dimensional cross section analogous to the B-mode scan of an ultrasound device. Similar to ultrasound, OCT images are strongly affected by noise, particularly additive Gaussian noise and multiplicative noise. Gargesha et al.[14] show that conventional filtering techniques (median filtering, Wiener filtering) and wavelet-based filtering with orthogonal wavelets do not sufficiently reduce noise while retaining structural information. The filter scheme proposed by Gargesha et al. is based on an idea by Kovesi[19] to perform denoising in the Fourier domain, shrinking only the magnitude coefficients and leaving the phase information unchanged. The improvement by Gargesha et al. consists of adaptively optimizing the filter parameters and using nonlinear diffusion to reduce noise at different scales.

Whereas wavelets are clearly most popular in image denoising, other areas benefit from the wavelet transform as well. To give two particularly interesting examples, the wavelet transform can be used in the image formation process of computed tomography images, and wavelet-based interpolation provides a higher interpolated image quality than do conventional schemes. In computed tomography, reconstruction is based on the Fourier slice theorem. Peyrin et al.[26] showed that tomographic reconstruction from projections is possible with the wavelet transform. Based on Peyrin et al.[26] approach, Bonnet et al.[1] suggest a wavelet-based modification of the Feldkamp algorithm for the reconstruction of three-dimensional cone-beam tomography data. Another interesting application of the wavelet transform is the reconstruction of tomographic images from a low number of projections that cover a limited angle.[28] The main challenge with reduced-angle backprojection lies in its inability to reconstruct reliably edges perpendicular to an x-ray beam. Therefore, discontinuities that are not tangential to any of the projections in the limited angle range will be represented poorly in the reconstructed image. The wavelet reconstruction by Rantala et al.[28] is based on modeling the statistical properties of the projection data and iterative optimization of a maximum a posteriori estimate.

Image interpolation can be envisioned as adding a new highest-detail scale, as shown in Figure 4.19. Decomposition of the original image provides the gray shaded decomposition levels. A new level is added for reconstruction, which has twice the side length of the original image. This level is shown in white in Figure 4.19 and labeled L/H*, H/L*, and H/H*. If the entire image is reconstructed (including the white areas), the image resolution will have doubled. The main task of the image interpolation technique is to find suitable information for the new (white) areas that provide the new interpolated image detail. Carey et al.[5] presented an interpolation method that enhances the edge information by measuring the decay of wavelet transform coefficients across scales and preserves the underlying regularity by extrapolating into the new subband (H/L*, L/H*, and H/H*) to be used in image resynthesis. By using the information that exists on multiple scales, the soft interpolated edges associated with linear and cubic interpolation schemes can be avoided. A different approach is taken by Woo et al.,[34] who use the statistical distribution of wavelet coefficients in two subbands of the original image (light gray in Figure 4.19) to estimate a probable distribution in the new extrapolated subband. The new subbands are then filled with random variables that have this distribution.

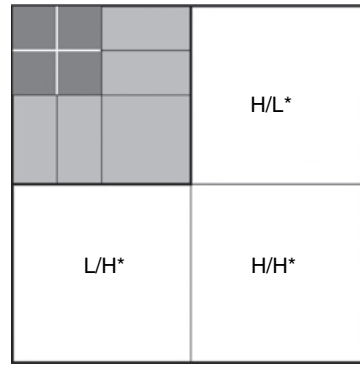


FIGURE 4.19 Principle of image interpolation. The gray shaded areas are the wavelet decomposition of the original image, and suitable information needs to be found for the white area.

Finally, a recent publication by Daszykowski et al.[8] gives an interesting example of a complete image processing chain (in this case the analysis of electrophoresis images) that includes wavelet denoising as one processing step. The image processing chain begins with digitally scanned images of electrophoresis gels. A spline-based function fit is used for removal of background inhomogeneities, followed by wavelet shrinking for denoising. After the denoising steps, geometrical distortions are removed and the image is binarized by Otsu's threshold. The features are finally separated by means of the watershed transform, and the blots are classified by using different statistical models. This publication describes a highly automated start-to-end image processing chain.

REFERENCES

1. Bonnet S, Peyrin F, Turjman F, Prost R. Nonseparable wavelet-based cone-beam reconstruction in 3-D rotational angiography. IEEE Trans Med Imaging 2003; 22(3):360–367.
2. Boutet de Monvel J, Le Calvez S, Ulfendahl M. Image restoration for confocal microscopy: improving the limits of deconvolution, with applications to the visualization of the mammalian hearing organ. Biophys J 2001; 80:2455–2470.
3. Bronnikov AV. A filtering approach to image reconstruction in 3D SPECT. Phys Med Biol 2000; 45(9):2639–2651.
4. Burrus CS, Gopinath RA, Guo H. Introduction to Wavelets and Wavelet Transforms: A Primer. Upper Saddle River, NJ: Prentice Hall, 1998.
5. Carey WK, Chuang DB, Hemami SS. Regularity-preserving image interpolation. IEEE Trans Image Process 1999; 8(9):1293–1297.
6. Charnigo R, Sun J, Muzic R. A semi-local paradigm for wavelet denoising. IEEE Trans Image Process 2006; 15(3):666–677.

7. Coifman RR, Donoho DL. Translation-Invariant De-noising. Lecture Notes in Statistics. New York: Springer-Verlag, 1995; 125.
8. Daszykowski M, Stanimirova I, Bodzon-Kulakowska A, Silberring J, Lubec G, Walczak B. Start-to-end processing of two-dimensional gel electrophoretic images. J Chromatogr A 2007; 1158(1–2):306–317.
9. Daubechies I. Orthonormal bases of compactly supported wavelets. Commun Pure Appl Math 1988; 41(7):909–996.
10. Daubechies I. Ten Lectures on Wavelets. Philadelphia: SIAM, 1992.
11. Donoho DL. Adapting to unknown smoothness via wavelet shrinkage. J Am Stat Soc 1995; 90:1200–1224.
12. Donoho DL. De-noising by soft-thresholding. IEEE Trans Inf Theory 1995; 41(3):613–627.
13. Donoho DL, Johnstone IM. Ideal spatial adaptation via wavelet shrinkage. Biometrika 1994; 81:425–455.
14. Gargesha M, Jenkins MW, Rollins AM, Wilson DL. Denoising and 4D visualization of OCT images. Opt Express 2008; 16(16):12313–12333.
15. Ghugre NR, Martin M, Scadeng M, Ruffins S, Hiltner T, Pautler R, Waters C, Readhead C, Jacobs R, Wood JC. Superiority of 3D wavelet-packet denoising in MR microscopy. Magn Reson Imaging 2003; 21(8):913–921.
16. Gupta S, Chauhan RC, Sexana SC. Wavelet-based statistical approach for speckle reduction in medical ultrasound images. Med Biol Eng Comput 2004; 42(2):189–192.
17. Jansen M, Bultheel A. Multiple wavelet threshold estimation by generalized cross-validation for images with correlated noise. IEEE Trans Image Process 1999; 8:947–953.
18. Jin Y, Angelini E, Laine A. Wavelets in medical image processing: denoising, segmentation, and registration. In: Suri JS, Wilson D, Laxminarayan S, editors. Handbook of Biomedical Image Analysis. New York: Springer, 2005.
19. Kovesi P. Phase preserving denoising of images. Proc DICTA 99 (Perth, Australia) 1999; 212–217.
20. Laine AF. Wavelets in temporal and spatial processing of biomedical images. Annu Rev Biomed Eng 2000; 2(1):511–550.
21. Lin JW, Laine AF, Bergmann SR. Improving PET-based physiological quantification through methods of wavelet denoising. IEEE Trans Biomed Eng 2001; 48(2):202–212.
22. Michailovich OV, Tannenbaum A. Despeckling of medical ultrasound images. IEEE Trans Ultrason Ferroelectr Freq Control 2006; 53(1):64–78.
23. Mihçak MK, Kozintsev I, Ramchandran K, Moulin P. Low-complexity image denoising based on statistical modeling of wavelet coefficients. IEEE Signal Process Lett 1999; 6(12):300–303.
24. Moss WC, Haase S, Lyle JM, Agard DA, Sedat JW. A novel 3D wavelet-based filter for visualizing features in noisy biological data. J Microsc 2005; 219(2):43–49.
25. Nevo U, Tvito I, Goelman G, Akselrod S. De-noising of MR images by complex GCV wavelet algorithm. Proc Intl Soc Mag Reson Med 2003; 2:934.
26. Peyrin F, Zaim M, Goutte R. Construction of wavelet decompositions for tomographic images. J Math Imaging Vis 1993; 3:105–121.
27. Portilla J, Strela V, Wainwright M, Simoncelli EP. Image denoising using scale mixtures of Gaussians in the wavelet domain. IEEE Trans Image Process 2003; 12:1338–1351.

28. Rantala M, Vanska S, Jarvenpaa S, Kalke M, Lassas M, Moberg J, Siltanen S. Wavelet-based reconstruction for limited-angle x-ray tomography. IEEE Trans Med Imaging 2006; 25(2):210–217.
29. Sakellaropoulos P, Costaridou L, Panayiotakis G. A wavelet-based spatially adaptive method for mammographic contrast enhancement. Phy Med Biol 2003; 48(6):787–803.
30. Scharcanski J, Jung CR. Denoising and enhancing digital mammographic images for visual screening. Comput Med Imaging Graph 2006; 30(4):243–254.
31. Stein C. Estimation of the mean of a multivariate normal distribution. Ann Stat 1981; 9:1135–1151.
32. Su Y, Shoghi KI, Panettieri V, Sempau J, Andreo P. Wavelet denoising in voxel-based parametric estimation of small animal PET images: a systematic evaluation of spatial constraints and noise reduction algorithms. Phys Med Biol 2008; 53(21):5899–5915.
33. Wink AM, Roerdink J. Denoising functional MR images: a comparison of wavelet denoising and Gaussian smoothing. IEEE Trans Med Imaging 2004; 23(3):374–387.
34. Woo DH, Eom IK, Kim YS. Image interpolation based on inter-scale dependency in wavelet domain. Proc ICIP 04 2004; 3:1687–1690.
35. Yue Y, Croitoru MM, Bidani A, Zwischenberger JB, Clark JW. Nonlinear multiscale wavelet diffusion for speckle suppression and edge enhancement in ultrasound images. IEEE Trans Med Imaging 2006; 25(3):297–311.
36. Zaroubi S, Goelman G. Complex denoising of MR data via wavelet analysis: application for functional MRI. Magn Reson Imaging 2000; 18:59–68.

5

ADAPTIVE FILTERING

Conventional (nonadaptive) filters were introduced in Section 2.3. These are operators that act equally on all areas of an image. Conversely, adaptive filters change their behavior with the properties of the local neighborhood of the image to be filtered. One example of an adaptive filter is local contrast enhancement by histogram equalization. Histogram equalization is a process by which the image's gray values are modified so that the cumulative histogram follows a straight line as closely as possible (Section 2.2). Linear histogram equalization is based on a mapping function of any image intensity I to a new intensity I':

$$I' = \frac{\int_{I_{\min}}^{I} P(i)\,di}{\int_{I_{\min}}^{I_{\max}} P(i)\,di} \tag{5.1}$$

For each pixel with intensity I, the area under the histogram from the lowest intensity $I_{\min}$ to the pixel intensity I is computed and normalized by the total histogram area. This is the new intensity I'. For images with integer elements (e.g., unsigned bytes with a value range from 0 to 255), Equation (5.1) simplifies to

$$I' = \frac{1}{N} \sum_{i=0}^{I} N(i) \tag{5.2}$$

Advanced Biomedical Image Analysis, By Mark A. Haidekker

where N is the total number of pixels and $N(i)$ is the histogram value at intensity i (i.e., the number of pixels of intensity i). For global histogram equalization, the total number of pixels N and the histogram $N(i)$ refer to the entire image. For a locally adaptive variation of this process, either a neighborhood of size $m \times m$ or a circular neighborhood of radius r of each pixel is selected, and N and $N(i)$ are restricted to this neighborhood. The locally restricted histogram equalization process adapts to the statistical properties of the neighborhood region, and its exact action depends on the neighborhood rather than on the entire image.

The effect of locally adaptive contrast enhancement is demonstrated in Figure 5.1. A scanning electron microscope (SEM) image of grown collagen fibers exhibits a histogram with few bright values—the cumulative probability reaches values close to unity at relatively low gray levels. Histogram stretching (global, nonadaptive histogram equalization) leads to a more uniform gray value distribution, but many gray levels are missing, as evidenced by the gaps in the histogram, predominantly in the center region. The enhanced image does not make use of the value range available and does not have more unique gray levels than the unprocessed image. Locally adaptive contrast enhancement operating on an area of about 1/100 of the image area expands contrast within the local area. Gray values are now distributed uniformly. Strong noise amplification can be seen in the dark gaps between the collagen fibers. When local histogram equalization is performed in regions with a very narrow gray-value distribution, noise (a significant source of contrast in flat regions) is strongly amplified.

The size of the neighborhood is variable and depends on the size of the features and the size of the noise clusters. Generally, it is possible to construct adaptive methods by subdividing an image into smaller subimages and applying a global operator on the subimage. However, such a process would lead to unacceptable boundaries between the subimages. For this reason, adaptive methods use sliding (or moving) windows centered on the pixel under observation. The shape of the neighborhood also influences the result. Although a square neighborhood is easier to implement, a circular neighborhood shows less directionality. In a square neighborhood, more diagonal pixels influence the result than horizontal or vertical pixels. The treatment of image edges also needs to be considered. Most commonly, four different methods of edge handling are implemented: (1) pixels outside the image area are considered to have zero value, (2) pixels outside the image area retain the nearest edge value, (3) the image is tiled, or (4) the image is mirrored. In the context of adaptive methods, avoiding discontinuities at the edge is of critical importance, and the second and fourth methods are acceptable.

Locally adaptive algorithms can be used for noise reduction, intensity-based segmentation, and shape classification. Most locally adaptive algorithms are based on the statistical properties of the neighborhood, but more advanced filters may employ artificial intelligence methods such as fuzzy logic and neural networks.

5.1. ADAPTIVE NOISE REDUCTION

Noise reduction is one of the most important image processing steps, particularly in biomedical image processing. Noise is broadly distributed over the frequency

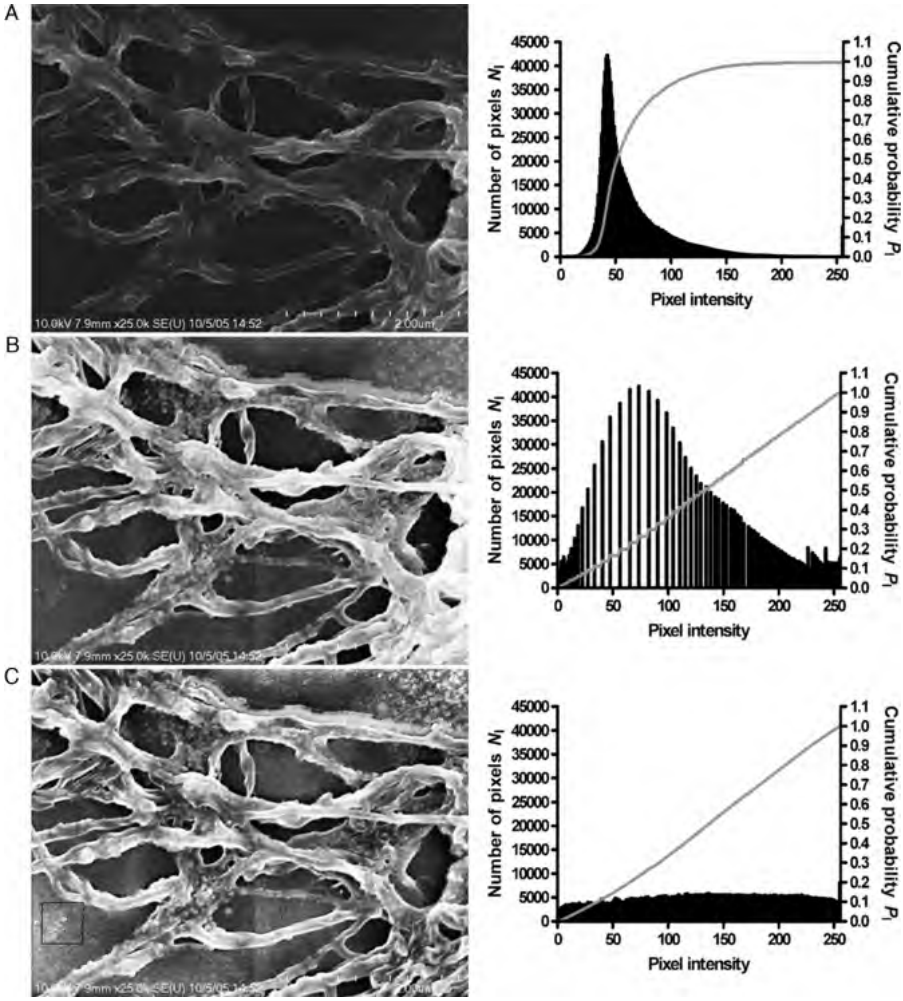


FIGURE 5.1 Global and locally adaptive histogram equalization. Image A shows a scanning electron microscope (SEM) image of grown collagen fibers. The histogram reveals an emphasis on dark shades with few bright image values. Global histogram equalization (B) enhances the contrast by brightening mid-gray pixels, but the histogram shows that the available value range is not fully used. Locally adaptive histogram equalization (C) leads to optimized contrast in each region. The neighborhood size in image C is indicated by a black rectangle near the bottom left corner.

spectrum of the image. The conventional approach to reducing noise is the application of a blurring filter (e.g., Gaussian smoothing or lowpass filtering in the frequency domain). Unfortunately, degradation of high-frequency components—edges and texture details—is an inevitable consequence of the blurring process. For this reason, many methods and algorithms have been developed that act more strongly in flat

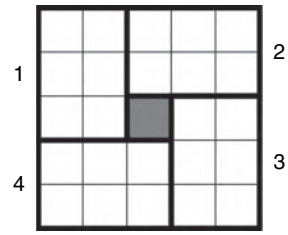


FIGURE 5.2 Pixel neighborhood definition for the Kuwahara filter in the example of a 5×5 neighborhood. The central pixel is replaced by the average value of the six pixels in the region (1 to 4) that has the lowest variance.

image regions than near edges. Generally, adaptive noise reduction results in nonlinear filters and filters that are defined only algorithmically. A good example for a nonlinear, edge-preserving noise reduction is the Kuwahara filter, although it is not an adaptive filter. The Kuwahara filter can be explained by the definition of four neighborhood areas of the central pixel (dark gray) in Figure 5.2. Four adjacent, nonoverlapping regions of the local window are examined and the central pixel value is replaced by the mean value of the pixels in the region with the lowest variance. The window size (in this example, 5×5, which leads to four six-pixel regions) can be freely selected. The averaging operation, in combination with choosing the lowest-variance quadrant, is responsible for noise reduction. If an edge runs through the window, the Kuwahara filter generally picks a region that does not contain the edge, because the edge causes a high variance where it intersects one of the four regions. The Kuwahara filter illustrates the basic idea behind edge-preserving noise-reduction filters, but superior adaptive filters have been developed.

5.1.1. Adaptive Minimum Mean-Squared-Error Filter

A simple and widely used locally adaptive noise reduction filter is the minimum mean-squared-error filter. It is based on variable-strength blurring to balance a strong filter action in flat regions with a weaker filter action in the presence of edges. If we assume that the noise component of the image is known and has an overall variance σ_N^2, we can intuitively demand that a locally adaptive filter acts strongest wherever it finds a local variance σ_{loc}^2 close to the global noise variance σ_N^2. In the presence of edges or other image features, the local variance can be expected to be higher than σ_N^2, and the adaptive filter reduces the strength of its blurring action. In its simplest implementation, the minimum mean-squared-error filter can be described by[20]

$$I'(x,y) = I(x,y) - \frac{\sigma_N^2}{\sigma_{\text{loc}}^2(x,y)}[I(x,y) - \bar{I}(x,y)] \tag{5.3}$$

where any pixel value $I\ (x,y)$ is substituted for by a new value $I'(x,y)$ that depends on the original intensity value, the local variance σ_{loc}^2 [computed inside an $M \times M$

square neighborhood with $l = M/2 - 1$, as defined in Equation (5.4)], the global noise variance σ_N^2, and the mean intensity value $\bar{I}(x,y)$ in the neighborhood:

$$\sigma_{\text{loc}}^2(x,y) = \frac{1}{(M^2-1)} \left\{ \sum_{i=x-l}^{x+l} \sum_{j=y-l}^{y+l} I(i,j)^2 - \frac{1}{M^2} \left[\sum_{i=x-l}^{x+l} \sum_{j=y-l}^{y+l} I(i,j) \right]^2 \right\} \tag{5.4}$$

Equation (5.4) is a one-pass equation that allows us to compute the local mean value and the local variance in the same loop over the square region by summing up the image values and the squared image values. It can be seen from Equation (5.3) that in areas of low local variance ($\sigma_{\text{loc}}^2 \approx \sigma_N^2$) the new image value is close to the local average value, whereas in areas with high local variance ($\sigma_{\text{loc}}^2 \gg \sigma_N^2$), the filter does not strongly modify the original intensity value. For practical application of the minimum mean-squared-error filter, the global variance can be approximated by the smallest local variance encountered:

$$\sigma_N^2 = \min_{x,y} \sigma_{\text{loc}}^2 \tag{5.5}$$

This assumption implies that areas with low image contrast exist and that the noise variance is the same in all regions of the image. An example is shown in Figure 5.3, where a CT image of the lung is subjected to a conventional Gaussian blurring filter and a local adaptive filter following Equation (5.3). The difference image (panel C) shows the noise removed, but it also shows dark areas along edges, indicating that the filtered image does not differ from the original image in those regions. A possible implementation is shown in Algorithm 5.1.

The filter described in Algorithm 5.1 is weaker than the smoothing filter that is used to compute $\bar{I}(x,y)$ in Equation (5.3). With the practical implementation described here, the minimum variance that is used for σ_N^2 may result in a very small value, making the overall effect of the filter even weaker. One possibility is to use a histogram-based approach to determine σ_N^2: for example, computing the histogram of `VARIMG` in Algorithm 5.1 and choosing σ_N^2 from the 5% or 10% quantile of the histogram. With this more refined method, the choice of σ_N^2 becomes more robust against image anomalies that violate the assumption of homogeneously distributed noise. Further improvements in the algorithm are possible by implementing a circular neighborhood. Also, the averaged image intensity $\bar{I}$ could be computed more advantageously by using a Gaussian rather than a box kernel.

The major disadvantage of this adaptive filter is the poor noise reduction near edges, which is a component of the filter design. Nonetheless, modification of this filter for improved noise reduction near edges is possible.[32] An edge detector can be used to determine if an edge exists in the local neighborhood: for example, by application of the Sobel operator or Canny edge detector with subsequent thresholding. If an edge exists, the local window can be split into two regions under the assumption that

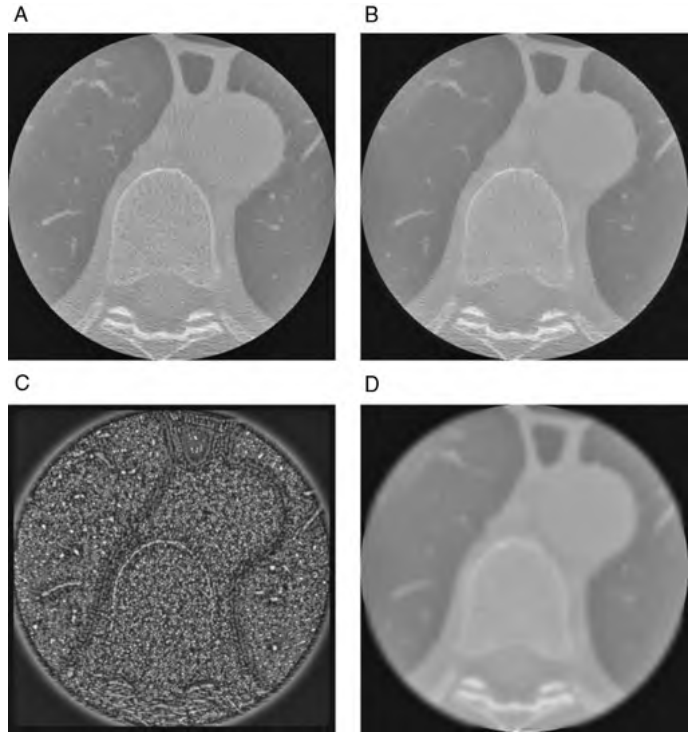


FIGURE 5.3 Application of a locally adaptive noise removal filter to a CT image of the lung, which contains a strong component of pseudostructure (A). The filtered image (B) shows good smoothing in homogeneous areas (lung, aorta), while the edges are retained. This is illustrated further in image C, the absolute difference of the original and filtered image, in other words, the component removed from image A by filtering. Dark regions in image C represent regions where the adaptive filter did not change the image. Most prominently, the lung contour can be seen as a dark line. Image D shows the application of conventional Gaussian blurring to image A. Although the noise largely been removed, the edges have deteriorated, and detail inside the lung has been lost.

the edge is straight. Computation of the local variance is then restricted to that side of the edge where the central pixel is located. Alternatively, the filter can be applied iteratively with an update of σ_N^2 after each iteration. Since σ_N^2 is reduced after each filter pass, the iterative application of the adaptive minimum mean-square filter converges.

5.1.2. Adaptive Bilateral Filter

Using a different formulation, the adaptive bilateral filter by Zhang and Allebach[42] also reduces noise while preserving edges. The adaptive bilateral filter is based on the convolution of the noisy image with two Gaussian kernels, one of which is adaptive.

```
set ngh=5;                          // Choose a suitable neighborhood size
set n = SQR(2*ngh+1);               // number of pixels in square neighborhood
set minvar = 1e+9;                  // some arbitrary high value for minimum variance
allocate AVGIMG(xmax, ymax);        // Allocate "image" for local average
allocate VARIMG(xmax, ymax);        // Allocate "image" for local variance

// Pass 1: Determine local average (=smoothed image), local variance, and minimum variance

for (y=0 while y<ymax increment y=y+1) do
   for (x=0 while x<xmax increment x=x+1) do
      sx=0; sxx=0;                  // Reset accumulators for sum of x and sum of x^2
      for (y1=y-ngh while y<=y+ngh increment y=y+1) do
         for (x1=x-ngh while x<=x+ngh increment x=x+1) do
            x2=x1;
            if (x2<0) then x2=0; else if (x2>=xmax) then x2=xmax-1;
            y2=y1;
            if (y2<0) then y2=0; else if (y2>=ymax) then y2=ymax-1;
            sx = sx + IM(x2,y2);            // Accumulate sum of x
            sxx = sxx + SQR(IM(x2,y2));     // Accumulate sum of x squared
         endfor
      endfor
      AVGIMG(x,y) = sx/n;                   // local average
      var = (sxx - sx*sx/n)/(n*(n-1));      // local variance
      if (var>0 and minvar > var) then
         minvar=var;                        // minimum variance only if variance defined
      endif;
      VARIMG(x,y)=var;                      // Store local variance
   endfor;
endfor;

// Pass 2: Compute filtered image

for (y=0 while y<ym increment y=y+1) do
   for (x=0 while x<xm increment x=x+1) do
      g = minvar / VARIMG(x,y);             // ratio of global/local variance
      if (g>1) then g=1;                    // force 0 <= g <= 1
      filt = IM(x,y)-g*(IM(x,y)-AVGIMG(x,y));   // This is the local adaptive filter step
      IMF(x,y) = filt;                          // Store result
   endfor;
endfor;

delete (VARIMG); delete (AVGIMG);
```

Algorithm 5.1 Adaptive minimum mean-square filter. The input image is `IM(x,y)`, and it is assumed that reading `IM` outside its defined area (0 to `xmax`-1 and 0 to `ymax`-1) returns defined values. The algorithm performs two passes. In the first pass, the local average, local variance, and the minimum variance are determined, and in the second pass, the actual filtering takes place, resulting in the filtered image `IMF(x,y)`.

The general filter equation is a convolution of the degraded (noisy) image $I(x,y)$ with a kernel $h(x,y,m,n)$ to obtain enhanced image I' through

$$I'(x,y) = \frac{1}{r(x,y)} \sum_{m=x-l}^{x+l} \sum_{n=y-l}^{y+l} h(x,y,m,n) I(m,n) \tag{5.6}$$

where $2l - 1$ is the square neighborhood size. The kernel h is computed through

$$h(x,y,m,n) = \exp\left[-\frac{(x-m)^2 + (y-n)^2}{2\sigma_d^2}\right] \exp\left[-\frac{[I(x,y) - I(m,n) - \zeta(x,y)]^2}{2\sigma_r^2}\right] \tag{5.7}$$

and the normalization factor $r(x,y)$ through

$$r(x,y) = \sum_{m=x-l}^{x+l} \sum_{n=y-l}^{y+l} h(x,y,m,n) \tag{5.8}$$

The convolution kernel contains two exponential terms that describe two Gaussian functions. The first exponential term is a two-dimensional Gaussian function with the standard deviation σ_d^2, and the second exponential term represents a Gaussian function of the image gradient. These exponential terms are referred to as the *domain* and *range filters*, respectively, and their combination was given the name *bilateral filter*. With a very large value of σ_r, the bilateral filter degenerates into a conventional Gaussian blurring filter with a standard deviation of σ_d. In the presence of edges, the second term causes a directional distortion of the Gaussian kernel: The kernel elongates along the edge, and averaging occurs predominantly along the edge and less strongly perpendicular to the edge. This behavior is the basis for the edge-preserving properties of the bilateral filter. σ_r is a variable parameter, which can be used to control the width of the filter. With large values of σ_r, the range filter will assign approximately equal weights for each pixel in the window, and the filter behaves predominantly like a Gaussian blurring filter. With small values for σ_r, the range filter dominates the bilateral filter and the adaptive behavior is more pronounced.

The final parameter, $\zeta(x,y)$, is responsible for the adaptive behavior. If $\zeta(x,y) = 0$, the filter is nonadaptive and corresponds to the conventional bilateral filter proposed by Tomasi and Manduchi.[37] To obtain adaptive properties, including sharpening of the edges, the maximum ($I_{\max}$), mean (I_{mean}), and minimum ($I_{\min}$) values of the pixel neighborhood need to be determined. The difference between any given pixel $I(x,y)$ and the neighborhood mean value I_{mean} can be defined as $\Delta(x,y) = I(x,y) - I_{\text{mean}}$. The behavior of the filter near edges is now determined by three preferred choices of $\zeta(x,y)$:

1. Shifting the image values in the neighborhood window toward I_{mean} by choosing $\zeta(x,y) = -\Delta(x,y)$ causes edges to blur, and the filter has an overall smoothing character.

2. Shifting the image values in the neighborhood window away from I_{mean} by choosing $\zeta(x,y) = +\Delta(x,y)$ causes noise reduction combined with edge sharpening.
3. Separating the image values and shifting them toward $I_{\max}$ and $I_{\min}$ by choosing $\zeta(x,y)$ according to the following equation depending on the sign of $\Delta(x,y)$ causes a drastic sharpening effect with a strong steepening of the edges:

$$\zeta(x,y) = \begin{cases} I_{\max} - I(x,y) & \text{for } \Delta(x,y) > 0 \\ I_{\min} - I(x,y) & \text{for } \Delta(x,y) < 0 \\ 0 & \text{for } \Delta(x,y) = 0 \end{cases} \tag{5.9}$$

The adaptive bilateral filter is best demonstrated by using a synthetic test image. Figure 5.4A shows some geometrical structures that were blurred and spoiled with additive Gaussian noise ($\sigma = 10$) (Figure 5.4B). Figure 4.4C demonstrates the effect of the conventional bilateral filter ($l = 9, \sigma_d = 2$, and $\sigma_r = 20$), which strongly reduces noise while not blurring the edges further. The adaptive bilateral filter, on the other hand, restores some of the edge sharpness (Figure 5.4D). With multiple applications of the adaptive bilateral filter, image restoration can be improved further (Figure 5.4E).

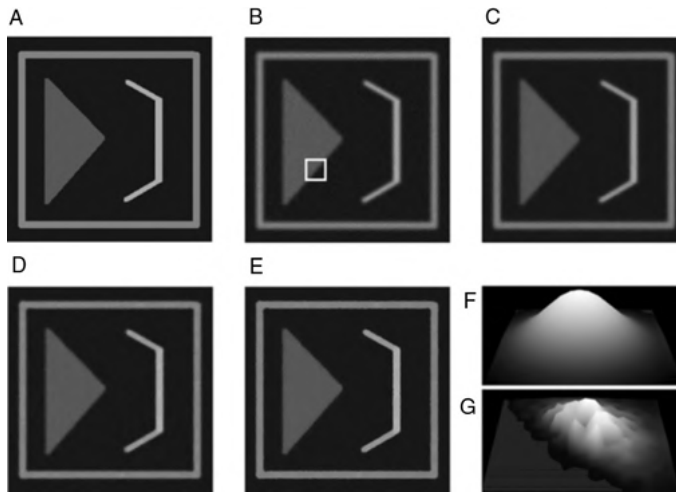


FIGURE 5.4 Demonstration of the operation of the adaptive bilateral filter. A test image (B) was created from a synthetic image (A) by performing Gaussian blur and adding Gaussian noise with a standard deviation of 10. The nonadaptive, conventional bilateral filter ($\zeta = 0$) already shows good noise suppression while retaining edge contrast (C). Application of the adaptive bilateral filter [$\zeta(x,y) = +\Delta(x,y)$] leads to image D, with similarly good noise suppression but an additional restoration of edge sharpness. Image E was created with two successive applications of the adaptive bilateral filter. Images F and G illustrate the composition of the kernel h: the white rectangle in image B indicates the position of the two kernels. Image F is the nonadaptive domain filter kernel, and image G is the adaptive bilateral kernel. It can be seen that the kernel contains zero values on one side of the edge, averaging only values on that side of the edge where the central pixel lies.

Figure 4.4F and G illustrate the kernel h. This kernel is computed at the location of the white rectangle in Figure 5.4B, and Figure 5.4F shows the Gaussian domain filter kernel, which is nonadaptive. The combination of the range and domain kernels h is shown in Figure 5.4G. It can be seen that the kernel becomes asymmetric along the edge as a consequence of the range filter: Pixels across an edge lose their influence on the kernel; therefore, the averaging takes place on pixels along one side of an edge.

A disadvantage of the adaptive bilateral filter is the empirical nature of the optimum values for σ_d and σ_r, and one of the main disadvantages is that the value of Δ, used to compute offset ζ in the range filter, is highly noise-dependent. Furthermore, the optimum choice of the parameters σ_d and σ_r are dependent on the gray-value range of the image. To optimize this filter, some experimentation is therefore needed. Even more advantageous is the use of a training set to optimize the filter parameters.[42]

5.1.3. Anisotropic Diffusion Lowpass Filter

Perona and Malik introduced a very powerful adaptive filter that operates by numerically simulating anisotropic diffusion.[28] Image intensity values can be thought of as local concentrations caught inside the pixel. If pixel boundaries are thought of as semipermeable, the pixel intensity would diffuse over time into neighboring pixels of lower intensity (i.e., follow the negative gradient). This process is most dominant wherever high contrast exists between neighboring pixels, and the diffusion process would end after infinite time with all contrast removed. This type of diffusion follows the partial differential equation

$$\frac{\partial I(x,y,t)}{\partial t} = c\Delta I(x,y,t) \tag{5.10}$$

where $I(x,y,t)$ is the image intensity, c is the diffusion constant, and Δ indicates the Laplacian operator. Equation (5.10) describes an isotropic diffusion process that leads to Gaussian blurring. The key to anisotropic diffusion is to introduce a locally adaptive diffusion constant $c(x,y,t)$ which changes the diffusion equation to

$$\frac{\partial I(x,y,t)}{\partial t} = \mathrm{div}(c(x,y,t)\,\nabla I(x,y,t)) = c(x,y,t)\,\Delta I(x,y,t) + \nabla c(x,y,t)\,\nabla I(x,y,t) \tag{5.11}$$

where ∇ indicates the gradient operator and div indicates the divergence operator. This diffusion constant needs to be chosen as a function of the local gradient[28] with a function g that is monotonically falling and normalized to $g(0) = 1$. An example of a suitable function g is

$$g(\nabla I(x,y)) = \frac{1}{1 + |\nabla I/K|^2} \tag{5.12}$$

By using g as a function of the local gradient, the diffusion speed $g(\nabla I) \cdot \nabla I$ becomes low at high local gradients (i.e., edges) and finds a maximum at—or, depending on the function g, near—the value of the constant K. The function g in Equation (5.12)

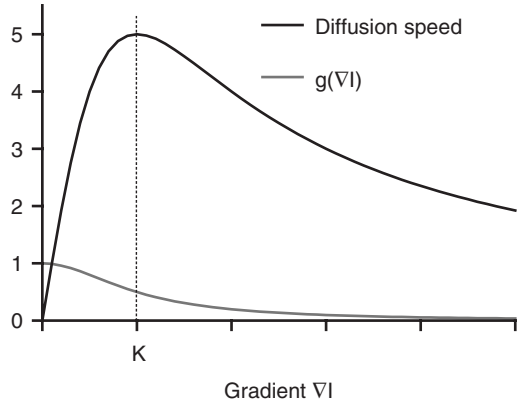


FIGURE 5.5 Graphical representation of the function $g(\nabla I)$ in Equation (5.12) and the diffusion speed $g(\nabla I)\cdot\nabla I$.

and the diffusion speed $g(\nabla I) \cdot \nabla I$ is shown qualitatively in Figure 5.5. The most interesting property of anisotropic diffusion, the enhancement of high-contrast edges, is related to the negative slope of the diffusion speed at high gradients.[28] Despite the comparatively complex theory behind the anisotropic diffusion filter, a numerical approximation of the diffusion equation, and therefore its computer implementation, are surprisingly simple. Let us discretize the intensity gradient into the intensity difference of a pixel to its four neighbors:

$$\begin{aligned}(\nabla I)_N(x,y) &= I(x,y-1) - I(x,y)\\ (\nabla I)_S(x,y) &= I(x,y+1) - I(x,y)\\ (\nabla I)_E(x,y) &= I(x+1,y) - I(x,y)\\ (\nabla I)_W(x,y) &= I(x-1,y) - I(x,y)\end{aligned} \tag{5.13}$$

Then the four directional components of the diffusion constant can be computed:

$$\begin{aligned}c_N(x,y) &= g((\nabla I)_N(x,y))\\ c_S(x,y) &= g((\nabla I)_S(x,y))\\ c_E(x,y) &= g((\nabla I)_E(x,y))\\ c_W(x,y) &= g((\nabla I)_W(x,y))\end{aligned} \tag{5.14}$$

Finally, the partial derivative toward time in Equation (5.11) can be discretized with an iterative time-stepping loop where one step advances the diffusion by the time Δt:

$$I(t+\Delta t) = I(t) + \Delta t\ [c_N(\nabla I)_N + c_S(\nabla I)_S + c_W(\nabla I)_W + c_E(\nabla I)_E]|_{\forall x,y} \tag{5.15}$$

If a function g is defined [Equation (5.12), termed `diffusion_g` in the computer implementation] that returns a floating-point value `g` from the floating-point parameters `grad_i` and `k`, we arrive at the implementation described in Algorithm 5.2.

```
set deltat=0.2;                    // Choose a suitably small time step size
set k = 100;                       // Choose a diffusion strength parameter K
allocate ITEMP(xmax, ymax);        // Allocate temporary storage for diffusion result

for (y=0 while y<ymax increment y=y+1) do
   for (x=0 while x<xmax increment x=x+1) do

      // Compute the four intensity gradients. Note boundary condition: the off-image intensity is zero.

      in=0; is=0; iw=0; ie=0;
      if (x>0) then iw = IM(x-1,y) - IM(x,y);
      if (x<xmax-1) then ie = IM(x+1,y) - IM(x,y);
      if (y>0) then in = IM(x,y-1) - IM(x,y);
      if (y<ymax-1) then is = IM(x,y+1) - IM(x,y);

      // Compute the anisotropic diffusion constant from given function g
      // Boundary condition: no diffusion off-image

      cn = 0; cw = 0; ce = 0; cs = 0;
      if (x>0) then cw = diffusion_g (iw, k);
      if (x<xmax-1) then ce = diffusion_g (ie, k);
      if (y>0) then cn = diffusion_g (in, k);
      if (y<ymax-1) then cs = diffusion_g (is, k);

      // Perform one time step in the iterative diffusion process

      ITEMP(x,y) = IM(x,y) - deltat * ( cn*in + cs*is + ce*ie + cw*iw );

   endfor
endfor

// Finally, copy ITEMP into IM to simulate in-place filtering

for (y=0 while y<ym increment y=y+1) do
   for (x=0 while x<xm increment x=x+1) do
      IM(x,y) = ITEMP(x,y);
   endfor;
endfor;

delete (ITEMP);
```

Algorithm 5.2 Anisotropic diffusion lowpass filter. Each execution of this algorithm performs one time step according to Equation (5.15). The input image is `IM(x,y)`, and the diffused image is stored temporarily in `ITEMP(x,y)`. Afterward, `ITEMP` is copied into `IM` so the diffusion is applied to the input image. This process needs to be repeated several times to achieve sufficient noise reduction.

The parameter K, which determines the strength of the filter by determining where the negative diffusion gradient starts, and the number of iterations strongly determine the outcome of the filter. An optimum value of K and the optimum number of iterations are normally determined experimentally, although K can be made dependent on the noise component of the image,[28] for example, by computing a histogram of the image gradient magnitude at each time step and selecting K to be the 90% quantile of the histogram. The main advantage of computing K in dependency of the noise component is the fact that one less parameter is arbitrarily chosen. Furthermore, as the noise component is reduced at higher iterations, K diminishes, and the filter strength diminishes as well. As a consequence, the number of iterations becomes less critical in determining the outcome. The automated selection of K is a major step toward an unsupervised anisotropic diffusion filter. The influence of the parameter K and the number of iterations are demonstrated in Figure 5.6.

A further option to fine-tune the filter is the choice of the function g. An alternative choice, the function $g(\nabla I) = \exp(-\nabla I^2/K^2)$, exhibits a steeper drop-off at higher gradients and therefore preserves smaller detail than the function given in Equation (5.12). Finally, an improvement in the directional behavior can be achieved by including the diagonal neighbors in the diffusion equation.

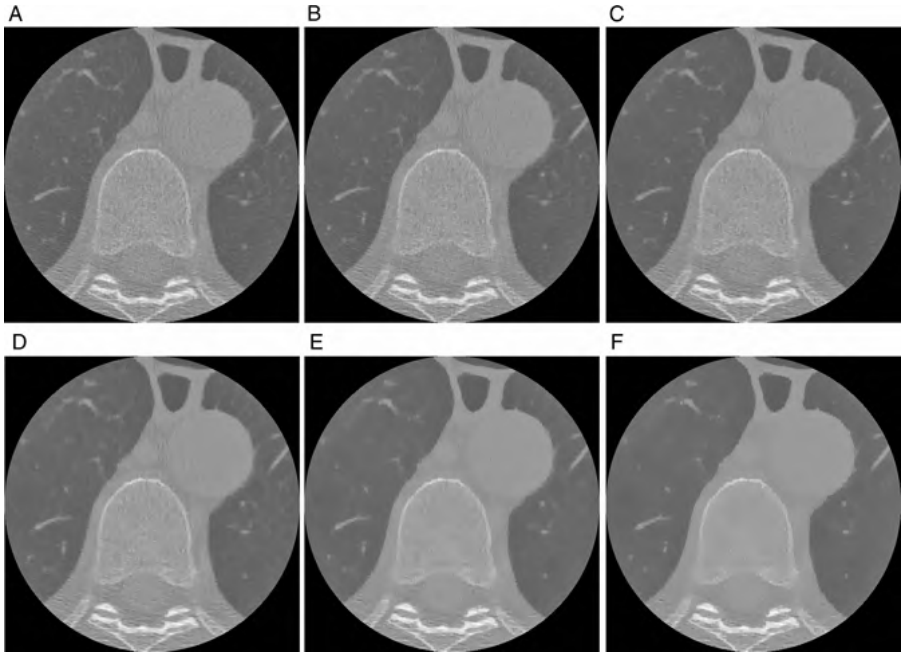


FIGURE 5.6 Application of the anisotropic diffusion lowpass filter on the CT slice in Figure 5.3A. For the top row (A, B, C), the parameter K was set to 50 and the diffusion process was iterated 20, 50, and 100 times, respectively. In the bottom row (D, E, F), K was set to 100 and the diffusion process was iterated 20, 50, and 100 times, respectively. At a high number of iterations, the edge-preserving property of the filter becomes particularly apparent.

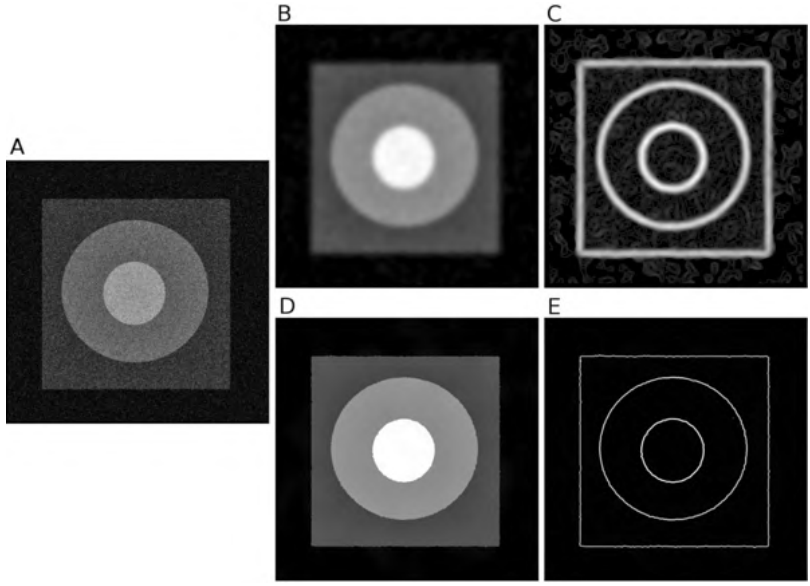


FIGURE 5.7 Application of anisotropic diffusion to denoise an image. The original image (A) consists of several areas of different gray values (range 0 to 255) with step transitions and superimposed Gaussian noise with $\sigma = 40$. Repeated application of strong Gaussian blurring leads eventually to image B, where noise is widely smoothed, but the transitions are blurred, too. As a consequence, the application of an edge detector (Sobel operator) on image B yields fairly broad edges with considerable background texture (C). Anisotropic diffusion (10 iterations, $K = 50$ followed by 300 iterations, $K = 12$) not only removes the noise component completely but also preserves the step edges (D). The application of an edge detector on image D results in sharp edges that are only minimally distorted (E).

An example of the application of anisotropic diffusion in noise removal is shown in Figure 5.7, where conventional nonadaptive Gaussian blurring is compared with anisotropic diffusion. With extremely noisy images, such as Figure 5.7A, a suitable denoising approach needs to be determined experimentally. In this example, two stages of anisotropic diffusion were applied. For the first stage, $K = 50$ was selected but only 10 iterations were executed. This value of K is large enough to blur the boundary of the dark rectangle, but blurring is limited because of the low number of iterations. However, with the large value of K, the initial diffusion step considerably reduced the noise component. In the second step, a smaller value for K was chosen ($K = 12$), which prevented blurring of the lowest-contrast edge, and with the high number of iterations, the remaining noise was removed almost completely.

5.1.4. Adaptive Median Filters

Although averaging filters are effective for removing Gaussian noise, the median filter is more suitable for *shot noise* (also known as *salt-and-pepper noise* or *impulse*

noise). Shot noise is characterized by pixels that assume the maximum or minimum image value with a certain probability. Images affected by shot noise typically occur when individual pixels are not covered in the image formation or reconstruction process. For comparison, additive noise affects every pixel by displacing its value by a random offset ϵ. An image degraded by Gaussian noise can be modeled as

$$g(x,y) = f(x,y) + \epsilon(x,y) \tag{5.16}$$

where $g(x,y)$ is the degraded image, $f(x,y)$ is the ideal image, and $\epsilon(x,y)$ is the Gaussian noise component with zero mean and an image-specific standard deviation.

On the other hand, shot noise affects pixels only with a probability p, but an affected pixel will either assume the image's maximum or minimum value. In 8-bit images, these pixels would be black [$g(x,y) = 0$] or white [$g(x,y) = 255$]. The degradation process can be modeled as

$$g(x,y) = \begin{cases} f(x,y) & \text{with probability } 1 - p \\ n(x,y) & \text{with probability } p \end{cases} \tag{5.17}$$

where $n(x,y)$ is either black, or white, or a combination of black and white pixels with a different probability p_2 for a black pixel. Therefore, an image that is affected by shot noise contains a number of white and black pixels.

A conventional, nonadaptive median filter sorts the values of a $n \times n$ neighborhood (most often a 3×3 neighborhood) and replaces each image value by the median of its neighborhood, thus eliminating runaway values such as black or white pixels. A locally adaptive median filter has been proposed[38] in which the neighborhood is expanded as long as the median value is typical for its neighborhood. More specifically, for each pixel, a starting neighborhood of $n \times n$ is set: for example 3×3. For this neighborhood, the minimum, median, and maximum intensities ($I_{\min}$, I_{med}, and $I_{\max}$, respectively) are determined and the condition $I_{\min} < I_{\text{med}} < I_{\max}$ is tested. If this condition is not satisfied, usually with either $I_{\min} = I_{\text{med}}$ or $I_{\text{med}} = I_{\max}$ (thus indicating that a likely runaway value exists in the neighborhood), the neighborhood is expanded. If a predetermined maximum neighborhood size is reached, the pixel remains unaffected; otherwise, the process is repeated. If, on the other hand, the condition $I_{\min} < I_{\text{med}} < I_{\max}$ is satisfied, another test is performed; that is, $I_{\min} < I(x,y) < I_{\max}$. If this condition tests true, the central pixel $I(x,y)$ is unlikely to be impulse noise, and the pixel remains unaffected. Otherwise, the central pixel $I(x,y)$ is probably impulse noise and gets replaced by I_{med}. For large neighborhood sizes, the adaptive median filter retains edges and details better than a conventional median filter.

The center-weighted median filter (CWMF) was proposed by Ko and Lee[18] as a weaker median filter with better edge-preserving properties than those of the conventional median filter. This attenuated median filter differs from the conventional median filter inasmuch as the neighborhood table, from which the median value is determined, is extended by $w - 1$ values and the central pixel value is repeated w times. An example is given in Figure 5.8. Depending on the value of w, the CWMF is

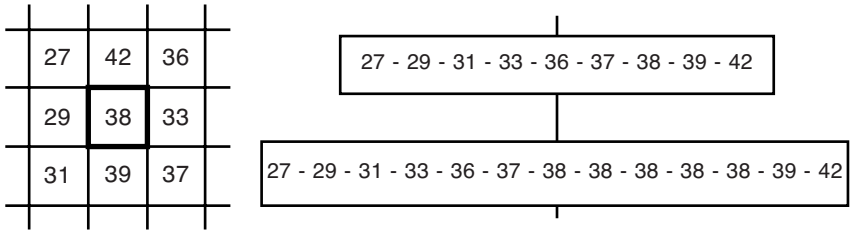


FIGURE 5.8 Center-weighted median filter (CWMF). Shown is a sample neighborhood (left) and the nine neighborhood values sorted ascendingly (right, top). The median value of the neighborhood is 36. The CWMF extends the neighborhood list by repeating the central value $w = 4$ times (right, bottom). In this example, the central pixel value and the median value coincide. The CWMF is less aggressive than the conventional median filter.

less likely to replace a pixel by the neighborhood median value than by the conventional median filter. For this reason, the CWMF preserves edges, corners, and lines better than the conventional median filter does at the expense of lower filtering power. A locally adaptive combination of the CWMF and the median filter was proposed by Chen et al.[11] and termed the *tristate median filter*, because the filter either replaces a pixel by the output of the median filter or by the output of the CWM filter, or leaves the pixel unchanged, depending on the local neighborhood properties.

More specifically, for each pixel $f(x,y)$, the median filter output $f^{\text{med}}(x,y)$ and the CWMF output $f^{\text{CWMF}}(x,y)$ are determined. A threshold value T is provided for the decision mechanism, and the final filtered image value $g(x,y)$ is determined through

$$g(x,y) = \begin{cases} f(x,y) & \text{for } T \geq d_1 \\ f^{\text{CWMF}}(x,y) & \text{for } d_2 \leq T < d_1 \\ f^{\text{med}}(x,y) & \text{for } T < d_2 \end{cases} \tag{5.18}$$

where $d_1 = |f(x,y) - f^{\text{med}}(x,y)|$ and $d_2 = |f(x,y) - f^{\text{CWMF}}(x,y)|$. The threshold value T determines the strength of the filter. Setting $T = 0$ leads to the conventional median filter. A very large threshold value leaves the image unmodified. Threshold values in between yield the adaptive behavior, and a smaller threshold leads to a stronger filter action. The choice of a suitable threshold value needs to be determined experimentally, ideally with a training set of images where the mean-squared error (or any other distance) is minimized.

The idea of adaptive median filters can be taken further to allow for a correction step between two successive applications of a median filter. The principle behind the correction of a median filter step lies in the filter's possible types of error: A median filter can either miss a corrupted pixel, correct a noncorrupted pixel, or replace a pixel by a new value that has a larger distance to the (unknown) uncorrupted pixel than the original corrupted pixel. This third type of error, referred to as *overcorrection error*, is minimized by the filter proposed by Xu et al.[41] The first step of this two-stage filter is the application of a conventional median filter, a CWM filter, or any other

adaptive rank filter such as the adaptive median filters described in previous sections. In preparation of the corrective step, a binary image is created, which contains white pixels (1's) where the original image and the median-filtered image differ, and which contains black pixels (0's) where they don't. Let us denote the input image **I**, the median filter output **F**, and the black-and-white image that indicates pixels that were replaced by the median filter **D**. To identify those pixels of **D** that have probably been overcorrected, an adaptive processor is proposed that creates another image **E**, which is a subset of **D** and contains 1's only for those pixels that are identified as overcorrected. The adaptive processor operates row by row and processes one row of **D** as follows:

Step 1. Choose two parameters a and b that determine the filter action.

Step 2. Count the probability of white pixels in all rows n. Denote this probability $w(n)$. $w(n)$ is therefore a column vector, and the mean and standard deviation of $w(n)$, denoted $\mu(w)$ and $\sigma(w)$, can be computed.

Step 3. Compute two threshold values $\eta = a\sigma(w)$ and $\kappa = \mu(w) - b\sigma(w)$. In addition, copy the first median-filtered image **F** into **G**.

Step 4. For each row n, execute steps 5, 6, and 7 only if $w(n) > \mu(w) + \eta$.

Step 5. For row n, determine the number K of pixels probably overcorrected using $K = w(n) - \mu(w) + b\sigma(w) = w(n) - \kappa$.

Step 6. In row n, compute the error vector $\boldsymbol{e}$ as the element-by-element squared difference of row n in **I** and **F**.

Step 7. Find the K smallest values in $\boldsymbol{e}$ and revert the action of the median filter for the corresponding elements in **G** (i.e., replace the corresponding elements in **G** by the original elements from **I**).

Step 8. After all rows have been processed, compute **E** as the binarized difference image between **F** and **G** in the same manner as **D** was computed from **F** and **I**. In this way, **E** contains 1's on all positions where the action of the first median filter was reverted.

In the second stage of this filter, **G** is subjected to a conditional median filter step: A pixel in **G** may be changed as a result of the median filter only if the corresponding element of **E** is zero (in other words, if the pixel was not reverted by the adaptive processor). The output of the conditional median filter step **G** is the output of the entire adaptive filter.

The filter algorithm can further be simplified if the reversal of overcorrected pixels takes place directly in **D** and **D** serves as decision mechanism whether to use pixels from **I** or **F** in the final filtered image. In this case, **E** and **G** are no longer needed. Although the filter as proposed by Xu et al.[41] preserves some detail over the nonadaptive median filter, its line-by-line processing sequence makes the filter action highly orientation-dependent. Furthermore, the optimum parameters a and b are not intuitively easy to find. The filter could be improved by examining the median filter correction statistics in a square local neighborhood rather than in a horizontal line.

5.2. ADAPTIVE FILTERS IN THE FREQUENCY DOMAIN: ADAPTIVE WIENER FILTERS

Frequency-domain filtering was introduced in Chapter 3. One particularly important filter is the Wiener filter which finds its main application in image restoration (Section 3.2.3). Let us recall the general equation of the Wiener filter in the frequency domain:

$$W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + S_\epsilon(u,v)/S_f(u,v)} \tag{5.19}$$

Here, $H(u,v)$ is the Fourier transform of the point-spread function of the degradation process $h(x,y)$ and $H^*(u,v)$ is its complex conjugate. $S_\epsilon(u,v)$ is the power spectrum (i.e., the squared magnitude of the Fourier transform) of the noise, and $S_f(u,v)$ is the power spectrum of the ideal image $f(x,y)$. A special case exists where blurring with the point-spread function is negligible [i.e., $H(u,v) \approx 1$] and Equation (5.19) simplifies to

$$W(u,v) = \frac{S_f(u,v)}{S_f(u,v) + S_\epsilon(u,v)} \tag{5.20}$$

Conversely, if the noise component is negligible, the filter in Equation (5.19) simply becomes the reciprocal of the point-spread function, $W(u,v) = 1/H(u,v)$. However, dividing the degraded image by the degradation function in the frequency domain is impractical because the degradation function $H(u,v)$ generally has lowpass characteristics (i.e., the magnitude drops off toward higher frequencies). Therefore, its reciprocal, $W(u,v)$, assumes very large magnitudes at higher frequencies. This may lead to high frequencies of the degraded image, $G(u,v)$, being divided by very small numbers or even by zero. Since noise has a broadband spectrum, the contribution of $S_\epsilon(u,v)$ to the denominator of Equations (5.19) and (5.20) prevents $W(u,v)$ from assuming very large values at high frequencies. In fact, a higher noise component reduces the high-frequency magnitude of $W(u,v)$, therefore giving the filter a more lowpass character. Conversely, if the noise component is small in Equation (5.19), the blurring term $H(u,v)$ dominates and the filter assumes more highpass characteristics. In most practical cases, the power spectrum $S_f(u,v)$ is unknown, and the spectrum of the noise component $S_\epsilon(u,v)$ is also unknown. Most frequently, the power spectrum ratio $S_\epsilon(u,v)/S_f(u,v)$ is therefore replaced by a constant k and the Wiener filter simplifies to

$$W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + k} = \frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + k} \tag{5.21}$$

The constant k is determined experimentally for each case. A large value of k, necessary when a large noise component is present, configures the filter to act more as a lowpass, while a small value of k, allowable in images with low noise component, configures the filter as a highpass (see Figure 3.11). Two adaptive variations of this filter are possible, a frequency-adaptive form where k becomes a function of

frequency, $K(u,v)$, and a spatially adaptive form where k becomes locally dependent as $k(x,y)$.

If the statistical properties of the noise are spatially variable, $S_\epsilon(u,v)$ in Equation (5.20) becomes a function of the spatial coordinate (x,y). In the simplified form [Equation (5.21)], the constant k would become spatially dependent and the noise-reducing Wiener filter can be formulated as

$$W(u,v) = \frac{S_f(u,v)}{S_f(u,v) + \sigma_n^2(x,y)} \tag{5.22}$$

where $\sigma_N^2(x,y)$ is the local variance of the noise component. The application of this spatially adaptive Wiener filter is computationally extremely expensive, because the inverse Fourier transform has to be performed for each pixel. It can be shown, however, that the adaptive minimum mean-square filter in Equation (5.3) is a very efficient spatial-domain implementation of the adaptive Wiener filter in Equation (5.22).[40]

This example explains the main challenge for the design of adaptive frequency-domain filters. Since spatial information is lost in the frequency domain, local properties can only be considered with a separate inverse transform for each region of the image where the statistical properties differ. Such a filter is computationally inefficient. However, there are alternative approaches. Abramatic and Silverman[1] proposed a parametric Wiener filter,

$$W(u,v) = \frac{S_f(u,v)}{S_f(u,v) + \gamma\sigma_n^2(u,v)} \tag{5.23}$$

where γ is a function to be optimized according to specific criteria. A nonadaptive version of this filter is obtained by minimization of the difference between the idealized, noise-free image and the restored image (Backus–Gilbert approach[3]). It can be seen that with $\gamma \to 0$, the filter becomes the identity filter (a desired behavior near edges), whereas for $\gamma \to 1$ the filter approaches the conventional Wiener filter in Equation (5.22). The Backus–Gilbert approach still requires γ to be a function of location, $\gamma(x,y)$, with the associated inverse Fourier transform necessary for each pixel x,y. Abramatic and Silverman propose to use a function for $\gamma(x,y)$ that falls monotonically from 1 to 0 as the local image gradient increases from 0 to ∞. Furthermore, it is possible to consider directionality of the gradient (e.g., the four directions of the compass operator) to formulate an anisotropic function $\gamma(x,y)$. The effect of such a filter would be related to the adaptive bilateral filter in Equation (5.6). The main reason to pursue a frequency-domain approach with a Wiener filter is the fact that the Wiener filter is the optimum filter for known image and noise variance. If either is unknown, an empirical approach such as the minimum mean-squared-error filter or the adaptive bilateral filter will lead to satisfactory results with less computational effort.

Using the same basic idea, Guy[17] recently proposed a Fourier-based approach to minimize Poisson noise in scintigraphic images. Poisson noise is a special case of multiplicative, intensity-dependent noise, where the standard deviation is approximately equal to the mean intensity value. The algorithm, termed *Fourier block noise*

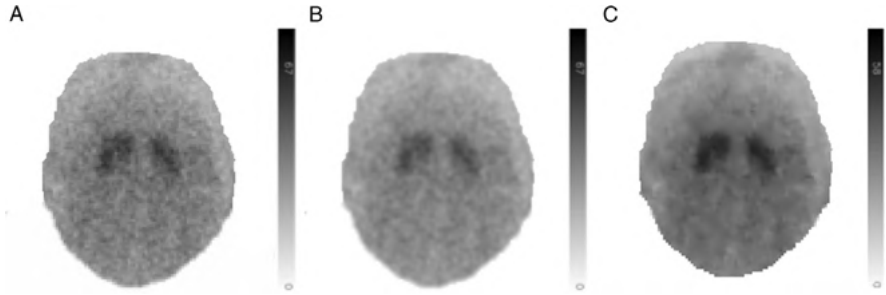


FIGURE 5.9 Fourier block noise reduction filter to remove Poisson noise. Shown is a dopamine transporter imaging scan, that is, a SPECT scan with specific radiopharmaceuticals, in one slice of the brain (A). Since the radiopharmaceutical itself is the source of the events recorded to form an image, the signal is collected from random decay events. Therefore, a higher signal is always associated with a larger noise component (Poisson noise). Gaussian smoothing (B) blurs the image to an extent where details vanish and contrast becomes low. The Fourier block noise reduction filter (C), applied with 4×4 blocks on a 128×128 pixel image, smoothes noise while preserving details such as the ventricles. (From ref. 17, courtesy of Dr. Matthew Guy.)

reduction, uses a moving-window Fourier filter. For each pixel, a square local window is specified in which the local variance $\sigma^2_{\text{loc}}(x,y)$ is computed in the spatial domain [Equation (5.4)]. The local noise component, $\sigma^2_N(x,y)$ can be estimated from the Poisson model to be approximately equal to the local mean intensity. Now, the local window is subjected to a Fourier-domain lowpass with variable cutoff frequency, and the cutoff frequency is increased in repeated steps. After each step, the residual noise of the filtered block, $\sigma^2_R(x,y)$, is computed by using Equation (5.4), and the algorithm halted when $\sigma^2_R(x,y) \leq \sigma^2_N(x,y)$ in the local neighborhood selected. With variable step size for the cutoff frequency, an improvement in computational efficiency can be achieved. One of the advantages of this filter is the application of the Fourier transform to a local neighborhood, which increases overall computational efficiency over the computation of the Fourier transform of the entire image for each pixel as proposed in the adaptive Wiener filter implementations. The other advantage is the optimized removal of the Poisson noise component without additional filter parameters, which makes it suitable for unsupervised image processing. An example of the filter applied to a SPECT image of the brain is given in Figure 5.9.

5.3. SEGMENTATION WITH LOCAL ADAPTIVE THRESHOLDS AND RELATED METHODS

Thresholding is one of the most common segmentation methods. The underlying assumption is that the feature (foreground) pixel brightness differs from the background pixel brightness. This behavior is usually evidenced by a bi- or multimodal

histogram, where one peak represents the background and one (or more) peaks represent the features (see Section 2.4). If the image is illuminated homogeneously, the valley that separates the background peak from the other peaks is the ideal threshold value for the segmentation. This assumption does not hold for images with inhomogeneously illuminated background. More specifically, if the intensity difference between background and features is smaller than the intensity variations of the background itself, thresholding fails. Consider the example in Figure 5.10, where several features are present over a strongly inhomogeneous background with the highest brightness in the center. No consistent segmentation threshold exists, because some features are darker than the brightest part of the background. Yet the human eye can easily identify the features. In many cases it is possible to flatten the background by using conventional methods such as unsharp masking or homomorphic filtering. However, when the features are relatively large, the filtering steps affect the features by reducing the intensity of large homogeneous areas. Furthermore, to some extent, filters used to remove the background have highpass characteristics and therefore amplify noise (Figure 5.10C).

A related aspect is the efficacy of automated thresholding methods. It is possible to use methods such as the isodata method[33] or Otsu's method[27] to find the optimum threshold in an image with a bimodal histogram. The application of filters can distort the histograms to a point where automated thresholding methods no longer find the optimum threshold. Figure 5.11 shows three histograms that correspond to the three examples in Figure 5.10. In Figure 5.11A, the background and feature peaks are clearly separable, and Otsu's method reliably finds a suitable threshold (arrow). Conversely, the strongly inhomogeneous background in Figure 5.10B causes the broad background peak in Figure 5.11B, in which Otsu's method can no longer detect a meaningful threshold. Restricting the area in which histograms are analyzed in Figure 5.10B again yields histograms with a clear valley between background and feature peaks. In the example of Figure 5.11C, two different regions were analyzed,

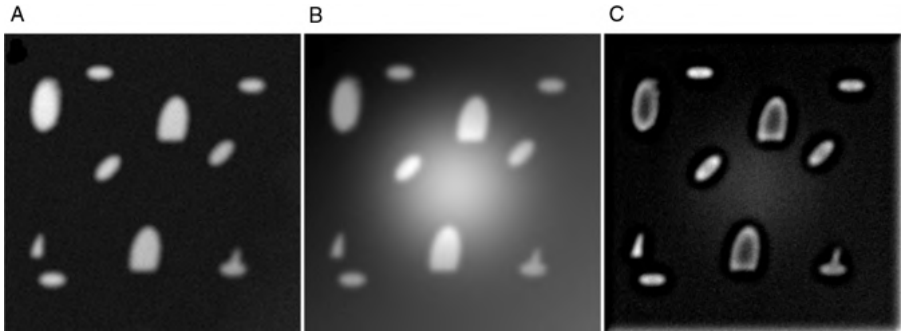


FIGURE 5.10 Example images of several features over relatively homogeneous (A) and strongly inhomogeneous (B) background. In example B, no global threshold can be found to separate the features from the background (e.g., the large feature top left is darker than the background in the center). Restoration by application of a highpass filter (C) affects the features themselves by reducing the intensity of large homogeneous areas inside the features.

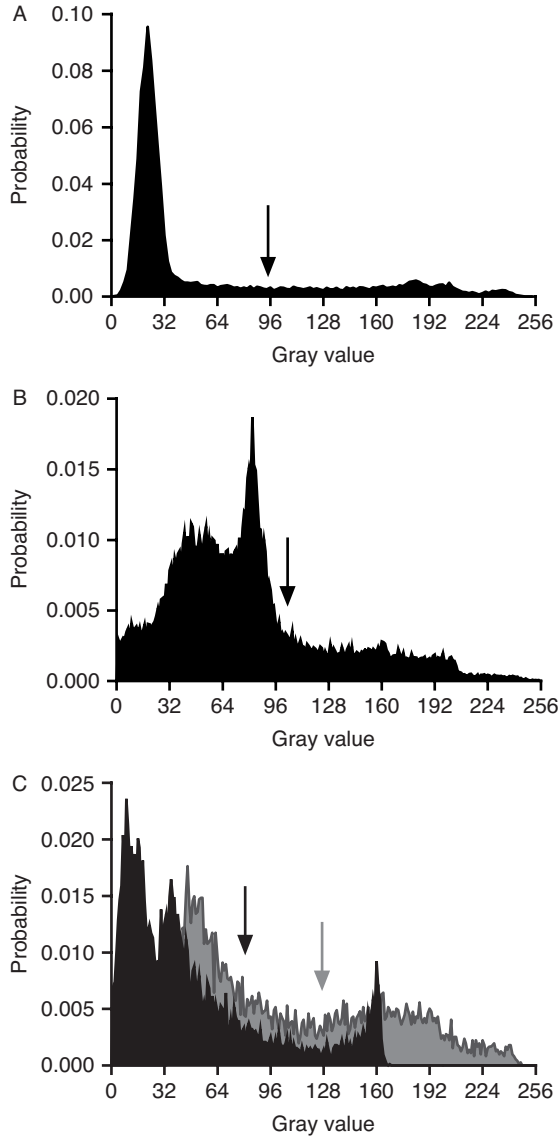


FIGURE 5.11 Histograms corresponding to the three examples in Figure 5.10. Arrows indicate the optimum threshold by Otsu's method. In part C the gray and black histograms represent two different regions.

and their different overall brightness is reflected by a local shift in Otsu's threshold. In the simplest implementation of a locally adaptive threshold, the histogram in a window around each individual pixel is determined and subjected to a threshold finding method (e.g., Otsu's method), which determines whether the pixel is assigned to the background or foreground. The local threshold can be recorded as a function

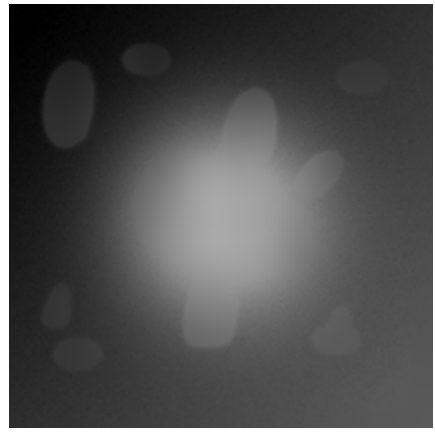


FIGURE 5.12 Threshold map for the image in Figure 5.10B. The adaptation-dependent increase in the threshold value in the image center can be seen clearly.

of the image location and examined independently. Figure 5.12 shows an example for a threshold map which corresponds to Figure 5.10B. It shows clearly an increase in the threshold toward the image center, caused by the increased local average, and it shows that the threshold is approximately homogeneous over the features. A threshold map does not need to be generated pixel by pixel as in the example of Figure 5.12. Interpolation or least-squares methods can be used alternatively to create smoother threshold maps. An example is shown in Figure 5.13, which corresponds to Figure 5.12. To arrive at the smooth threshold map in Figure 5.13, the original image (Figure 5.10B) was subdivided into 36 (6×6) tiles, and in each tile, the pixel

FIGURE 5.13 Threshold map created by performing a parabolic least-squares fit into the local gray-value minima of 6×6 tiles into which the image was divided.

with the minimum intensity was located. A parabolic intensity function, $I_B(x,y) = a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2$, was chosen and the unknown parameters a_0 through a_5 were determined using a least-squares fit. This intensity function, evaluated for each pixel, resulted in the threshold map in Figure 5.13. In the special case of Figure 5.10B, the smooth threshold map with its fairly rigid parabolic function is not able consistently to separate the features from the background, as opposed to Figure 5.12. The type of map that shows superior performance needs to be determined experimentally for each application. For example, Ball et al.[4] subdivided digitized mammography x-ray images into 16×16 nonoverlapping tiles and determined the intensity mean and standard deviation for each. Through interpolation, matching 512×512 maps of the local threshold and the local standard deviation were created and used as joint threshold maps. The use of threshold maps is functionally similar to background flattening. It is either possible to use the threshold map in Figure 5.12, or the image in Figure 5.12 can be subtracted from Figure 5.10B to allow a nonadaptive threshold to be used.

The window size plays an important role, because it must be larger than the features and it must be small enough to include large background inhomogeneities. Choosing a window that is smaller than the features will misclassify feature pixels. Conversely, a window that is too large would not only be affected by the inhomogeneous background but would also misclassify small features with relatively low contrast to the background, especially if the small feature is located near a larger feature. These requirements may conflict, and thus more advanced adaptive threshold segmentation methods have been developed.

Local connectedness can become an important additional criterion for segmentation. A combination of adaptive thresholding and region growing is often referred to as *hysteresis thresholding* (Section 2.4.4). The algorithm for hysteresis thresholding is a two-pass process, where thresholding with a relatively high threshold value is used to identify high-probability foreground pixels. In the second pass, the local neighborhood of the initial pixels identified is region-grown with a lower threshold value. By considering the neighborhoods of the initial seed pixels in a region-growing fashion, local connectedness is observed. The hysteresis thresholding algorithm can be combined with adaptive thresholds. It is possible to use an adaptive threshold map for either the upper or lower threshold, or both.

Up to this point, a constant neighborhood size was assumed. However, the window size may become adaptive as well. For example, an iterative variable-window adaptive threshold algorithm was proposed by Bieniecki and Grabowski.[7] For this multistep segmentation, the image is pre-thresholded based on an initial global threshold, such as the image mean or Otsu's threshold. In this first pass, each pixel is therefore classified as either foreground or background. The window size is initially set to 1×1 (a single pixel). Iteratively, the window size is doubled, and any background pixel is reexamined based on local statistics, upon which it can be reclassified as a feature pixel. The iteration ends at a predetermined window size or when no more pixels have been reclassified. One example of local statistics as suggested by Bieniecki and Grabowski[7] is the Bernsen threshold,[6] where the threshold is the mean of the darkest and brightest pixels in the neighborhood. However, using single pixels for

a decision, such as threshold selection, makes the algorithm sensitive to noise. One possible alternative would be iterative threshold selection[33] (see Section 2.4), which considers all pixels in the current window. In addition, threshold selection comparable to nonadaptive thresholds (e.g., image mean or Otsu's threshold) would increase the probability of classifying too many pixels as foreground pixels. Since the iterative reclassification in this algorithm always increases the number of foreground pixels, a higher initial threshold is desirable. These considerations would lead to a hybrid algorithm combining adaptive window size, local iterative threshold selection, and hysteresis thresholding.

The window size does not need to be adapted in a predetermined fashion. Instead, the local window can be grown on the basis of the local gray-level distribution. An example for an algorithm with variable local neighborhood size is presented as one step to segment microscopic images of cells.[23] The classification of a pixel as belonging to a feature (cell) or to the background is based on the local application of Otsu's method. In this example, the size of the neighborhood is grown until the neighborhood variance meets the global variance. Intuitively, it can be assumed that the neighborhood has grown to meet adjoining different regions. From that neighborhood, the histogram is computed, which serves as the local basis for Otsu's method.

A related algorithm is multitolerance region growing.[34] Multitolerance region growing is an adaptive variant of region growing and allows a pixel $I(x,y)$ to be added to the existing region if it satisfies the condition

$$(1+\tau)\frac{I_{\max}+I_{\min}}{2} \geq I(x,y) \geq (1-\tau)\frac{I_{\max}+I_{\min}}{2} \tag{5.24}$$

where $I_{\max}$ and $I_{\min}$ are the maximum and minimum intensities in the region at the present iteration, and τ is a user-selectable value between 0 and 1. In its application to detect microcalcifications in digitized mammograms,[34] however, the parameter τ is not chosen by the user but rather is determined automatically by analyzing descriptive parameters of the grown feature. A good choice of τ is found when variations of the feature classifiers over the region-growing process become very small. With the underlying idea of using adaptive neighborhoods, Dhawan and Le Royer developed a method of targeted contrast enhancement in x-ray mammographies to emphasize specific features, including microcalcifications.[13] The key step in the image processing chain proposed is the computation of local contrast information based on adaptive neighborhoods. For each pixel $I(x,y)$, the first criterion for classifying the neighboring pixels as "center" or "surround" (i.e., feature and background) is gray-value similarity. Any neighborhood pixel $I(x+\Delta x, y+\Delta y)$ is classified as center if its intensity meets the condition $I(x,y) - T \leq I(x+\Delta x,\ y+\Delta y) \leq I(x,y) + T$, where T is a threshold value, in this study set to $T = 3$ in an 8-bit digitized image. Any neighborhood pixel is classified as surround if it does not meet the intensity criterion and is at the same time 8-connected to at least one center pixel. Note that pixels outside the threshold window that are not 8-connected to the center count as unclassified. At this time, a second adaptive criterion comes into play. Pixels are counted in the outer row of the square region. For example, if the window is 5×5 pixels, the

only pixels counted are those for which $\Delta x = \pm 2$ or $\Delta y = \pm 2$. The number of center pixels as a percentage of the total pixels must drop below a percentage threshold, which in this study was set at 40%. If this criterion is not met, the square window surrounding the center coordinate (x,y) is grown by one pixel (e.g., from 3×3 to 5×5 to $7 \times 7 \ldots$) and the classification as center or surround is repeated. Once the percentage criterion is met, the local contrast C for pixel (x,y) can be computed as

$$C(x,y) = \frac{|\bar{I}_c(x,y) - \bar{I}_s(x,y)|}{\max[\bar{I}_c(x,y), \bar{I}_s(x,y)]} \tag{5.25}$$

where $\bar{I}_c$ and $\bar{I}_s$ are the mean intensity values of the pixels classified as center and surround, respectively. Thus, a contrast image corresponding to the input image can be created. The values of the contrast image lie between 0 and 1. A pixel-by-pixel transform, $C'(x,y) = g(C(x,y))$, serves to enhance the contrast. The function g is a monotonically increasing function with $g(0) = 0$ and $g(1) = 1$. Dhawan and Le Royer propose a piecewise exponential function, but it appears that functions related to the gamma function $g(c) = c^{1/\gamma}$ with $\gamma > 1$ are best suited for the purpose of contrast enhancement. In the last step, each pixel of the original image is transformed based on

$$I'(x,\, y) = \begin{cases} \dfrac{\bar{I}_s(x,y)}{1 - C'(x,y)} & \text{for } \bar{I}_c(x,y) \geq \bar{I}_s(x,y) \\ \bar{I}_s(x,y)[1 - C'(x,y)] & \text{for } \bar{I}_c(x,y) < \bar{I}_s(x,y) \end{cases} \tag{5.26}$$

which results in a contrast-enhanced image enhanced by local adaptive contrast criteria.

A fundamentally different idea for determining a locally adaptive threshold is the use of diffusion, analogous to the anisotropic diffusion presented in Section 5.1. However, as the basis for thresholding, diffusion normal to image gradients was proposed by Manay and Yezzi.[22] Diffusion parallel to the edges smoothes noise but retains the edges. Diffusion normal to the edge direction, on the other hand, blurs the edges, and diffusion is strongest at the steepest gradients. The diffusion equation, introduced in Equation (5.10), can be made anisotropic in a tangential sense and reduces to

$$\frac{\partial I}{\partial t} = \frac{I_x^2 I_{xx} + 2 I_x I_y I_{xy} + I_y^2 I_{yy}}{I_x^2 + I_y^2} \tag{5.27}$$

where I is the pixel intensity and the indices x and y indicate a partial derivative toward one direction. Equation (5.27) was termed by the authors *antigeometric heat flow*. The implementation of differential equation (5.27) is possible with the discretization in time described by

$$I^{k+1}(x,y) = I^k(x,y) + \Delta t\, I_R^k(x,y) \tag{5.28}$$

which leads to the iteration step $k + 1$ by advancing the time-dependent intensity at iteration k by Δt. Here $I_R(x,y)$ is a term that contains the spatial derivatives and is defined as

$$I_R(x,y) = \frac{I_x(x,y)^2 I_{xx}(x,y) + 2I_x(x,y)I_y(x,y)I_{xy}(x,y) + I_y(x,y)^2 I_{yy}(x,y)}{I_x(x,y)^2 + I_y(x,y)^2} \tag{5.29}$$

where the partial derivatives may be implemented through central differences as described by

$$\begin{aligned} I_x(x,y) &= I(x+1,\ y) - I(x-1,\ y) \\ I_y(x,y) &= I(x,\ y+1) - I(x,\ y-1) \\ I_{xx}(x,y) &= I(x+1,\ y) - 2I(x,y) + I(x-1,\ y) \\ I_{yy}(x,y) &= I(x,\ y+1) - 2I(x,y) + I(x,\ y-1) \\ I_{xy}(x,y) &= \frac{I(x+1,\ y+1) + I(x-1,\ y-1) - I(x+1,\ y-1) - I(x-1,\ y+1)}{4} \end{aligned} \tag{5.30}$$

In its simplest form, several time steps of the differential equation (5.28) are executed and the difference, $|I^{k+1} - I^k|$, is examined for any pixels that exceed a certain threshold T. These pixels are classified as belonging to a segmented edge. Since heat diffusion is a smoothing process, this form of edge detection is very robust against noise. Furthermore, the diffusion is independent of the actual image intensity level and is therefore implicitly adaptive to brightness. A simple extension is the application of region growing to the resulting edge mask: Unclassified pixels are classified as white (foreground) if they touch a white neighbor, and are classified as black (background) if they touch a black neighbor. This region-growing classification is repeated iteratively until all pixels are classified. For images with multiple regions, the classification step after the antigeometric diffusion step can be refined further[22] by applying a split-merge approach where regions of similar gray-scale statistics are merged until only a certain number of regions are left in the image or the squared error within the regions becomes too high. Subsequently, the region with the highest squared error is split again by using antigeometric diffusion on this specific region only, after which the merge step is repeated. As outcome of this process, the squared error usually converges, and the iterative application of merge and split steps can be stopped.

5.4. BIOMEDICAL EXAMPLES

Adaptive methods have been under development for many years and are finding their way into biomedical image analysis. In most cases, adaptive filters constitute an intermediate step in the image processing chain. For this reason, adaptive filtering is found most frequently as a tool rather than as the research objective of a study. With the

development of computers that have higher processing power, the algorithms available become considerably more complex. An early example of adaptive contrast enhancement is a study on the effect of local histogram equalization in CT images by Pitzer et al.[31] The focus of this study was not only the effect of local histogram equalization, but primarily a computationally effective implementation suitable for the processing power available at that time. Another early study demonstrated the superiority of adaptive sharpening over nonadaptive unsharp masking in image enhancement for the detection of lung nodules in chest x-ray images.[12] The conventional unsharp masking step made use of two masks, where the image was convolved with box filters of 35×35 and 101×101 pixels window size, respectively. The resulting unsharp masks were subtracted from the original image with weight factors of 0.5 and 1.0. This method of unsharp masking is often referred to as the *difference-of-Gaussian* (DoG) *operator* and the operation has a strong highpass character. In the adaptive variant, the same convolution was used to obtain unsharp masks, but the weight factors were allowed to vary from 0 to 0.5 and 0 to 1.0, respectively, depending on the radiolucency of the local area. In dark regions of the digitized image (i.e., the radiolucent regions of the lung), the weight factors were low, and consequently, the sharpening effect of the filter was also low. The maximum effect of the filter was reached in the projections of bone and probably in the nodule regions. In an ROC analysis, radiologists most reliably identified the nodules in the adaptively filtered x-ray images, whereas there was no difference in unfiltered and nonadaptively filtered images. The effect of the two filters (nonadaptive unsharp masking and locally adaptive unsharp masking) is demonstrated in Figure 5.14.

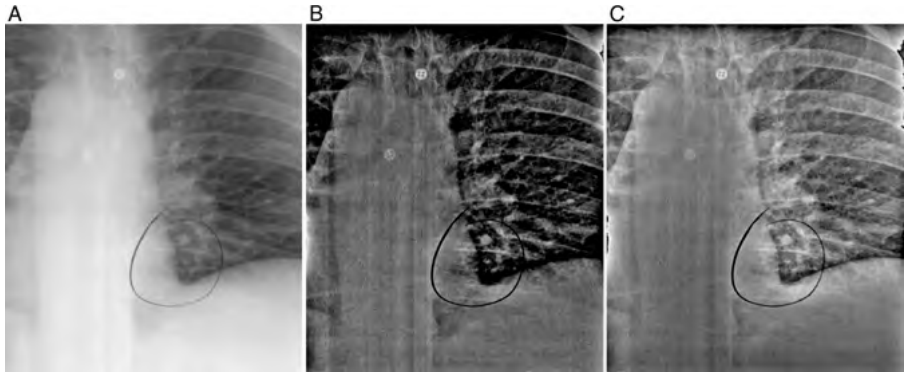


FIGURE 5.14 Effect of a locally adaptive unsharp masking filter on a scanned chest x-ray image. The original image (A) shows a high dynamic range with low contrast in critical regions, such as the lung region with the nodule (circled with a marker). Note also the two buttons in the top part of the image. Nonadaptive DoG (B) overly increases local contrast in areas of high radiolucency (i.e., the lung area), while the region of the spinal column retains low contrast. The critical areas in the lung appear overly sharp, even partially burned out. The adaptive filter that reduces its strength in areas of high radiolucency (C) balances contrast and, according to the study, allows radiologists to detect lung nodules more reliably. (From ref. 12.)

Another study where an adaptive filter was contrasted with an equivalent non-adaptive filter was based on MR data that were affected by image bias.[14] Image bias is conventionally removed by homomorphic (i.e., multiplicative) highpass filtering. In the new algorithm, a bias field model was generated iteratively by first generating a threshold map, then by creating an unsharp mask of the tissue areas in the MR image, and finally, by dividing the threshold map by the unsharp mask. The process was repeated about five times, after which features such as blood vessels, were more clearly visible than in homomorphic filtered images. Adaptive threshold masking also leads to a more uniform histogram.

X-ray mammography continues to be a challenging area for image analysis, with the goal to aid the radiologist in the detection of microcalcifications or suspicious masses. Adaptive filtering steps are frequently employed. For example, an adaptive filter that emphasizes medium-contrast areas was the basis of density-weighted contrast enhancement segmentation.[30] With additional image processing steps, namely, region growing of the objects found in the adaptive segmentation step and subsequent morphological classification, a reliable detection of suspicious masses with 97% true-positive detection and 35.5% false-positives was achieved.

A fundamentally different segmentation approach for the segmentation of microcalcifications was presented by Bankman et al.[5] The hill-climbing algorithm presented in this study is based on the fact that the edge of a microcalcification is a closed contour around an area of higher intensity. This observation implies that a microcalcification has one pixel with a local intensity maximum $I(x_0,y_0)$, which can be found algorithmically. To find the edges of the microcalcifications, pixel intensities are probed along 16 straight lines at 0°, 22.5°, 45°, ... with respect to the horizontal, starting at the local intensity maximum at x_0,y_0. For each line, a slope $s(x,y)$ is defined through

$$s(x,y) = \frac{I(x_0,y_0) - I(x,y)}{\sqrt{(x - x_0)^2 + (y - y_0)^2}} \tag{5.31}$$

and the edge is defined as the highest slope value $s(x,y)$ along the probing line (Figure 5.15). With this method, 16 edge points can be found. In addition, the edge direction is defined as the line perpendicular to the probing line. The actual hill climbing takes place in the second part of the algorithm: Starting with the 16 edge pixels as reference pixels, each pixel that is 8-connected to a reference pixel is added to the feature cluster (a) if it is an uphill pixel (i.e., its intensity is higher than the intensity of the reference pixel) and if it lies on the local maximum side of a line that goes through the reference pixel and is perpendicular to the line connecting the reference pixel to the local maximum, or (b) if its intensity is lower than the intensity of the reference pixel and the pixel is closer to the local maximum by the distance of one or more pixels, that is,

$$\sqrt{(x_0 - x_r)^2 + (y_0 - y_r)^2} \geq \sqrt{(x_0 - x)^2 + (y_0 - y)^2} + 1 \tag{5.32}$$

Pixels that are added to the cluster serve iteratively as reference pixels, and the iteration stops when no more pixels are added to the cluster. The result is a

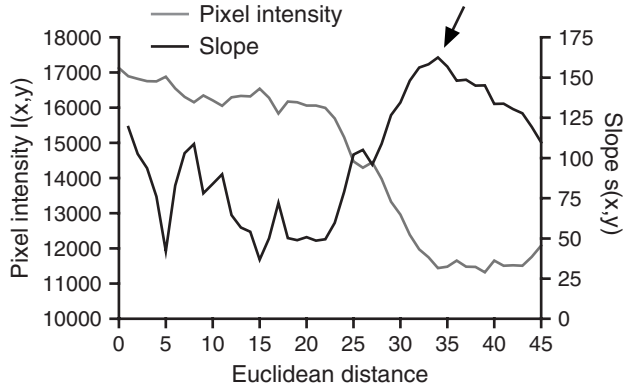


FIGURE 5.15 Intensity and slope along one representative probing line. The probing line starts at the local maximum (Euclidean distance 0), and pixel intensity drops off somewhat monotonically with increasing distance. The slope shows a distinct maximum at the steepest drop-off, approximately 35 pixels from the local maximum (arrow). This point on the probing line is marked as an edge point.

pixel mask that defines the area of microcalcification. Thus, the hill-climbing algorithm segments the microcalcification, but due to the definition of the slope in Equation (5.31), it does so in an adaptive manner: Hill climbing is independent of the local intensity, and it is also independent of local statistics. The hill-climbing algorithm has no external parameters except the maximum length of the probing ray and is therefore excellently suited for fully unsupervised segmentation.

An interesting filter in the context of the detection of suspicious breast masses is the iris filter,[19] which is based on the same idea of enhancing image areas to which the gradient field converges: that is, uphill, when the image is interpreted as an elevation landscape. To implement the iris filter, a number of probing lines of equal length R start radially at an arbitrary pixel at x_0,y_0. For each pixel along these lines, the convergence angle θ is determined as the angle between the gradient direction and the direction of the line. If the gradient exists (i.e., is nonzero), $\cos\theta$ therefore indicates the degree of convergence, because $\cos\theta$ is close to unity when the gradient points toward x_0,y_0 and close to zero when the gradient is perpendicular to the probing line. The degree of convergence for a given point i on probing line k is defined as

$$C_{i,k}(x_0,y_0) = \frac{\sum_{Q=Q_0}^{Q_i} \cos\theta[G(Q_i) > 0]}{\overline{Q_0 Q_i}} \tag{5.33}$$

where the expression $[G(Q_i) > 0]$ assumes the value 1 if a gradient G at pixel Q_i exists (i.e., is nonzero) and 0 otherwise. For each probing line, the maximum convergence $C_k(x_0,y_0)$ is determined as the maximum of all $C_{i,k}$ of a probing line k. The final output value of the iris filter at x_0,y_0 is the average of all $C_k(x_0,y_0)$. A variant of the

iris filter was proposed by Varela et al.[39] where the convergence determination on each probing ray starts at a minimum distance R_{min} from the central point x_0,y_0 rather than at the center point itself, which reduces noise influence and therefore improves the contrast of suspicious masses. The iris filter was followed by segmentation with a histogram-dependent adaptive threshold and subsequent feature classification. Final evaluation was done using a neural network. The iris filter is a powerful enhancement function for convex shapes with increasing intensity toward the center. However, the iris filter requires an additional segmentation step, as opposed to the hill-climbing algorithm, which implicitly segments the area.

Ultrasound imaging is another modality where adaptive filters have been shown to be a powerful tool in image enhancement. Ultrasound images are generally affected by a random texture of relatively large scale, which has been described as intensity- and location-dependent: The noise level increases with the overall signal intensity, and lateral (but not axial) speckle size increases with depth.[26] One example where a locally adaptive minimum mean-square filter as introduced in Section 5.1 was used to reduce variations within adjoining regions (blood region and myocardial region in the heart) was presented by Nillesen et al.[25] Noise reduction was performed with an adaptive filter similar to the filter described in Equation (5.3) where the filter strength was controlled locally by the degree of homogeneity, defined as σ^2/μ (i.e., the intensity variance divided by the mean intensity). Application of this filter strongly reduced the echo level overlap between blood and myocardium, thus improving segmentation results.

A different adaptive filter, adaptive lightening and darkening, was found to be a core step in the segmentation of tissue affected by periventricular leukomalacia in the neonatal brain.[35] Once again, ultrasound noise speckles obscured the difference between normal and diseased tissue. Adaptive lightening is defined as increasing the intensity value of a pixel by 1 when the intensity of both pixels of a diagonally opposite pixel pair are higher than the central pixel. In this context, there are four pixel pairs (horizontal, vertical, and two diagonal neighbor pairs), but the horizontal pixel pair is not considered, probably because of the anisotropic nature of the speckles. Conversely, adaptive darkening is defined as decreasing the intensity value by 1 when both pixel intensities of the same pixel pairs are lower than the intensity of the central pixel. Since diseased tissue is brighter (more echogenic) than healthy tissue, brightening is applied to image areas above a certain threshold, and darkening is applied to areas below a different and lower threshold. In the transition zone in between the thresholds, weaker darkening is applied. The result of this filter is a form of texture smoothing, but in a directed manner. Within-region contrast is reduced, while at the same time the overall intensity of diseased tissue is raised and the overall intensity of healthy tissue is lowered. Final segmentation took place by means of active contours.

With the goal of providing the radiologist with a subjectively adjustable speckle suppression filter, Thakur and Anand[36] proposed a filter termed the *gray-level co-occurrence matrix-based adaptive speckle filter*. The idea is based on region growing, where region growth and region smoothing are based on second-order statistical parameters: the local mean intensity μ and the local intensity variance σ^2. The noise variance is assumed to scale with the intensity (i.e., $\sigma^2 \propto \mu$). This assumption allows

use of the ratio σ^2/μ as a measure of homogeneity. The co-occurrence matrix provides several texture descriptors, such as the local contrast and the inverse difference moment[9] (see Section 8.2). The adaptive speckle filter is based on these texture parameters, where, starting from a square window, the region is shrunk and areas of high contrast and low inverse difference moment (i.e., probably containing discontinuities) are excluded from the subsequent region-growing step. In the next step, regions are grown along homogeneous areas with low contrast and high inverse difference moment, and these regions are smoothed using mean-value computation. Depending on various filter settings, the ultrasound image can be freed almost completely of speckle noise while the contrast of critical regions is preserved.

Various other methods of adaptive filtering exist in different contexts. For example, an improved thresholding algorithm for three-dimentional micro-CT images of trabecular bone has been presented.[8] In this approach, the core of the segmentation was performed using a three-dimentional Sobel operator on a smoothed version of the original image volume. A hysteresis threshold provided an edge map. A local threshold map was formed to include or exclude specific edge voxels to provide a smooth edge contour. Examples for the application of the anisotropic diffusion filter (Section 5.1) were given in a review article.[43] In this study, the superiority of the anisotropic diffusion filter as a processing step in the segmentation of the brain cortex surface in MR images was shown. Finally, active contours ("snakes"; see Chapter 6) are frequently employed to parametrize the contour of a feature. Even the snake algorithm can be made adaptive by employing a local modified trimmed mean noise-reduction step in the vicinity of a snake vertex. In addition, for the specific application of snakes in ultrasound images, where edges have a relatively low slope, the gradient force usually employed is replaced by a ramp integration force to steer the snake toward the edge.[10]

Adaptive filters have become part of the fundamental image processing toolbox. A selection of adaptive filtering, image enhancement, and segmentation algorithms was presented in this chapter. As computers become more powerful, more complex adaptive image processing algorithms are likely to be developed. In the future, adaptive filtering will probably be combined more and more often with other techniques. Examples include frequency-domain adaptive filters,[21] the combination of wavelet decomposition with adaptive thresholding for ultrasound speckle reduction,[16] or adaptive wavelet filtering techniques for speckle reduction in optical coherence tomography images.[2] Finally, the combination of adaptive filters with artificial intelligence methods is a subject of active research.[29] There are three major artificial intelligence techniques that are suitable to improving adaptive filtering: fuzzy logic, neural networks, and evolutionary computation. With fuzzy logic, a more human-vision-based approach to characterizing image areas as dark or bright, as having an edge or being smooth, or to characterize properties of different textures is possible. One recent example is the combination of the adaptive median filter with fuzzy logic to reduce heavy shot noise in MR images.[15] Neural networks are frequently employed when a closed-form description of classifiers is not possible. A training image set is used to train the network, and the network is then to some extent capable of applying the learned criteria to new images. With evolutionary

computation, optimization problems (such as image enhancement) can be performed using genetic algorithms. One recent example is a genetic algorithm attempt at ultrasound speckle reduction.[24] Genetic algorithms are search algorithms with a variable parameter space and a fitness function. The parameter space (the "genetic representation") can be randomly mutated and its effect on the fitness function observed. The simulation of evolutionary processes can be highly detailed with mutation, selection, breeding, crossover inheritance, and other phenomena.

REFERENCES

1. Abramatic JF, Silverman LM. Nonlinear restoration of noisy images. IEEE Trans Pattern Anal Mach Intell 1982; 4(2):141–149.
2. Adler DC, Ko TH, Fujimoto JG. Speckle reduction in optical coherence tomography images by use of a spatially adaptive wavelet filter. Opt Lett 2004; 29(24):2878–2880.
3. Backus GE, Gilbert JF. Uniqueness in the inversion of inaccurate gross Earth data. Philos Trans R Soc A 1970; 266:123–192.
4. Ball J, Butler T, Bruce L. Towards automated segmentation and classification of masses in mammograms. Conf Proc IEEE Eng Med Biol Soc 2004; 3:1814–1817.
5. Bankman IN, Nizialek T, Simon I, Gatewood OB, Weinberg IN, Brody WR. Segmentation algorithms for detecting microcalcifications in mammograms. IEEE Trans Inf Technol Biomed 1997; 1(2):141–149.
6. Bernsen J. Dynamic thresholding of grey-level images. Proceedings of the Eighth International Conference on Pattern Recognition (ICPR), Paris, 1986; 1251–1255.
7. Bieniecki W, Grabowski S. Multi-pass approach to adaptive thresholding based image segmentation. Proceedings of the 8th International IEEE Conference CADSM, Lvov-Polyana, Ukraine, 2005; 418–423.
8. Burghardt AJ, Kazakia GJ, Majumdar S. A local adaptive threshold strategy for high resolution peripheral quantitative computed tomography of trabecular bone. Ann Biomed Eng 2007; 35(10):1678–1686.
9. Chanda B, Majumder DD. Note on the use of the graylevel co-occurrence matrix in threshold selection. Signal Process 1988; 15(2).
10. Chen CM, Lu HH. An adaptive snake model for ultrasound image segmentation: modified trimmed mean filter, ramp integration and adaptive weighting parameters. Ultrason Imaging 2000; 22(4):214–236.
11. Chen T, Ma K-K, Chen L-H. Tri-state median filter for image denoising. IEEE Trans Image Process 1999; 8(12):1834–1838.
12. Correa J, Souto M, Tahoces PG, Malagari KS, Tucker DM, Larkin JJ, Kuhlman J, Barnes GT, Zerhouni EA, Fraser RG. Digital chest radiography: comparison of unprocessed and processed images in the detection of solitary pulmonary nodules. Radiology 1995; 195(1):253–258.
13. Dhawan AP, Le Royer E. Mammographic feature enhancement by computerized image processing. Comput Methods Programs Biomed 1988; 27(1):23–35.
14. Gaudanek MA, Hess A, Obermayer K, Sibila M. Adaptive threshold masking. In: Tolxdorff T, Braun J, Deserno TM, Handels H, Horsch A, Meinzer H-P, editors. Bildverarbeitung

für die Medizin 2008 [Medical Image Processing 2008]. CEUR Workshop Proceedings. Berlin: Springer-Verlag, 2008; 347:373–376.

15. Güler I, Toprak A, Demirhan A, Karakis R. MR images restoration with the use of fuzzy filter having adaptive membership parameters. J Med Syst 2008; 32(3):229–234.
16. Gupta S, Chauhan RC, Saxena SC. Homomorphic wavelet thresholding technique for denoising medical ultrasound images. J Med Eng Technol 2005; 29(5):208–214.
17. Guy MJ. Fourier block noise reduction: an adaptive filter for reducing Poisson noise in scintigraphic images. Nucl Med Commun 2008; 29(3):291–297.
18. Ko S-J, Lee YH. Center weighted median filters and their applications to image enhancement. IEEE Trans Circuits Syst 1991; 38(9):984–993.
19. Kobatake H, Murakami M, Takeo H, Nawano S. Computerized detection of malignant tumors on digital mammograms. IEEE Trans Med Imaging 1999; 18(5):369–378.
20. Lee JS. Digital image enhancement and noise filtering by use of local statistics. IEEE Trans Pattern Anal Mach Intell 1980; 2(2):165–168.
21. Li W, Meunier J, Soucy JP. A 3D adaptive Wiener filter for restoration of SPECT images using MRI as reference images. Conf Proc IEEE Eng Med Biol Soc 2005; 3:3100–3103.
22. Manay S, Yezzi A. Anti-geometric diffusion for adaptive thresholding and fast segmentation. IEEE Trans Image Process 2003; 12(11):1310–1323.
23. Metzler V, Bienert H, Lehmann T, Mottaghy K, Spitzer K. A novel method for quantifying shape deformation applied to biocompatibility testing. ASAIO J 1999; 45(4): 264–271.
24. Munteanu C, Morales FC, Ruiz-Alzola J. Speckle reduction through interactive evolution of a general order statistics filter for clinical ultrasound imaging. IEEE Trans Biomed Eng 2008; 55(1):365–369.
25. Nillesen MM, Lopata RG, Gerrits IH, Kapusta L, Huisman HJ, Thijssen JM, de Korte CL. Segmentation of the heart muscle in 3-D pediatric echocardiographic images. Ultrasound Med Biol 2007; 33(9):1453–1462.
26. Oosterveld BJ, Thijssen JM, Verhoef WA. Texture of B-mode echograms: 3-D simulations and experiments of the effects of diffraction and scatterer density. Ultrason Imaging 1985; 7(2):142–160.
27. Otsu N. Threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 1979; 9(1):62–66.
28. Perona P, Malik J. Scale-space and edge detection using anisotropic diffusion. IEEE Trans Pattern Anal Mach Intell 1990; 12(7):629–639.
29. Perry SW, Wong H-S, Guan L. Adaptive Image Processing: A Computational Intelligence Perspective. Boca Raton, FL: CRC Press, 2002.
30. Petrick N, Chan HP, Sahiner B, Helvie MA. Combined adaptive enhancement and region-growing segmentation of breast masses on digitized mammograms. Med Phys 1999; 26(8):1642–1654.
31. Pitzer SN, Zimmerman JB, Staab EV. Adaptive grey level assignment in CT scan display. J Comput Assist Tomogr 1984; 8(2):300–305.
32. Rangayyan RM, Ciuc M, Faghih F. Adaptive-neighborhood filtering of images corrupted by signal-dependent noise. Appl Opt 1998; 37(20):4477–4487.
33. Ridler TW, Calvard S. Picture thresholding using an iterative selection method. IEEE Trans Syst Man Cybern 1978; 8:630–632.

34. Shen L, Rangayyan RM, Desautels JEL. Detection and classification of mammographic calcifications. Int J Pattern Recogn Artif Intell 1993; 7(6):1403–1416.
35. Stippel G, Philips W, Govaert P. A tissue-specific adaptive texture filter for medical ultrasound images. Ultrasound Med Biol 2005; 31(9):1211–1223.
36. Thakur A, Anand RS. Speckle reduction in ultrasound medical images using adaptive filter based on second order statistics. J Med Eng Technol 2007; 31(4):263–279.
37. Tomasi C, Manduchi R. Bilateral filtering for gray and color images. Sixth Int Conf Comput Vis 1998; 839–846.
38. Umbaugh SE. Computer Imaging: Digital Image Analysis and Processing. Boca Raton, FL: Taylor & Francis, 2005.
39. Varela C, Tahoces PG, Mendez AJ, Souto M, Vidal JJ. Computerized detection of breast masses in digitized mammograms. Comput Biol Med 2007; 37(2):214–226.
40. Westin C, Kikinis R, Knutsson H. Adaptive image filtering. In: Bankman I, editor. Handbook of Medical Imaging: Processing and Analysis. San Diego, CA: Academic Press, 2000.
41. Xu X, Miller EL, Chen D, Sarhadi M. Adaptive two-pass rank order filter to remove impulse noise in highly corrupted images. IEEE Trans Image Process 2004; 13(2):238–247.
42. Zhang B, Allebach JP. Adaptive bilateral filter for sharpness enhancement and noise removal. IEEE Trans Image Process 2008; 17(5):664–678.
43. Zingale R, Zingale A. Detection of MRI brain contour using isotropic and anisotropic diffusion filter: a comparative study. J Neurosurg Sci 1998; 42(2):111–114.

6

DEFORMABLE MODELS AND ACTIVE CONTOURS

In simple terms, two-dimensional deformable models (two-dimensional closed-contour deformable models are also called *snakes*) can be thought of as the numerical modeling of rubber bands subjected to image-dependent external forces. Deformable models are used in image processing to delineate and segment features interactively. The most interesting property of deformable models is their closed shape, even if image features are disturbed by noise. The outline of an artery in an ultrasound image, for example, may be composed of a series of disjoint ellipse segments after gradient computation and thresholding. An active contour, locking onto the gradient, will form a closed contour, thus allowing easy quantification, such as diameter and area computation. Another key feature is the ability of a deformable contour to adapt when the underlying image changes. In the analysis of time-varying image sequences (e.g., the beating heart or blood vessels dilating with blood pressure), active contours can be used for rapid determination of time-dependent parameters.

Active contours exist in two and three dimensions. A two-dimensional active contour can be compared with a rubber band that tends to contract on its own but locks onto image features as if slipped over an irregular, prismatic object. A three-dimensional active contour is an elastic balloon that encloses a surface and gets drawn to the surface contour. Imagine a thick rubber band such as an O-ring. Normally, it assumes a circular shape of a certain diameter. If stretched or bent (compressed) and then released again, it returns to its original shape. Two forces in equilibrium, the stretching and bending force, control this behavior. Let us first imagine that the rubber band is composed of a large number of infinitesimal springs that are

Advanced Biomedical Image Analysis, By Mark A. Haidekker

connected in sequence at nodes. Each node can move freely. The nodes are at rest in a force equilibrium between the spring forces of the springs at each side. If the spring constants are the same over the length of the rubber band, the nodes try to position themselves at an equal distance. Furthermore, the application of an external force to stretch the rubber band introduces energy. Let us denote the path s of the rubber band in a two-dimensional case as $s(p) = (s_x(p), s_y(p))$ such that $p = 0$ marks the start and $p = 1$ marks the end of the path, irrespective of its length. In this case, the first derivative of the path, $\partial s/\partial p$, is proportional to the length and, through Hooke's law, to the stretching force of each spring element. We can formulate an expression akin to a potential energy term that describes the energy needed to stretch the rubber band:

$$\mathscr{E}_{\text{stretch}} = \int_0^1 \alpha(p) \left| \frac{\partial s}{\partial p} \right|^2 dp \tag{6.1}$$

The constant α, which may be variable over the length p of the rubber band, is analogous to Hooke's constant in springs. Integration takes place over all infinitesimal spring elements, that is, in our definition from $p = 0$ to $p = 1$. The idealized rubber band, under the influence of stretching forces only and without external forces, would contract to a point.

In a similar manner, the springs can be bent, and a force needs to be applied to bend the springs, which would otherwise have a natural tendency to relax into a straight shape. Furthermore, the stiffness against bending causes the rubber band not to have kinks and sharp edges. The bending force is proportional to the second derivative of the path, $\partial^2 s/\partial p^2$. Similar to Equation (6.1), we can formulate a *bending energy*:

$$\mathscr{E}_{\text{bend}} = \int_0^1 \beta(p) \left| \frac{\partial^2 s}{\partial p^2} \right|^2 dp \tag{6.2}$$

Here the constant β represents the stiffness against bending. The idealized rubber band, under the influence of bending forces only and without external forces, would relax to a straight line. The stretching and bending energies are called *internal energies*,

$$\mathscr{E}_{\text{int}} = \mathscr{E}_{\text{stretch}} + \mathscr{E}_{\text{bend}} \tag{6.3}$$

In its natural state, the rubber band would try to minimize the internal energies. If the rubber band is an O-ring, that is, $s(0) = s(1)$, the two energies come in conflict: To minimize the stretching energy, the O-ring wants to contract. In this process the radius becomes smaller and its curvature increases. Therefore, the bending energy increases. Depending on α and β, there exists a radius where the sum of the two energies becomes minimal, and this is the equilibrium radius of the O-ring in the absence of external forces. The implicit regularity constraints force the O-ring into a circular shape.

If an external force is applied (e.g., somebody takes the O-ring and pinches it), its shape would get distorted, to some extent yielding to the externally applied force. As the rubber band or snake moves to assume its final shape, the energy sum $\mathscr{E}$ of internal and external forces is minimized:

$$\mathscr{E} = \mathscr{E}_{\text{ext}} + \mathscr{E}_{\text{int}} = \mathscr{E}_{\text{ext}} + \int_0^1 \alpha(p) \left| \frac{\partial s}{\partial p} \right|^2 + \beta(p) \left| \frac{\partial^2 s}{\partial p^2} \right|^2 dp \rightarrow \min \tag{6.4}$$

External image-dependent forces need to be applied to steer the snake toward the image feature of interest. If we envision the image as an elevation landscape where the image value corresponds to height, the rubber band has a location-specific potential energy. Under its own weight and the influence of gravity, the rubber band would try to slide down the image slopes until held back by stretching forces. In the extreme case, we can envision the rubber band gliding down a canyon and snapping tight along the canyon bottom.

How can we create such a canyon? Image features are usually characterized by their edges, and the most straightforward approach is to use the image gradient, with a negative sign, as a steering force. Total snake energy is lowered when the snake descends along the negative gradient. Therefore, the external energy component could be formulated as

$$\mathscr{E}_{\text{ext}} = -G(x,y) \circledast \nabla I(x,y) \tag{6.5}$$

where ∇ is the gradient operator and $G(x,y)$ is a Gaussian kernel. The convolution $\circledast$ of the gradient image with a Gaussian smoothing function is important not only to reduce noise, but also to blur and therefore broaden the edge gradients. This increases the capture range of the snake (Figure 6.1). Other formulations for the external energy and additional external energy terms exist and are discussed later in this chapter.

To explain further how a snake gets attracted to an image feature, let us consider synthetic image where a square feature exists embedded in a noisy background (Figure 6.2A). As sketched in Figure 6.1, the steep edges are spread by a Gaussian blurring step (Figure 6.2B) before an edge detector (Sobel operator) provides a potential energy field that is maximal at the feature edges (Figure 6.2C). To be

FIGURE 6.1 Sketch of an image edge (A) causing an external energy. The edge is shown in cross section (A). After Gaussian blurring (B) the edge is spread out. Its negative gradient (C) forms a canyon that attracts the snake by gravity. If the snake is near enough to the canyon walls, a strong external force exists (gray arrow) that pulls the snake toward the canyon bottom to lower its potential energy. Outside the canyon, however, the snake experiences no external forces (thin dashed arrow).

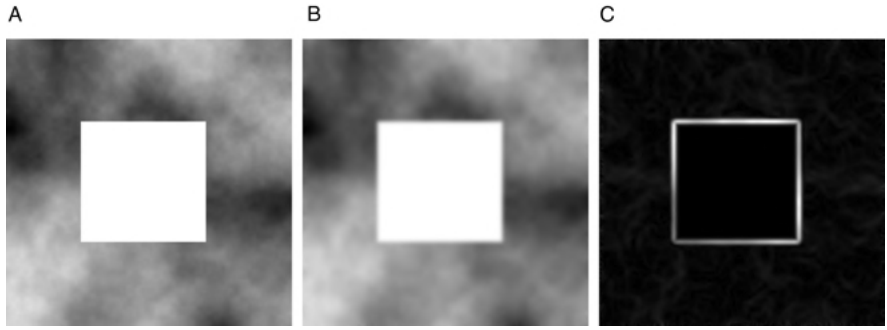


FIGURE 6.2 Synthetic test image to demonstrate how a snake is attracted to a feature. The feature of interest is a white square in a noisy background (A). Blurring is necessary to increase the width of significant edges (B). Edge enhancement (Sobel operator) provides the image gradient with high values, representing the feature edges (C).

consistent with the notion of minimizing the snake's total energy, the gradient needs to be inverted. The feature edges become a potential energy minimum, a deep canyon into which the snake can slip to minimize its potential energy (Figure 6.3).

Whereas the minimization of the internal energy forces the snake to assume a smooth shape, inclusion of the external energy term forces the snake to lock onto image features even if they are not smooth (Figure 6.4). The balance of internal and external forces, controlled by α and β, determines the final shape and therefore the extent to which the snake follows an irregular shape or to which it takes a shortcut, even if the shortcut does not represent the gradient maximum. A medical example for this behavior in a closed deformable contour model (snake) is shown in Figure 6.5.

For a practical computer implementation, two additional steps are necessary. First, a solution needs to be found for the problem in Equation (6.4), and second, the continuous equations, including Equations (6.4) and (6.5), need to be discretized. A balance-of-forces approach to solving the minimization problem in Equation (6.4)

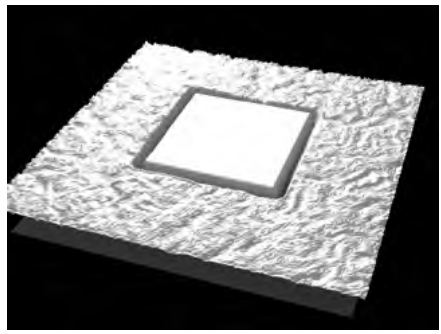


FIGURE 6.3 Three-dimensional surface rendering of the negative gradient in Figure 6.2C. This image visualizes the potential energy field of the image. The steep edges of the feature create an energy valley into which the snake can slip, thus minimizing its total energy.

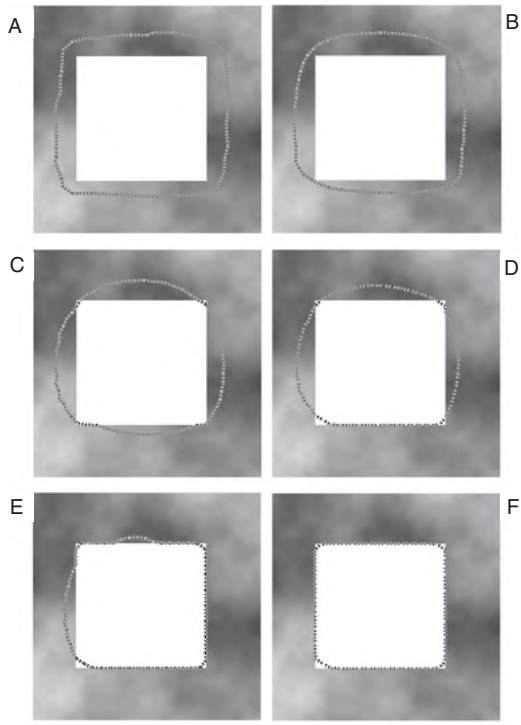


FIGURE 6.4 Convergence of a snake around a feature (Figure 6.2). Starting with a user-selected initial placement (A), the snake starts to contract while most vertices are too far away from the edge to experience its energy (B). While the snake continues to contract in subsequent iterations (C, D), some vertices experience the energy-minimum canyon depicted in Figure 6.3 and lock onto the edge. In image E, most vertices are captured in the energy-minimum canyon, and after a few more iterations, the snake reaches its final shape (F). The internal bending forces are responsible for the snake to "cut corners."

leads to the following equation,[7] which models the time-dependent motion of the snake contour $s(p)$:

$$\frac{\partial s}{\partial t} - \alpha \frac{\partial^2 s}{\partial p^2} + \beta \frac{\partial^4 s}{\partial p^4} + \frac{\partial \mathscr{E}_{\text{ext}}}{\partial p} = 0 \tag{6.6}$$

When a force equilibrium is reached, the snake motion stops and the term $\partial s/\partial t$ vanishes. Equation (6.6) then simplifies to the solution given by Kass et al.[16]:

$$-\alpha \frac{\partial^2 s}{\partial p^2} + \beta \frac{\partial^4 s}{\partial p^4} + \frac{\partial \mathscr{E}_{\text{ext}}}{\partial p} = 0 \tag{6.7}$$

In the discrete form, the snake is a polygon with N vertices s_i, and the path p becomes the vertex index i with $0 \leq i < N$. It is sometimes helpful to add another vertex s_N

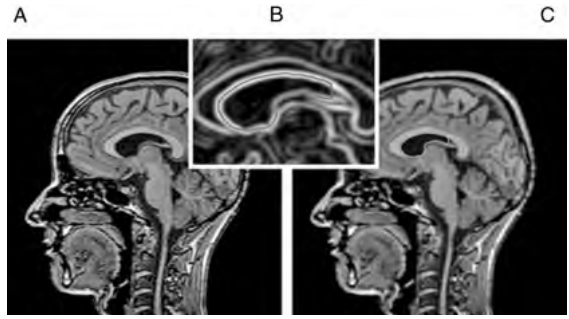


FIGURE 6.5 Application of a deformable contour for the segmentation of the ventricular space in an MR image slice of the brain (A). The snake tends primarily to follow the external energies, which are represented by the magnitude in the gradient image (inset B). Note that the snake uses a weak gradient ridge to take a "shortcut" on the right-hand side, where the bending energy would get very high. This behavior can be controlled with the parameter β. Image C shows the slice after the application of a smoothing filter with the snake superimposed. (*See insert for color representation of the figure*.)

with the constraint $s_0 = s_N$. Equation (6.7) can now be discretized by using finite differences, which leads to a discrete balance-of-forces formulation:

$$\begin{aligned}&\alpha_i(s_i - s_{i-1}) - \alpha_{i+1}(s_{i+1} - s_i) + \beta_{i-1}(s_{i-2} - 2s_{i-1} + s_i)\\ &\quad - 2\beta_i(s_{i-1} - 2s_i + s_{i+1}) + \beta_{i+1}(s_i - 2s_{i+1} + s_{i+2}) + F_i = 0\end{aligned} \tag{6.8}$$

Here F_i is the external force term at each node i:

$$F_i = \nabla \mathscr{E}_{\text{ext}} \tag{6.9}$$

which can also be discretized in each dimension through central differences. Equation (6.8) is a general representation of n-dimensional active contours for nonconstant terms α and β. In the two-dimensional case, s_i becomes (x_i,y_i) and F_i becomes

$$F_i = \begin{pmatrix} \dfrac{E(x_{i+1}) - E(x_{i-1})}{2\Delta x} \\ \dfrac{E(y_{i+1}) - E(y_{i-1})}{2\Delta y} \end{pmatrix} \tag{6.10}$$

According to Kass et al.,[16] the set of equations (6.8) for all N vertex points of the snake leads to a matrix representation that can be solved in discrete time steps. A very efficient solution to the discretized problem in (6.8) is the "greedy snake" algorithm, which is covered in the next section.

Alternatively, a one time-discrete step [Equation (6.6)] can be taken by using a first-order Euler approximation:

$$s^{k+1} = s^k + \Delta t \left(\alpha \frac{\partial^2 s}{\partial p^2} - \beta \frac{\partial^4 s}{\partial p^4} - \frac{\partial \mathscr{E}_{\text{ext}}}{\partial p} \right) \tag{6.11}$$

Modeling the snake through the time-dependent balance-of-forces approach allows us to include mass and friction terms, thus giving the snake a more intuitive physical meaning, particularly when using the snake to segment time-variable shapes. By adding mass and friction terms, Equation (6.7) generalizes to

$$m\frac{\partial^2 s}{\partial t^2} + D\frac{\partial s}{\partial t} - \alpha\frac{\partial^2 s}{\partial p^2} + \beta\frac{\partial^4 s}{\partial p^4} + \frac{\partial \mathscr{E}_{\text{ext}}}{\partial p} = 0 \tag{6.12}$$

with m representing the mass of the snake and D its friction or damping. Moreover, the balance-of-forces approach allows us to consider additional external forces. Examples include ballooning forces,[7] which allow the user to inflate or deflate the snake, and springs, repulsors, and attractors, which make it possible to impose local forces on the snake. These elements are normally controlled by user interaction and help steer the snake toward the desired edge. This feature is particularly helpful when multiple local energy minima exist, onto which the snake would arbitrarily lock without user interaction. The presence of multiple minima can be seen clearly in Figure 6.5B, where the outer boundary of the *corpus callosum* provides a less intense, yet still dominant edge for the snake to snap onto.

While several related approaches to deformable models exist—for example, geometric deformable models that use closed-term functions to describe curves[6,19] or a curve description that uses finite elements[9]—this chapter focuses on the parametric models introduced above because of their relatively simple implementation and widespread use in biomedical imaging.

There are two main uses for snakes. First, the snake contour can be used to segment an object. Second, the snake vertices serve as parametrization of the feature segmented. One important representation is the Fourier parametrization of the snake.[27] The snake contour $s(p)$ along a path parameter p can be described by a set of orthonormal coefficients following

$$s(p) = \begin{bmatrix} x(p) \\ y(p) \end{bmatrix} = \begin{bmatrix} a_0 \\ c_0 \end{bmatrix} + \sum_{k=1}^{\infty} \begin{bmatrix} a_k & b_k \\ c_k & d_k \end{bmatrix} \begin{bmatrix} \cos 2\pi kp \\ \sin 2\pi kp \end{bmatrix} \tag{6.13}$$

The Fourier coefficients can be computed from snake vertices x_i and y_i through

$$\begin{aligned} a_0 &= \frac{1}{N}\sum_{i=0}^{N-1} x_i \\ a_k &= \frac{1}{N}\sum_{i=0}^{N-1} x_i \cos 2\pi ik \qquad k > 0 \\ b_k &= \frac{1}{N}\sum_{i=0}^{N-1} x_i \sin 2\pi ik \end{aligned} \tag{6.14}$$

with $b_0 = 0$ and the coefficients c_k and d_k computed with the same equations but using y_i instead of x_i. The coefficients a_0 and c_0 are the x- and y-coordinates of the

snake centroid, and the coefficients a_1 through d_1 provide information on the size and orientation of the shape. Higher-order coefficients provide information on shape irregularities at different scales. The Fourier coefficients facilitate shape identification with shape templates and comparison of shapes by means of the cross-correlation function. The use of Fourier descriptors to describe a shape is covered in more detail in Section 9.5.

6.1. TWO-DIMENSIONAL ACTIVE CONTOURS (SNAKES)

6.1.1. Greedy Snake Algorithm

Probably the most widely used implementation of a snake is the greedy snake algorithm introduced by Williams and Shah.[34] Some investigators noted stability problems with the original numerical implementation by Kass et al.[16] and presented alternative approaches to improve numerical stability and to reduce the tendency of snake vertices to cluster around strong edges.[1,7,34] Williams and Shah devised a particularly attractive implementation, which is not based on the time-dependent representation of the snake forces [Equation (6.6)] but, rather, uses the energy formulation [Equation (6.4)] directly.

In the numerical model, a snake is represented by $N + 1$ vertices (sometimes called *snaxels* for snake pixels) $v_k = (x_k, y_k)$, $0 \le k \le N$. Furthermore, the additional condition $v_0 = v_N$ turns the snake into a closed contour. The internal energies [Equations (6.1) and (6.2)] can be discretized by finite differences. One possible representation of the stretching energy is

$$E_{\text{stretch}} = \sum_{i=1}^{N} |v_i - v_{i-1}|^2 = \sum_{i=1}^{N} \left[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2\right] \tag{6.15}$$

However, this representation tends strongly to shrink the curve and creates the tendency for the vertices to cluster together. A better approach is the computation of a continuity energy that assumes its minimum when the vertices are most evenly spaced, as described by

$$E_{\text{contin}} = \sum_{i=1}^{N} \left(\bar{D} - |v_i - v_{i-1}|\right)^2 = \sum_{i=1}^{N} \left[\bar{D} - \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right]^2 \tag{6.16}$$

$$\bar{D} = \frac{1}{N} \sum_{i=1}^{N} |v_i - v_{i-1}| = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{6.17}$$

Equation (6.17) describes the average distance between adjoining vertices, and Equation (6.16) computes the total continuity energy as the total squared deviation from the average distance. Therefore, the continuity energy creates a tendency for the

snake to space its vertices evenly. This is of importance, because the next energy term, the curvature energy, relies on evenly spaced vertices. The curvature energy can be described through

$$E_{\text{curv}} = \sum_{i=0}^{N-1} \left[\frac{x_i - x_{i-1}}{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}} - \frac{x_{i+1} - x_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \right]^2 + \left[\frac{y_i - y_{i-1}}{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}} - \frac{y_{i+1} - y_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \right]^2 \quad (6.18)$$

In this formulation, the curvature is the second derivative of the curve implemented with finite differences that are normalized by the distance between corresponding vertices. Other formulations exist, such as the nonnormalized version of Equation (6.18),

$$\begin{aligned} E_{\text{curv}} &= \sum_{i=0}^{N-1} |v_{i-1} - 2v_i + v_{i+1}|^2 \\ &= \sum_{i=0}^{N-1} \left[(x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2 \right] \end{aligned} \quad (6.19)$$

but they may have numerical disadvantages such as not being rotationally invariant.

The image energy is represented by the smoothed gradient [Equation (6.5)], where the gradient image $I_G(x,y)$ is precomputed as demonstrated in Figure 6.2 by applying a gradient operator (such as the Sobel operator) preceded or followed by Gaussian smoothing. The corresponding discrete energy term is therefore

$$E_{\text{ext}} = -\sum_{i=0}^{N-1} I_G(x_i, y_i) \quad (6.20)$$

and the total snake energy may be obtained by adding the continuity, curvature, and external energy terms in Equations (6.17), (6.18), and (6.20). The crucial element of the greedy snake algorithm is the iterative search for energy minima in the neighborhood of each vertex. For this purpose, the energy terms are not added yet. For each vertex i in succession, the energy components in Equations (6.17), (6.18), and (6.20) are computed separately for each pixel j of a $m \times m$ neighborhood of the vertex. Each individual energy matrix is then normalized to fall into the range 0 to 1:

$$E'_j = \frac{E_j - E_{\min}}{E_{\max} - E_{\min}} \quad (6.21)$$

where $E_{\max}$ and $E_{\min}$ are the highest and lowest values of the corresponding energy in the $m \times m$ neighborhood matrix. A weighted sum of all normalized energies is computed,

$$E_0 = \min_{j=1}^{m^2} \left(\alpha_i E'_{\text{cont},j} + \beta_i E'_{\text{curv},j} + \gamma_i E'_{\text{ext},j} \right) \tag{6.22}$$

and the neighborhood position with the minimum energy E_0 is determined. The vertex is moved immediately to the pixel associated with the lowest energy. The next vertex is examined and displaced in the same fashion until all vertices have been examined and displaced once in each iteration. The exception is vertex v_0, which is examined and displaced twice. The reason for processing v_0 twice is the asymmetric behavior of the algorithm with respect to the curvature where vertex v_{i-1} has already been displaced, whereas v_{i+1} has not. The algorithm stops after a preset number of iterations or after the snake converges (i.e., no displacement takes place over one full iteration). The search process for one vertex is illustrated in Figure 6.6. The neighborhood in this example is $m = 3$, and all energy terms in the neighborhood are computed individually before the weighted sum is computed through Equation (6.22). In the example of Figure 6.6, the energy minimum E_0 is assumed to be found below and to the left of the pixel, leading to a displacement of $(-1,1)$.

The weight parameters α, β, and γ can be made adaptive. In this case, each vertex is assigned one individual value α_i, β_i, and γ_i. Initially, all weight values are initialized to the same value. Williams and Shah propose using $\alpha_i = \beta_i = 1.0$ for all i, and $\gamma_i = 1.2$ for all I, and adapting β during the iterative process. In each iteration, β is set to 0 for any vertex i that meets two conditions[34]: (1) the curvature exceeds a predetermined threshold value (i.e., the curve strongly bends at this vertex), and (2) the gradient magnitude must exceed a predetermined gradient threshold (i.e., a strong edge exists at the position of this vertex).

A disadvantage of the Williams–Shah algorithm is the local normalization of the image forces [Equation (6.21)]. As in local histogram equalization, this step tends to amplify small local gradients and noise. For this reason, a normalization threshold

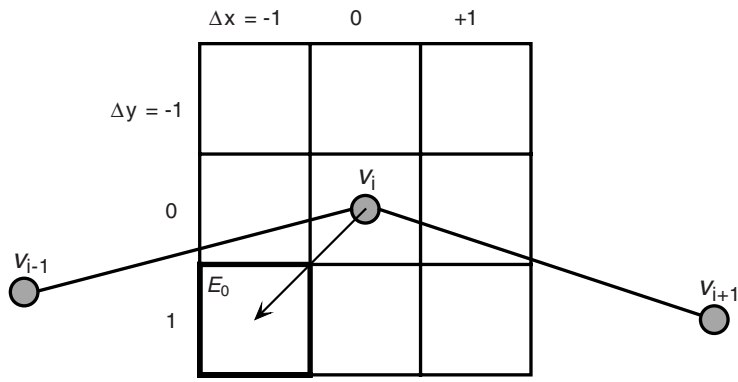


FIGURE 6.6 Search neighborhood of size 3×3 for the minimum energy for vertex v_i. In this example, the minimum sum of energies is found at the pixel below and to the left of the current vertex position, and the vertex is therefore displaced by $(-1,1)$.

is proposed[34] whereby the denominator in Equation (6.21) cannot drop below approximately 2% of the gradient image's dynamic range. Despite this threshold, local gradients have an unproportionally high influence on the snake.

Variations of the greedy snake algorithm exist. For example, the tendency to contract a snake that uses Equation (6.15) for the first-order internal energy can be useful. The initial shape of such a snake would be user-selected to be larger than the image feature of interest, and the snake would shrink until it locks onto the edges of the feature. The continuity term is still necessary to maintain equidistant vertices. The greedy algorithm by Williams and Shah has the tendency to move the entire snake by the same displacement as the first vertex, v_0, in homogeneous regions. Also, the curvature energy term creates an additional tendency of the snake to shrink, since the curvature energy is lowest when the vertex is displaced toward the center of the snake. Finally, the greedy algorithm by Williams and Shah reacts in an unpredictable manner if parameters α, β, and γ are adjusted interactively during the iteration. A more robust approach is described in Algorithm 6.1. For this algorithm, it is assumed that a function to calculate the total snake energy exists:

$$E_{\text{snake}} = \sum_{i=0}^{N-1} [\alpha E_{\text{stretch}}(v_i) + \beta E_{\text{curv}}(v_i) + \gamma E_{\text{ext}}(v_i) + \delta E_{\text{cont}}(v_i)] \tag{6.23}$$

The function would require the gradient image and the snake vertex coordinates as parameters to be able to compute the energies. In Algorithm 6.1, this function would be supplied as `compute_energy(sx,sy,GRIM)`, where `sx` and `sy` are the arrays containing the vertex coordinates and `GRIM` is the gradient image, computed through Equation (6.5). The algorithm further differs from the Williams and Shah algorithm, because it computes all displacements beforehand on the undisplaced snake. Only after all displacements are known are they applied to the snake.

Further modifications and improvements are possible. The energy formulation allows for easy implementation of additional constraint energies. Ballooning forces could be represented in Equation (6.15) with a variable weight factor that can assume positive and negative values. If the weight factor is positive, the energy diminishes as the snake shrinks in overall length, thus inducing a tendency to deflate the snake contour. Conversely, if the weight factor is negative, the energy increases as the snake shrinks in overall length and therefore exerts an inflating force. In Algorithm 6.1, this feature can be implemented simply by allowing α to assume negative values as well as positive. In addition, repulsor and attractor points can be placed interactively to help steer the snake. If Q attractors and R repulsors exist with the coordinates $x_{a,k}$, $y_{a,k}$ and $x_{r,k}$, $y_{r,k}$, respectively, the constraint energy computed in the following equation can be added to the total energy term [Equation (6.23)]:

$$\begin{aligned} E_{\text{const}} = {} & \epsilon_1 \sum_{k=0}^{Q-1} \left\{ \sum_{i=0}^{N-1} \left[(x_{a,k} - x_i)^2 + (y_{a,k} - y_i)^2 \right]^n \right\} \\ & + \epsilon_2 \sum_{k=0}^{R-1} \left\{ \sum_{i=0}^{N-1} \frac{1}{\left[(x_{r,k} - x_i)^2 + (y_{r,k} - y_i)^2 \right]^n} \right\} \end{aligned} \tag{6.24}$$

```
set ngh=1;                                          // Choose a suitable neighborhood size
set m = SQR(2*ngh+1);                               // number of pixels in square neighborhood
set alpha=1, beta=1, gamma=1.2, delta=1;            // weight parameters
set N = 50;                                         // number of snake vertices
allocate sx[N];                                     // Allocate N vertices with x...
allocate sy[N];                                     // ... and y
allocate dx[N];                                     // We also need storage for the displacements with x...
allocate dy[N];                                     // ...and y
allocate e_field[m];                                // energies in vertex neighborhood
allocate GRIM(xmax,ymax);                           // the GRadient IMage

// Prepare the gradient image by convolution

GRIM = convolve (Gauss(xmax,ymax), sobel(im,xmax,ymax));

// Iterations: Repeat snake algorithm while at least one vertex is displaced
itercnt=0;
do
   for (i=0 while i<N increment i=i+1) do     // Examine all vertices by test-displacing them
      x0 = sx[i]; y0 = sy[i];                 // Save present vertex position
      k=0;
      for (y=-ngh while y<=ngh increment y=y+1) do
         for (x=-ngh while x<=ngh increment x=x+1) do
            sx[i]=x0+x; sy[i]=y0+y;           // test-displacement
            e_field[k] = compute_energy (sx,sy,GRIM);
            k = k+1;
         endfor;
      endfor;

      E0 = e_field[0]; k=0;
      for (y=-ngh while y<=ngh increment y=y+1) do          // Search the
         for (x=-ngh while x<=ngh increment x=x+1) do       // neighborhood
            if (E0 > e_field[k]) then
               E0 = e_field[k];
               deltax=x; deltay=y;                  // displacement
            endif
            k = k+1;
         endfor;
      endfor;
      dx[i]=deltax; dy[i]=deltay;      // Store displacement until all vertices examined
   endfor;                             // done examining all vertices

   movement=0;
   for (i=0 while i<N increment i=i+1) do       // Now displace the vertices
      if ( (dx[i]!=0) and (dy[i]!=0)) then
         sx[i] = sx[i]+dx[i];
         sy[i] = sy[i]+dy[I];
         movement = movement+1;
      endif;
   endfor;
   itercnt = itercnt+1;
while ((itercnt<100) and (movement>0));

delete sx; delete sy;
delete dx; delete dy;
delete e_field;
delete GRIM;
```

Algorithm 6.1 Modified greedy snake algorithm. In this algorithm, the neighborhoods for all vertices are examined before any vertex is displaced. Only after all displacements are known are the displacements applied and the iterations are repeated until convergence is achieved or until a preset number of iterations is exceeded. At the end, the algorithm contains the new shape of the snake in the arrays `sx` and `sy`.

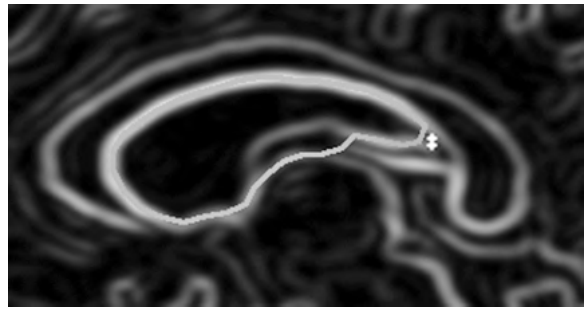


FIGURE 6.7 The snake is pulled into the high-bending zone by two attractors (green dots). (*See insert for color representation of the figure*.)

where the first summation term represents the forces of the Q attractors on each of the N vertex points (with weight factor ϵ_1), and the second summation term represents the forces of the R repulsors on each of the N vertex points (with weight factor ϵ_2). The exponent n may be used to choose a decay of the attractor and repulsor forces with $1/r$ or $1/r^2$ decay, respectively. Care should be taken with the repulsor terms when a repulsor is in close proximity to a vertex. In this case, the repulsive force may grow exceedingly high. For this reason, the summation terms of the repulsors are often capped: for example, not to exceed unity. The resulting profile gives it a volcanolike top, which is the reason that repulsors are often termed *volcanoes*.[16] It is possible to provide individual strength terms for each attractor and repulsor. However, each user-selectable parameter makes the algorithm more complex to control with increased user interaction and therefore creates a more subjective interpretation of the image. The application of attractors is demonstrated in Figure 6.7, where the snake in Figure 6.5B is helped to reach the narrow area of high bending energy with two closely spaced attractors, placed interactively by the user.

Increasing the search neighborhood of each vertex allows the greedy snake to "see" edges from a larger distance at the expense of computational efficiency. One possibility of increasing computational efficiency is to use rectangular neighborhoods with the dominant length perpendicular to the snake path. In addition, a larger amount of blurring broadens the edges and therefore increases the capture range. However, more blurring also removes details. A solution for this dilemma is multilevel blurring. The snake starts its search on a severely blurred image and switches to a less blurred gradient image upon convergence. Multilevel blurring may be a powerful strategy with modest additional computational effort, because the levels of blurring can be computed iteratively before applying the snake algorithm.

6.1.2. Balance-of-Forces Approach

The formulation of the balance-of-forces equation of a snake [Equation (6.6)] will be referred to as the *force implementation*, as opposed to the greedy or energy implementation based on Equation (6.4). The central governing equation for one

iteration is Equation (6.11). The discrete velocity of the snake contour (related to the steepness of the negative force gradient) can be described by

$$v(t) = \alpha \frac{\partial^2 s}{\partial p^2} - \beta \frac{\partial^4 s}{\partial p^4} - \frac{\partial \mathscr{E}_{\text{ext}}}{\partial p} \tag{6.25}$$

where $v(t)$ has two components in the two-dimensional case. The internal forces can be approximated through central differences [Equation (6.26) for the x-direction and an analogous equation for the y-direction]:

$$\begin{aligned} \left.\frac{\partial^2 x}{\partial p^2}\right|_i &\approx x_{i-1} - 2x_i + x_{i+1} \\ \left.\frac{\partial^4 x}{\partial p^4}\right|_i &\approx x_{i-2} - 4x_{i-1} + 6x_i - 4x_{i+1} + x_{i+2} \end{aligned} \tag{6.26}$$

The external force is represented simply by the central difference of the gradient image. The weighted sum of external and internal forces, multiplied by a small Δt, constitutes the displacement of the vertex. The displacement may be a floating-point number, and a displacement generally does not move a vertex from one grid point to another, making interpolation necessary. Yet this algorithm can be made very efficient. The factors in Equation (6.26), combined with either constant or adaptive α and β, are

$$\begin{cases} \beta_{i-1} \\ -(\alpha_i + 2\beta_{i-1} + 2\beta_i) \\ \alpha_i + \alpha_{i+1} + \beta_{i-1} + 4\beta_i + \beta_{i+1} \\ -(\alpha_{i+1} + 2\beta_i + 2\beta_{i+1}) \\ \beta_{i+1} \end{cases} \tag{6.27}$$

It can be seen that the factors can be arranged for all vertices in matrix form, yielding a pentadiagonal matrix $\mathbf{A}$, which for nonadaptive α and β is

$$\mathbf{A} = \begin{bmatrix} c & d & e & 0 & 0 & \cdots & 0 & 0 & 0 & a & b \\ b & c & d & e & 0 & \cdots & 0 & 0 & 0 & 0 & a \\ a & b & c & d & e & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & d & \cdots & 0 & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & a & b & c & d & e \\ e & 0 & 0 & 0 & 0 & \cdots & 0 & a & b & c & d \\ d & e & 0 & 0 & 0 & \cdots & 0 & 0 & a & b & c \end{bmatrix} \tag{6.28}$$

with the coefficients derived from Equation (6.27) as $a = e = \beta$, $b = d = -\alpha - 4\beta$, and $c = 2\alpha + 6\beta$. This matrix needs to be combined with the parameter γ, added to each element of the main diagonal (effectively using $c = 2\alpha + 6\beta + \gamma$), and inverted

to yield $\mathbf{A}_\gamma^{-1}$. Let X and Y be the vectors of the snake's x and y coordinates and F_x and F_y the external force field. Then the snake position can be iterated from step k to step $k+1$ through

$$\begin{aligned} X^{k+1} &= A_\gamma^{-1}(\gamma X^k + \tau F_x) \\ Y^{k+1} &= A_\gamma^{-1}(\gamma Y^k + \tau F_y) \end{aligned} \tag{6.29}$$

Note that the computation of the matrix $\mathbf{A}_\gamma^{-1}$ can be done beforehand if α, β, and γ are constant, so that the matrix inversion does not enter the iterative loop. The new parameter τ may be used to fine-tune the balance of image forces, although generally it is possible to choose $\tau = 1$. The two directional gradients of the image energy may be precomputed as well. Therefore, each iteration takes little computational effort.

On the other hand, the force formulation tends to be numerically unstable. The second-order term is responsible for the shrinking of the snake. The fourth-order term that represents the bending rigidity is extremely noise-sensitive and tends to cause time-variable ripples in the snake. In practice, the performance of the snake does not deteriorate if $\beta = 0$. Another critical issue is the choice of the time step τ. Small τ cause more stable behavior but slower convergence. Large τ may cause a vertex to overshoot the edge and either start oscillations or a vertex getting attracted by a neighboring edge. The choice of a suitable set of parameters is much more critical in the force formulation than in the greedy snake formulation. This critical behavior of the force formulation is possibly one reason for the popularity of the greedy snake algorithm. Nonetheless, the force formulation is necessary for snakes that simulate physical behavior, that is, snakes with defined mass and friction, and snakes that are subjected to spring forces as additional constraint forces. However, careful implementation of the numerical methods becomes very important.

6.1.3. Gradient Vector Flow

In the context of the greedy snake algorithm, multilevel blurring was mentioned as one method to increase the capture range of the snake. Another powerful method to increase the capture range and to provide an external force field that stabilizes the behavior of the snake is to diffuse the edge image. Xu and Prince[35] propose a gradient vector flow field, computed from the edge image through

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mu \nabla^2 u - \left(u - \frac{df}{dx}\right)(\nabla f)^2 \\ \frac{\partial v}{\partial t} &= \mu \nabla^2 v - \left(v - \frac{df}{dy}\right)(\nabla f)^2 \end{aligned} \tag{6.30}$$

where u and v are the forces in the x and y directions, respectively, f is the edge image, obtained, for example, by applying the Sobel operator, and μ is a normalization factor that determines the amount of smoothing. Equation (6.30) can be discretized by using the convolution-based discrete Laplacian operator (Section 2.3) for $\nabla^2 u$

and $\nabla^2 v$, first-order central differences to obtain df/dx and df/dy, and the squared result of the Sobel operator applied on f to obtain $(\nabla f)^2$. By using a first-order Euler approximation, u and v can be computed iteratively with a time step Δt:

$$\begin{aligned} u^{k+1} &= u^k + \Delta t \left[\mu \nabla^2 u^k - \left(u^k - \frac{df}{dx} \right) (\nabla f)^2 \right] \\ v^{k+1} &= v^k + \Delta t \left[\mu \nabla^2 v^k - \left(v^k - \frac{df}{dy} \right) (\nabla f)^2 \right] \end{aligned} \tag{6.31}$$

The gradient fields $(\nabla f)^2$, df/dx, and df/dy can be computed outside the iteration loop. The gradient vector flow field can be computed before the application of the snake algorithm and used directly for the forces f_x and f_y in Equation (6.29). Assume that two convolution-based operators, acting on an image $I(x,y)$, `Laplace(I)` and `Sobel(I)`, are available. A variant, `NSobel(I)`, computes the edge image but ensures that all values are normalized to lie in the range 0 to 1. Furthermore, the directional differences `diff_x (I)` and `diff_y (I)` are defined as pointwise central differences

$$\mathrm{diff}_x(x,y) = \frac{I(x+1,y) - I(x-1,y)}{2} \tag{6.32}$$

$$\mathrm{diff}_y(x,y) = \frac{I(x,y+1) - I(x,y-1)}{2} \tag{6.33}$$

With these prerequisites, the gradient vector flow can be computed using Algorithm 6.2.

Algorithm 6.2 shows slow convergence, and a large number of iterations need to be performed to provide a reasonable spread of the edges. More efficient implementations are possible, primarily by using multigrid techniques. On a 256×256 image, Algorithm 6.2 takes on the order of tens of seconds to converge. Even with an efficient implementation, several seconds of computation time are required. Numerical stability depends critically on the selection of a small enough Δt, which in turn depends on μ. If the grid space is assumed to be unity, Δt should be smaller than $1/4\mu$.[35] In practical applications, even smaller values were necessary to prevent diverging behavior, and the condition $\Delta t < 0.2513/(\mu + 0.1774)$ was determined experimentally. Therefore, Δt should be selected to be a safe margin less than $0.2513/(\mu + 0.1774)$ in the range $0.1 \leq \mu \leq 0.4$. Convergence can easily be monitored by displaying ϵ over the progression of the time-stepping loop. Furthermore, it is critically important that the edge image be normalized to the value range of 0 to 1; otherwise, the algorithm will not converge.

Figures 6.8 and 6.9 demonstrate the superior capture range of the gradient vector field compared with a blurred edge image. It can be seen in Figure 6.8B that almost the entire interior of the feature has nonzero forces, whereas the interior of a conventional blurred edge image [Equation (6.5)] does not exhibit any forces in most of the interior space (Figure 6.9B). This behavior has two consequences. In a regular gradient image, the snake is placed outside the feature and shrinks until it locks onto the edge. When

```
set mu=0.2;                          // Regularization parameter; higher mu, more smoothing.
set deltat=0.5;                      // Time step size; large values cause instable behavior
set iter=500;                        // number of time-stepping iterations
allocate GRIM(xmax,ymax);            // the GRadient of the primary IMage
allocate FX(xmax,ymax);              // the image of directional derivatives in x
allocate FY(xmax,ymax);              // the image of directional derivatives in y
allocate FSQR(xmax,ymax);            // the image of nondirectional squared derivatives
allocate TMPX(xmax,ymax);            // temporary storage for the U increment
allocate TMPY(xmax,ymax);            // temporary storage for the V increment
allocate U(xmax,ymax);               // output: force field in x-direction
allocate V(xmax,ymax);               // output: force field in y-direction

// Prepare the gradients of the gradient image
GRIM = NSobel(IM,xmax,ymax);              // edge image, normalized to range 0...1
FSQR = SQR(Sobel(GRIM,xmax,ymax));        // pixel-by-pixel squared
FX = diff_x (GRIM,xmax,ymax);             // directional derivative in x
FY = diff_y (GRIM,xmax,ymax);             // directional derivative in y
U = FX; V = FY;                           // initialization of U and V

// Iterations: Repeat the diffusion time step sufficiently often to spread the edges

for (i=0 while i<iter increment i++)

   TMPX = Laplace(U);
   TMPY = Laplace(V);
   for (y=0 while y<ymax increment y=y+1)
      for (x=0 while x<xmax increment x=x+1)
         TMPX(x,y) = mu*TMPX(x,y) - (U(x,y) - FX(x,y))*FSQR(x,y);
         TMPY(x,y) = mu*TMPY(x,y) - (V(x,y) - FY(x,y))*FSQR(x,y);
      endfor;
   endfor;
   epsilon = MAX(TMPX,TMPY);     // largest value of the increments
   for (y=0 while y<ymax increment y=y+1)
      for (x=0 while x<xmax increment x=x+1)
         U(x,y) = U(x,y) +deltat*TMPX(x,y) ;
         V(x,y) = V(x,y) +deltat*TMPX(x,y) ;
      endfor;
   endfor;

endfor;                              // Done iterating - U and V are ready

delete (GRIM, FX, FY, FSQR, TMPX, TMPY);
```

Algorithm 6.2 Gradient vector flow. This algorithm computes the gradient vector flow field of an input image. The input image is `IM(x,y)` and is subjected to the Sobel operator immediately to obtain the edge image. From the edge image, the directional and nondirectional gradients `FX`, `FY`, and `FSQR` are computed. A temporary image per direction is needed to hold the Laplacian images (`TMPX` and `TMPY`). The output of the algorithm are the forces `U` and `V`. Instead of stepping through the specified number of iterations, the variable epsilon may be used as a criterion of convergence and iteration stopped once epsilon drops below a predetermined threshold.

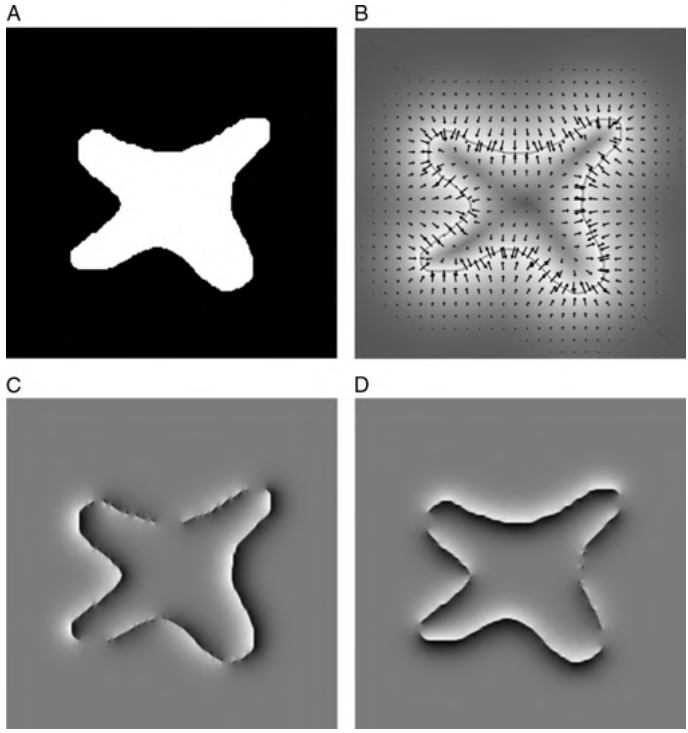


FIGURE 6.8 Gradient vector flow used as the external image force for snakes. In part A, a simple image feature with concave regions is presented. Part B shows the magnitude of the gradient vector flow field with the forces superimposed as small arrows. Parts C and D are gray-scale representations of the forces acting in the x and y directions, respectively (U and V in Algorithm 6.2), where medium gray represents no force, dark shades represent forces in the negative direction (left and up, respectively), and light shades represent forces in the positive direction (right and down, respectively).

exposed to the gradient vector flow forces, the initial snake size is widely irrelevant[35] because forces exist in most of the image area that pull the snake toward the edge. Even a small snake placed inside the shape will be exposed to forces that expand it until it reaches the edge. Furthermore, a snake exposed to gradient vector flow forces is much more likely to follow a concave shape than a snake exposed to a simple gradient, because a snake that tries to stretch across the concave section with gradient vector flow is exposed to forces that pull it inside the concave region.

Although gradient vector flow is most commonly used in conjunction with the force formulation, it is possible to implement gradient vector flow forces in a greedy algorithm as well. In this case, the force vectors can be used to fill an $n \times n$ neighborhood with energy values that represent the external energy in the greedy search algorithm (Figure 6.10). The gradient vector flow imposes a force $\mathbf{F}$ on vertex v_i of the snake. Each of the eight neighborhood pixels is connected to the vertex through a pixel vector $\mathbf{p}$.

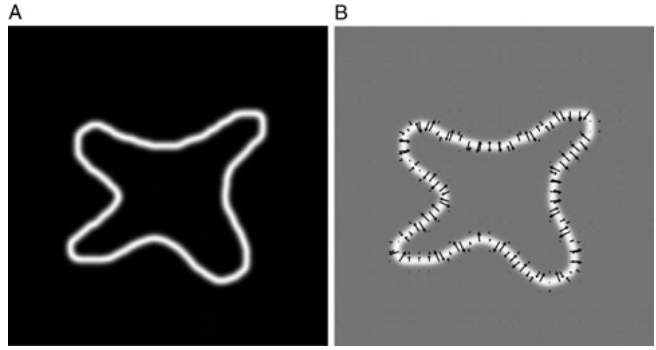


FIGURE 6.9 Conventional computation of the gradient field for the snake algorithm using Equation (6.5). (A) shows the edge image of Figure 6.8A, blurred twice by a Gaussian kernel with $\sigma = 1.6$ and a 11×11 neighborhood. In (B), the gradient forces are shown by magnitude (gray shade) and in vector form (arrows). The capture range (i.e., the area surrounding the edge in which the forces are nonzero) is considerably smaller than the gradient vector flow forces in Figure 6.8B.

The corresponding energy at the pixel with displacement Δx and Δy can therefore be computed through

$$E_k(\Delta x, \Delta y) = -|\mathbf{F}|\,|\mathbf{p}_k| \cos(\Phi - \varphi_k) \tag{6.34}$$

where the index k covers all eight neighbor pixels, and the displacement $(\Delta x, \Delta y)$ that minimizes the energy of the vertex in its neighborhood is chosen.

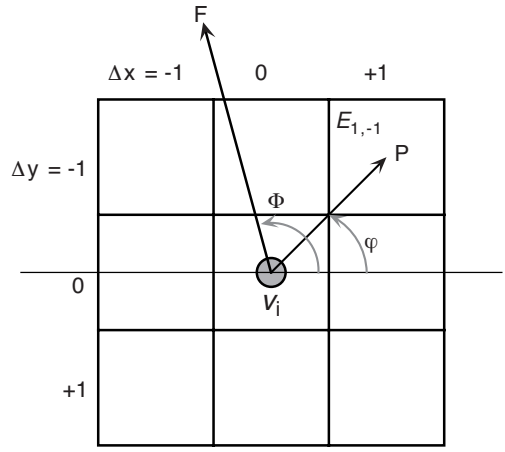


FIGURE 6.10 The search neighborhood of a greedy snake vertex can be filled with energy values based on gradient vector flow by multiplying the force vector **F** by the potential pixel displacement vector **p**.

6.1.4. Topologically Adaptive Snakes

An extension of the snake model was presented by McInerney and Terzopoulos.[21] Up to this point, snakes were assumed to be the topological equivalent of a circle. This restrictive assumption can be overcome if a snake is allowed to split or if two intersecting snakes are allowed to merge. McInerney and Terzopoulos propose projecting the snake onto a simplicial cell decomposition of the image surface. In two-dimensional space, this decomposition can be realized by subdividing the plane into equal squares, then subdividing each square into two triangles (Freudenthal triangularization), as shown in Figure 6.11. In the presence of a single snake, each triangle will therefore be either inside or outside the snake, or it will intersect the snake. Triangles that intersect the snake can be thought of as boundary cells, and the boundary cells approximate the snake contour. By using cells as a shape approximation, a snake can either be found to be self-intersecting or two snakes can be found to share a series of consecutive cells. In these cases, the single snake is split or the two snakes are merged, respectively. This process implies that the snake implementation needs the ability to handle multiple snakes and to allocate and delete snakes dynamically.

Figure 6.12 demonstrates the evolution of a topology-adaptive snake. The object is a CT cross section of a spine phantom, of which the edge image is shown in Figure 6.11. Initially, two snakes that have been placed near the vertebral processes start to expand, seeking the edges (part A). The two snakes intersect below the spinal canal and merge. The resulting single snake self-intersects and splits off a small snake that encloses the spinal canal (part B). The major snake expands along the vertebral cortex until it self-intersects near the apex of the vertebra and splits off a third snake that encloses the vertebral spongiosa (part C).

FIGURE 6.11 Edge image of a CT slice of a spine phantom with a grid of triangular cells superimposed.

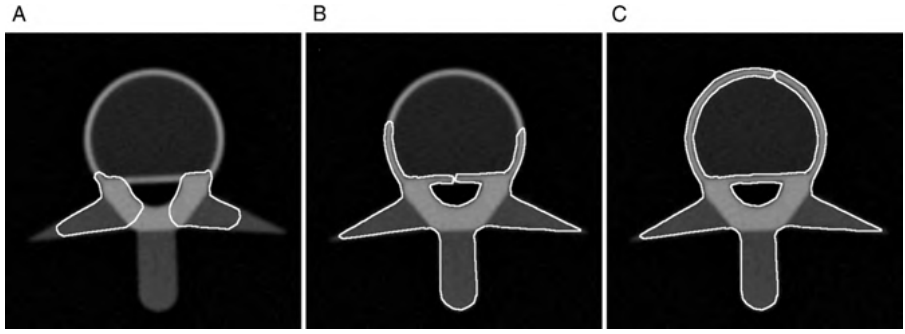


FIGURE 6.12 Evolution of topology-adaptive snakes along the edges of a CT image of a spine phantom. (A) Two snakes start expanding and seeking the phantom edges. The snakes will intersect below the spinal channel and merge. (B) Above the spinal channel, the snake self-intersects and splits off a snake enclosing the spinal canal. (C) The dominant snake self-intersects again near the apex of the vertebra and splits off a third snake that encloses the spongy bone area, resulting in a total of three snakes.

6.2. THREE-DIMENSIONAL ACTIVE CONTOURS

The snake, a one-dimensional curve that encloses a two-dimensional image feature, was discussed in the previous sections. The governing equations can easily be extended to three dimensions. For example, the balance of forces described in Equation (6.7) would be discretized through finite differences in the x, y, and z directions. The external force (image force) would be represented by a gradient in three directions, broadened by three-dimensional Gaussian blur or three-dimensional gradient vector flow. The vertices (which now are allowed to move in three spatial directions toward their energy minimum) constitute a closed surface that surrounds a three-dimensional image feature, much like a balloon that contracts and shrink-wraps the object. An example of this process is shown in Figure 6.13, where a balloonlike active surface is dragged toward the edge of the object and shrink-wraps around it, closely following the brain gyri and sulci in the last iteration.

Specialized formulations, such as greedy formulation, ballooning forces, and gradient vector flow, can also readily be adapted for three dimensions. However, the added dimension drastically increases computation time. In addition, interactive initial placement of the shape becomes more difficult. One alternative is the placement of a series of snakes, slice by slice, around the object. The individual snakes can then be joined to form a closed surface and be subjected to the iterative minimization of the snake energy.

A detailed formulation of a three-dimensional active surface, termed a *three-dimensional active net*, was provided by Takanashi et al.[30] A mesh, initially rectangular, is discretized into vertices $v(p,q) = (x(p,q), y(p,q), z(p,q))$. Here p and q are parameters of the surface with $0 \leq p \leq 1$ and $0 \leq q \leq 1$. Index variables i and j can be

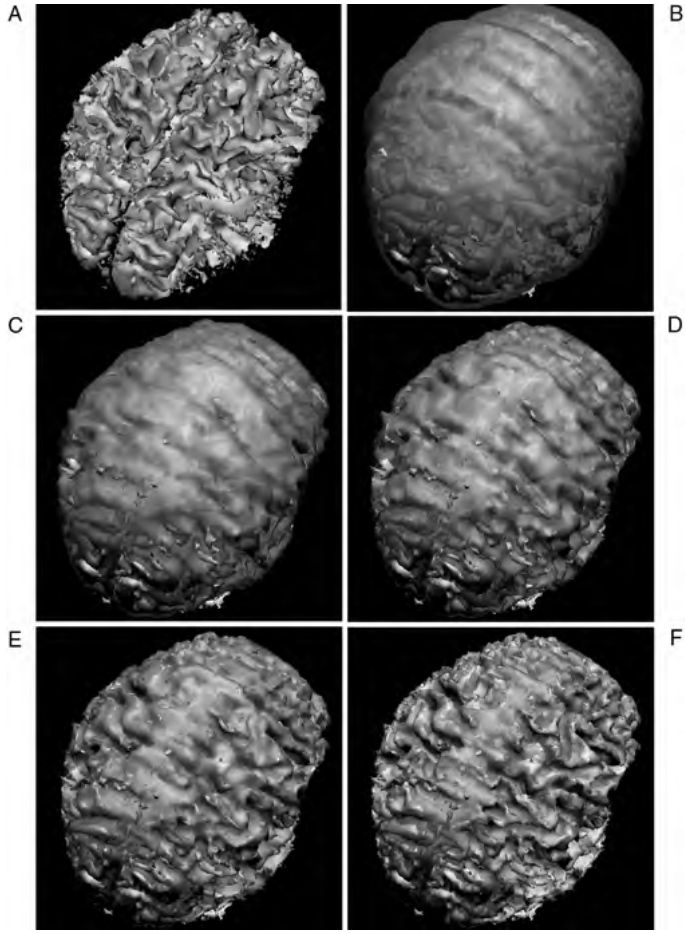


FIGURE 6.13 Surface rendering of a human brain from MR data (A) after segmentation of the brain and deepening of the sulci by thresholding. A balloon-line active surface is shown step by step to shrink around the segmented brain and follow the gradient forces into the sulci (B–F). Particularly in image F, the active surface can be seen to take some "shortcuts" to minimize its bending energy. (*See insert for color representation of the figure.*)

used to enumerate the vertices. For a mesh with $(n+1)\,m$ vertices, we can therefore write

$$\begin{aligned} v(p,q) = v(ik,jl) &= (x_{ij}, y_{ij}, z_{ij}) \qquad 0 \le i \le n, 0 \le j \le m-1 \\ k = \frac{1}{n} &\qquad l = \frac{1}{m-1} \end{aligned} \tag{6.35}$$

Each vertex with the exception of the boundary vertices is connected with four neighboring vertices. To enclose a convex, three-dimensional shape, the mesh is now

deformed into an ellipsoid in two steps. First, a tube surface is created when matching vertices along the left and right borders are connected with the boundary condition $v(n, jl) = v(0, jl)$. Additional mesh connections are created between diagonally opposing vertices by duplicating the mesh outside the range $0 \leq i \leq n$ and with the additional boundary condition

$$\begin{aligned} v(r, -s) &= v(\bar{r}, s) \\ v(r, m + s) &= v(\bar{r}, m - s) \\ \bar{r} &= \begin{cases} r + nk/2 & \text{if } (r + nk/2) \leq m \\ r - nk/2 & \text{if } (r + nk/2) > m \end{cases} \end{aligned} \tag{6.36}$$

where n is required to be an even number. In the second step, the cylinder is distorted to form an ellipsoid, and the vertices of the upper and lower end rings of the tube collapse into a single vertex. The resulting mesh is a three-dimensional network rather than a balloon. Energy minimization takes place analogously to the snake through

$$E_{\text{net}} = \sum_{i=0}^{n} \sum_{j=0}^{m-1} E_{\text{int}}\left[v(i,j)\right] + E_{\text{image}}\left[v(i,j)\right] + E_{\text{const}}\left[v(i,j)\right] \tag{6.37}$$

E_{int} is the sum of the stretching energy (computed through the first derivative toward the two index variables) and the bending energy (computed through the second derivative toward the two index variables). E_{const} is the energy contribution from constraints such as attractors and repulsors. Since this energy depends strongly on the user interface, $E_{\text{const}} = 0$ can be assumed in most cases. The image energy, E_{image}, is the absolute image intensity in this formulation as opposed to the gradient magnitude in the snake formulations. In the numerical implementation, the minimization of E_{net} gives rise to three independent Euler–Lagrange differential equations that cover the partial derivatives of all three spatial components:

$$\begin{aligned} -\alpha\left(\frac{\partial^2 x}{\partial p^2} + \frac{\partial^2 x}{\partial q^2}\right) + \beta\left(\frac{\partial^4 x}{\partial p^4} + \frac{\partial^4 x}{\partial p^2 \partial q^2} + \frac{\partial^4 x}{\partial q^4}\right) + \gamma\frac{\partial E_{\text{image}}}{\partial x} &= 0 \\ -\alpha\left(\frac{\partial^2 y}{\partial p^2} + \frac{\partial^2 y}{\partial q^2}\right) + \beta\left(\frac{\partial^4 y}{\partial p^4} + \frac{\partial^4 y}{\partial p^2 \partial q^2} + \frac{\partial^4 y}{\partial q^4}\right) + \gamma\frac{\partial E_{\text{image}}}{\partial y} &= 0 \\ -\alpha\left(\frac{\partial^2 z}{\partial p^2} + \frac{\partial^2 z}{\partial q^2}\right) + \beta\left(\frac{\partial^4 z}{\partial p^4} + \frac{\partial^4 z}{\partial p^2 \partial q^2} + \frac{\partial^4 z}{\partial q^4}\right) + \gamma\frac{\partial E_{\text{image}}}{\partial z} &= 0 \end{aligned} \tag{6.38}$$

α, β, and γ are user-selectable weight factors with a function analogous to the snake [Equation (6.23)]. Equation (6.38) can readily be discretized with finite

differences. For the x component of Equation (6.38), the following finite differences can be applied:

$$\begin{aligned}
\frac{\partial^2 x}{\partial p^2} &= \frac{x_{i-1,j} - 2x_{i,j} + x_{i+1,j}}{k^2} \\
\frac{\partial^2 x}{\partial q^2} &= \frac{x_{i-1,j} - 2x_{i,j} + x_{i+1,j}}{l^2} \\
\frac{\partial^2 x}{\partial p^4} &= \frac{x_{i-2,j} - 4x_{i-1,j} + 6x_{ij} - 4x_{i+1,j} + x_{i+2,j}}{k^4} \\
\frac{\partial^4 x}{\partial p^2 \partial q^2} &= \frac{x_{i-1,j-1} - 2x_{i,j-1} + x_{i+1,j-1} - 2x_{i-1,j} + 4x_{i,j} - 2x_{i+1,j} + x_{i-1,j+1} - 2x_{i,j+1} + x_{i+1,j+1}}{k^2 l^2} \\
\frac{\partial^4 x}{\partial q^4} &= \frac{x_{i,j-2} - 4x_{i,j-1} + 6x_{i,j} - 4x_{i,j+1} + x_{i,j+2}}{l^4}
\end{aligned} \tag{6.39}$$

Equation (6.39) lists one spatial component only; the partial derivatives of the y and z components can be obtained in an analogous manner. For the gradient of the image energies, the finite difference scheme

$$\frac{\partial E_{\text{image}}}{\partial x} = |I(x+1, y, z) - I(x-1, y, z)| \tag{6.40}$$

or a Sobel-like difference operator may be used. Again, the image derivatives toward the spatial directions y and z are similar. The vertex coordinates (x,y,z) are not necessarily integer coordinates, so it will generally be required to perform some form of interpolation to obtain the image values I. By substituting Equations (6.39) and (6.40) into Equation (6.38) and moving the image energy term to the right-hand side, the system of Equations (6.38) for each vertex can be written in matrix form, analogous to Equation (6.25) and its discretization. The matrix system assumes the form

$$\mathbf{A}\mathbf{x} = (\mathbf{D} + \mathbf{E} + \mathbf{F})\mathbf{x} = \mathbf{b} \tag{6.41}$$

$\mathbf{A}$ contains the discretization elements and the weight factors α and β, $\mathbf{b}$ contains the image energy terms, and $\mathbf{x}$ contains the vector of vertices. $\mathbf{A}$ can be decomposed into the diagonal elements $\mathbf{D}$, the lower left triangle matrix $\mathbf{E}$, and the upper right triangle matrix $\mathbf{F}$. With this decomposition, an iterative time-stepping scheme becomes possible:

$$\begin{aligned}
\xi^{T+1} &= \mathbf{D}^{-1}\left[\mathbf{b} - (\mathbf{E} + \mathbf{F})\mathbf{x}^T\right] \\
\mathbf{x}^{T+1} &= \mathbf{x}^T + \kappa(\xi^{T+1} - \mathbf{x}^T)
\end{aligned} \tag{6.42}$$

Application of Equation (6.42) advances the position vector $\mathbf{x}$ from time step T to time step $T+1$. The additional parameter κ improves convergence behavior through

underrelaxation and should be chosen to be $\kappa < 1$. Like snakes, the active net tends to lock onto local minima, so the final convergence result will depend on the initial placement of the ellipse and on the choice of the weight parameters α, β, and γ.

Because of the high computational complexity, several modified approaches have been introduced. For example, Cohen and Cohen[8] suggested the use of finite-element modeling to reach the energy minimum in shorter time. Bro-Nielsen proposed a model of active cubes.[4] The term *active cubes* points at the fundamental difference between a three-dimensional active surface and the active cubes model: The active cube model simulates a filled solid. Active cubes combine ideas from snakes and active nets. Energy minimization is performed, where the total energy consists of the internal energy (bending and stretching energy) and the external energy. Contrary to the principle used in snakes, but similar to the active nets,[30] direct image intensity data are used for the external forces instead of a gradient, and the cubes deform toward highest image intensities. Active cubes are well suited to simulate and parametrize elastic deformation (e.g., knee bending or jaw movement).

6.3. LIVE-WIRE TECHNIQUES

The *live wire* is a highly interactive segmentation technique, aimed at aiding the user delineate an object boundary. The user selects a start point on the boundary. As the user hovers over the image with the mouse, the live wire tries to connect both points with an energy-minimizing path. The live wire tends to snap onto image edges. If the user is satisfied with the live-wire selection, the endpoint under consideration can be fixed and becomes the start point for the next segment. A live wire can be used to parametrize a path (e.g., a sulcus in a brain tomography image) or a closed contour. In the latter case it is possible to convert the live wire into a snake and continue with a snake algorithm for further processing.

In the context of live wires, the energy function is referred to as the *cost function*. The principle of minimizing an energy or cost functional are the same. Apart from the higher level of user interaction, there are two more differences between live wires and snakes. First, the live wire is a path between start and end vertices with a global cost minimum. This is a consequence of the second difference, that is, the formulation of a live wire as a graph. Cost functions have even more parameters and user-definable features than the energy functions of a snake, therefore, a training step generally precedes the actual segmentation. The training step automatically adjusts numerous parameters to minimize the cost function for the specific type of edge to be segmented. An example is given in Figure 6.14, where a live wire was used to segment features in a chest CT image. First, the live wire was trained to detect the relatively low-contrast edge of the vertebral cortex (part A). The associated cost function (part C) shows low-cost values (dark shades) predominantly in the vertebral region and in regions of similar contrast. When trained for the high-contrast edge of the lung (B), the cost function looks different (D). In this case, edges of very high contrast show the lowest cost (darkest shades). When the live wire is trained for the lung edge, the vertebral edge actually repels the live wire. The ability to get trained

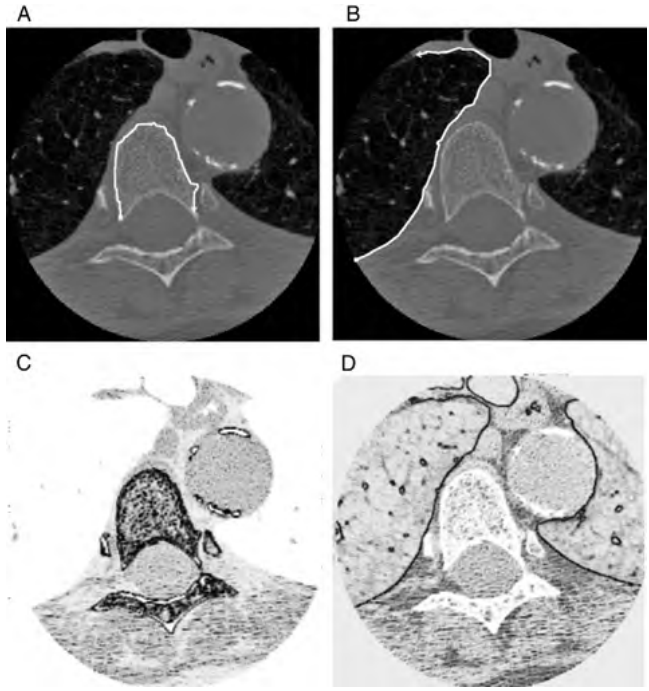


FIGURE 6.14 Examples of live wires (white curves beginning with a circle and ending in an arrow) following the cortex of a vertebra (A) and the lung boundary (C) in chest CT images. Images B and D show the corresponding cost functions after training, where dark shades indicate low-cost pixel boundaries, and light shades indicate high-cost boundaries. Interpreted as valleys, the low-cost boundaries attract the live wire, which tries to follow the lowest-cost path.

for a specific type of edge contrast and width combined with the flexibility to set an arbitrary number of waypoints (intermediate end vertices that become the start vertex of a new segment) make the live wire a very flexible interactive tool for boundary delineating and segmentation.

The live-wire model was introduced by Falcão et al.[12] The key to this model is the interpretation of each boundary between two pixels of an image as a graph arc and each corner of a pixel as a graph node. This configuration is sketched in Figure 6.15, which shows one pixel and its four nodes, which are shared with the neighboring pixels. Two nodes are labeled A and B, and a boundary exists that connects A and B. The boundary is actually a vector, and the boundaries A $\rightarrow$ B and B $\rightarrow$ A are not identical. Every boundary (i.e., each arc of the graph) gets assigned a set of features that will later be used to compute the cost. The process takes place in two steps. First, a cost image (Figure 6.14C and D) is computed that stores the cost for each boundary. This cost image may later be modified by the training process. Second, from the user-selected start pixel $P(x,y)$, the optimum path is computed for each pixel in the image, resulting in two two-dimensional arrays, the cumulative cost for the

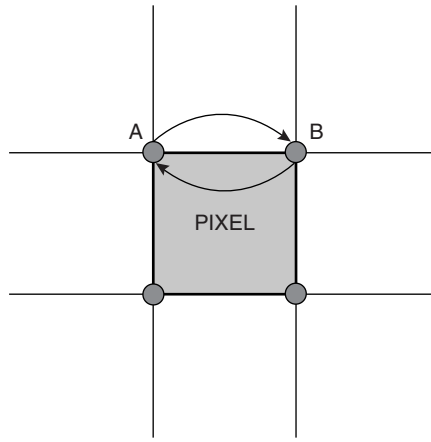


FIGURE 6.15 Interpretation of the image as a graph. Each pixel corner is a node (circle), and each pixel boundary that connects two nodes is a directed arc.

optimum path from the start pixel to each pixel in the image, and a two-dimensional array of directions indicating the immediate neighbor along the optimum path. The algorithm to obtain the arrays that contain the optimum path is explained below.

With the mouse hovering over any arbitrary pixel, the shortest path to the start pixel $P(x,y)$ can be found extremely fast by following the negative directions in the two-dimensional array of directions. This path can be displayed in real time, thus providing the main interactive element: The user can observe—in real time—where the active contour lies with respect to the image features and can add control points if necessary.

To understand the live wire algorithm and many related algorithms, two issues need to be considered: how to relate image features to cost, and how efficiently to find the lowest-cost path between two pixels of the image. To relate image features to cost, Falcão et al. proposed simultaneous use of eight feature functions[12] that are related to either image intensity or image gradients. Assume a directed boundary b between pixels p and q, where p lies on the left side of b and q on the right side. Feature functions f_1 and f_2 are simply the image intensity values of pixels p and q, respectively. Feature function f_3 is the magnitude of the difference between f_1 and f_2 (i.e., the absolute value of the gradient across the boundary b). Assuming the neighborhood definitions of the boundary b in Figure 6.16, feature functions f_4, f_5, and f_6 emerge as slightly different formulations of local gradients:

$$
\begin{aligned}
f_4 &= \frac{|I(r) + I(p) + I(t) - I(s) - I(q) - I(u)|}{3} \\
f_5 &= |0.25I(r) + 0.5I(p) + 0.25I(t) - 0.25I(s) - 0.5I(q) - 0.25I(u)| \qquad (6.43) \\
f_6 &= \frac{|I(p) - I(s)| + |I(r) - I(q)| + |I(p) - I(u)| + |I(t) - I(q)|}{4}
\end{aligned}
$$

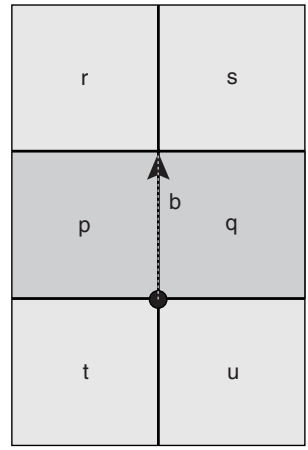


FIGURE 6.16 Definition of the pixel neighborhood of a boundary b.

Whereas f_3 through f_6 are scalar values, f_7 is an oriented magnitude, defined as $+f_6$ if the direction of the gradient remains consistent with respect to the boundary direction and $-f_6$ otherwise. An alternative and maybe more intuitive formulation would be the definition of f_5 without the magnitude function, that is,

$$f_7 = 0.25I(r) + 0.5I(p) + 0.25I(t) - 0.25I(s) - 0.5I(q) - 0.25I(u) \qquad (6.44)$$

The idea behind feature function f_7 is to force the live wire to stay on the same side of a slope. For example, if the gradient is positive (when walking along the boundary in the direction of the arrow, image values go uphill from left to right), the cost associated with f_7 is low. If the direction of the gradient changes, and the hill (positive gradient) is on the left side, the cost will go up.

Finally, feature function f_8 is the projected distance from a previously accepted live wire in a different image slice or in a previous image frame. Function f_8 exists only if (1) the user analyzes a stack of images (either a three-dimensional image or a sequence of images evolving over time), (2) the evolution of the feature between successive slices is small, and (3) one initial slice has already been segmented.

The values of the feature functions are now converted to costs by means of cost functions, also called *feature transforms*. Six feature transforms, $c_1(f)$ through $c_6(f)$ have been proposed,[12] and the final cost of a boundary b, $c(b)$, is the weighted sum of all six cost functions, in its most general form,

$$c(b) = \frac{\sum_{i=1}^{6} \sum_{j=1}^{8} w_{ij} c_i(f_j(b))}{\sum_{i=1}^{6} \sum_{j=1}^{8} w_{ij}} \qquad (6.45)$$

Equation (6.45) reflects the notion that any feature function can be used with any feature transform. The weight factors w_{ij} make it possible to emphasize one feature function over others.

The individual feature transforms are defined by the following mapping functions. Mapping function c_1 linearly maps the interval $[l_1,h_1]$ of the feature value to the interval [0,1]. Values of f above h_1 are clamped to 1, and values below l_1 are clamped to 0. The inverse linear mapping function c_2 can be defined as $c_2 = 1 - c_1$. It is possible, however, to use a different interval $[l_2,h_2]$. Mapping function c_3 is a Gaussian function with the constant mean value f_0 and the standard deviation σ. Mapping function c_4 is an inverted Gaussian function, for example $c_4 = 1 - c_3$, where mean and standard deviation may differ from c_3. Finally, mapping functions c_5 and c_6 are clamped hyperbolic functions with $c_6 = 1 - c_5$ and c_5 defined as

$$c_5(f) = \begin{cases} 1 & \text{for } f \leq l_3 + a^2/2 \\ (a^2/2)(f - l_3) & \text{for } l_3 + a^2/2 \leq f \leq h_3 \\ 0 & \text{for } f > h_3 \end{cases} \tag{6.46}$$

Again, c_5 and c_6 do not necessarily use the same threshold values l_3 and h_3. Here a is a variable parameter that represents the focus of the hyperbola from its asymptotes, l_3 is the distance of the vertical asymptote from the origin, and h_3 is the upper cutoff value. Each pair of mapping functions provides a different emphasis. The Gaussian functions c_3 and c_4 provide high and low costs, respectively, for values near the center of the interval. The hyperbolic functions c_5 and c_6 provide a strong cost emphasis for the extrema of the interval.

The flexibility introduced with eight feature functions and six mapping functions is impractical, with each mapping function providing at least two variable parameters. Imagine a user interface where each parameter is represented by a user-modifiable slider. In the extreme case, this user interface would present the user with 14 parameters for the individual feature transforms and 48 weight parameters. In practical implementations, only a few combinations of feature functions, mapping functions, and their respective parameters will be used. Furthermore, some parameters can be assigned or refined through the training process, which is described below.

The next step in the live wire implementation is the algorithm to determine the lowest-cost path from the start pixel $P(x,y)$ to each pixel of the image. The problem of finding the shortest path in a graph has been given much attention in the literature, and in conjunction with live wire implementations, Dijkstra's algorithm[11] is most frequently used. Most fundamental to the optimum path search algorithm is the property of optimum subpaths. If an optimum path from node P to node Q goes over node S, the path from P to S must also be the optimum path. Taken to the pixel level, the path from the starting point $P(x,y)$ to any arbitrary endpoint $Q(x,y)$ in the image plane consists of hops along the lowest-cost arc out of four possible choices from one pixel vertex to the next.

Dijkstra's algorithm can be implemented with the following steps after it is provided with a starting vertex P.

Step 1. Create a data structure that contains, for each vertex v of the image, a cumulative cost $cc(v)$, a direction for the optimum path to the immediate predecessor vertex, and a flag that indicates the processing status of vertex v. The direction may assume the values (U, W, N, E, S), where U stands for undefined. The flag may assume three values, (U, V, Q), where U stands for unprocessed, V stands for visited, and Q stands for queued.

Step 2. Initialization: All cumulative cost values in the data structure are set to infinity (a very high value) except the cumulative cost associated with the starting vertex, which is set to zero. All status flags are set to U (unprocessed). All directions associated with the vertices are set to U (undefined).

Step 3. Begin iteration: For vertex P, set its status flag to Q (queued).

Step 4. Iteration: As long as there are queued vertices, repeat steps 5 through 8.

Step 5. From all vertices that are queued, pick the one with the lowest associated cumulative cost. This is now vertex v_i. Change the status flag of v_i from Q to V.

Step 6. Examine all four neighbors of v_i: Perform steps 7 and 8 for each of the four neighbors provided that its associated flag has the value U. The neighbor under consideration is v_j.

Step 7. Compute a temporary cost $c_T = \text{cc}(v_i) + \text{cost}(v_j \rightarrow v_i)$, which is the cumulative cost of vertex v_i plus the cost of reaching v_i from v_j.

Step 8. If $c_T < \text{cc}(v_j)$, (a) update the cost $\text{cc}(v_j) = c_T$ and set the direction of v_j to point to v_i, and (b) set the status flag of v_j to Q.

Step 9. The iteration ends when no vertices are flagged with a Q (i.e., the queue is empty).

At the end of Dijkstra's algorithm, a cost structure exists for each pixel in the image. The cost functions described earlier come into play in step 7, where the cost of one arc (represented by the function cost $v_j \rightarrow v_i$) is added to the cumulative cost of an existing path. There are modifications of the algorithm above which make use of ordered lists to reduce the time needed to search for the queued vertex with the lowest cumulative cost. The algorithm described above is a good starting point and is sufficiently efficient for most applications, considering that the algorithm is only executed every time a new starting point is selected. Once the algorithm has finished, the lowest-cost path from any arbitrary point in the plane to the start point P is found simply by following the directions stored with each vertex. This is an extremely fast process which allows updating the path in real time.

Another variation of the algorithm uses the pixel centers rather than the corners as nodes. In essence, this moves the grid by $\frac{1}{2}$ pixel in each direction and uses an arc that goes from pixel center to pixel center. Here the feature functions need to be adjusted to reflect the fact that an arc no longer coincides with a pixel boundary. Using the definitions in Figure 6.17, feature functions f_1 and f_2 could be represented by the

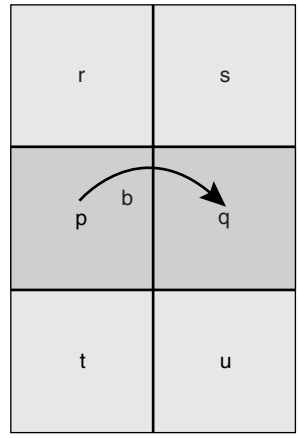


FIGURE 6.17 Definition of a pixel neighborhood for a center-to-center arc.

averaged intensities in pixels r and s, and t and u, respectively. Feature functions f_3, f_4, and f_5 are now identical and are the magnitude of the difference between f_1 and f_2. Feature function f_6 now becomes the average of the absolute difference between the intensities of r and u, and t and s, respectively, but it is doubtful whether this feature function would cause a measurably different behavior of the live wire from f_3. Finally, feature function f_6 is the simple difference between f_1 and f_2, so that the gradient direction can be considered. With the function f_8 unchanged, the number of feature functions is now reduced to five. This variation reduces the level of complexity at the expense of some of the ability to follow extremely thin structures.

The last aspect of the live wire technique to be considered is the training process. Ideally, training is based on a set of user-selected points that represent typical feature properties. Once a segment of the live wire is established and the user is satisfied with the behavior of the live wire, the system may be retrained to include a larger number of points, and ideally, also directions. Unfortunately, for the large number of freely selectable parameters, no straightforward training scheme exists. For a single image feature and a single cost function, a least-squares approach to determine suitable parameters from several user-selected points may be used, but the complexity of this approach increases unacceptably with additional feature and cost functions. Arguably the most important mapping functions are the Gaussian mapping functions, because they can be trained to minimize the cost for any gradient strength. An example was given in Figure 6.14, where the bone–tissue interface shows a lower gradient than the lung–tissue interface. In regard to the Gaussian mapping function, the cost minimum (the mean of the inverted Gaussian function c_4) would match the gradient strength. If the boundary is very homogeneous, a small standard deviation may be chosen. In fact, the live wire shows very good detection of features with intermediate contrast when only the Gaussian mapping function is used and the following training steps are taken.

Step 1. The user selects four to eight points along one representative boundary. For each point, the values of feature function f_3 are collected, and their mean and standard deviation is computed. These values are used as initial trained values f_0 and σ for mapping function c_4.

Step 2. Based on the values of step 1, a live wire is drawn. Once the user accepts the course of the live wire, mean and standard deviation for feature function f_3 for all pixels along the present live wire are computed and used as new values for f_0 and σ.

Step 3. Optionally, the user may, in a separate training step, select a number of points that do *not* represent the boundary. For these points the minimum and maximum values are computed and used as threshold values in the hyperbolic cost functions c_5 and c_6. By adding contributions from these cost functions, points that are an unlikely part of the contour sought make the path more expensive and therefore repel the live wire.

An alternative feature mapping technique was proposed by Marx and Brown.[20] Here, two equally weighted inverse Gaussian mapping functions (c_4) are used in conjunction with feature functions f_1 and f_2. During the training step, values for f_1 and f_2 are sampled, and the mean and standard deviation values are used as values f_0 and σ in the Gaussian mapping functions. Clearly, the effort to experiment with several different combinations of feature functions and mapping functions pays off when an optimum combination for a specific application is found.

The specific feature function f_8 was mentioned briefly. This function may be used when advancing frame by frame in an image sequence or when using the live wire to move through a three-dimensional image stack. Feature function f_8 makes the path more expensive when it deviates more strongly from the path chosen in the previous frame or slice. A more advanced technique was proposed by Marx and Brown.[20] For progression through image slices or successive image frames, the closed-contour live wire is converted into a snake. For this purpose, snake vertices are sampled along the live wire with a curvature-dependent density. The snake is allowed to converge on the next slice and gets converted back into a live wire by performing a training run along the snake contour. A related technique for the progression from slice to slice was presented by Salah et al.[24] In this variant, the live wire control points (the start points of the segments) are reexamined upon progression to the next slice. To advance to the next slice, the local neighborhood of the control points is examined for the cost minimum according to the cost function selected, and the control points are then moved to the new minimum. This method that can be implemented with an unsupervised algorithm. After moving the control points, the live wire searches the optimum path between the control points by using conventional methods. The live wire concept can also be extended into three-dimensional by a different technique, proposed by Hamarneh et al.[14] The basic idea is to use seed points in different slices to create orthogonal live wires. Points along the mesh of live wires generated by this method can be sorted to generate meaningful additional (and automatically generated live wires) that eventually provide the contour of the three-dimensional object being segmented.

Live wires are designed to find the globally optimal path. However, it is possible to combine the path cost term with an internal cost term, analogous to the internal energy of a snake.[15] The inclusion of an internal energy term requires extension of Dijkstra's optimum path search algorithm to three dimensions. With the well-known weight parameters α and β for stretching and curvature, the modified live wire, termed a *G-wire*, tends to follow a smoother path and can be prevented from digressing to neighboring similar features.

6.4. BIOMEDICAL EXAMPLES

The primary use of active contours in biomedical imaging is user-guided segmentation. Active contour models help improving accuracy and repeatability of the segmentation results.[2] In addition, user-guided segmentation techniques such as snakes and active contours speed up the segmentation process, since the user specifies only a few points instead of tracing the entire contour.

After their inception, snakes were almost instantly used in medical image segmentation and registration. Examples include the contour extraction and segmentation of neuronal dendrites in electron micrographs,[5] the extraction of brain structures from MR images,[3,29] or the detection of blood–wall interfaces in CT, MR, and ultrasound images.[26] In most cases, application of the active contour model is one step in a sequence of image processing steps. The live-wire technique is used in a similar manner. Examples of studies that use medical images include the determination of the thickness of femoral cartilage in MR images of osteoarthritic patients,[28] the determination of cartilage volume in MR images,[13] and the segmentation of the left heart ventricle in electron-beam CT images.[31]

One interesting extension is the inclusion of image texture in the energy function.[18] In this example, texture is represented by the local variance. Therefore, the image becomes two-valued with an intensity and a local variance value assigned to each pixel. The rate of change, to be used in the potential energy function is then determined from both values. A boundary that attracts the snake may therefore be an intensity transition or a transition of the local variance. According to Lorigo et al.,[18] the two-valued method was more stable and less prone to leaking the contour into adjacent areas than was a conventional intensity-based snake.

Another example of using image information beyond the intensity gradient is a study where coronary arteries were segmented in three-dimensional CT images.[36] In this study, the crucial preprocessing step was the computation of a *maximum a posteriori* (MAP) estimation of the probability of each pixel belonging to a class of either blood, myocardium, or lung. The generation of a MAP image yielded a presegmented image where blood and myocardium areas already belonged to different classes. Active contours were used to separate the coronary arteries from other blood-filled regions, such as the aorta or heart chambers. The presegmentation step allowed the active contour algorithm to perform particularly robust final segmentation and subsequent three-dimensional reconstruction of the coronary arteries from two-dimensional active contours.

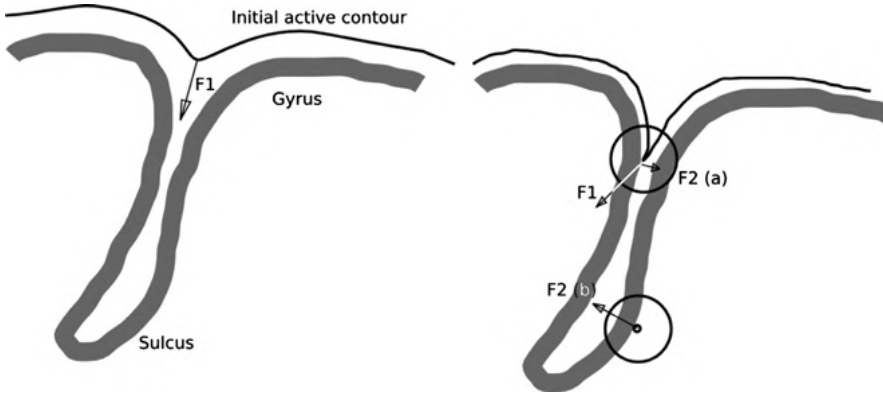


FIGURE 6.18 External forces to obtain a parametric representation of brain sulci. The initial active contour is pulled into the sulcus by a normal force F_1 that acts on every point of the active contour (left). A second force, F_2, is responsible for keeping the active contour centered within the sulcus as contour deformation into the sulcus takes place. Force F_2 is computed by determining the center of mass of the brain surface inside a small circle around any point of the active contour. Two examples are shown (right): a case (a) where the contour is almost centered and F_2 is close to zero, and a hypothetical case (b) where a point outside the sulcus is pulled back inside by a force caused by the mass of the outer cortical surface lying to the left of the point. (From ref. 32.)

An area where active contours are widely used is segmentation and mapping of the brain and brain structures. A three-dimensional active contour model introduced by Vaillant and Davatzikos[32] uses two forces to pull an elastic balloon model into the sulci (the valleys) of the brain surface. The forces are illustrated in Figure 6.18. A force F_1, acting normal to the active contour, pulls the elastic surface into the sulcus, while a force F_2, determined from the center of mass in a circle around each point of the contour, ensures that the active contour remains centered with respect to the sulcus. Such a model can be used for the registration of brain images[10] and to generate projections of the sulcal depth and shape, because the force acting on each point can be projected back on the initial contour.

Fourier descriptors have also been useful in the segmentation of MRI brain features,[29] in this case the segmentation of the corpus callosum. Through a training process, a mean model of the corpus callosum shape was obtained. Biological variation was analyzed by performing a principal component analysis of the covariance matrix of the Fourier coefficients, and it was found that the variance becomes very small after the twelfth eigenvector. Consequently, a flexible model was built from the first 12 eigenmodes. After training, two steps were applied to segment the corpus callosum: First, the Hough transform, together with a small set of dominant modes, was used to find the initial optimum placement of the mean model. Second, elastic deformation, restricted by the eigenmodes selected, was allowed to determine the ideal contour of the individual corpus callosum. In an extension, the two-dimensional model was extended into three-dimensional Fourier models.

An interesting application for two-dimensional active contours is the improvement in the three-dimensional surface representation of volumetric image features. Most frequently, biomedical images (primarily, CT and MRI) are acquired with an anisotropic voxel size; that is, the slice thickness in the z (axial)-direction is much larger than the in-plane resolution. A typical example would be a CT image with 0.2×0.2 mm pixel size in each slice with a slice thickness of 2 mm. Three-dimensional reconstructions from such a data set do not look smooth, but jagged, showing the steps in the z-direction. An example is shown in Figure 6.19, where a segmented lung obtained from a three-dimensional CT image was rendered. Resolution in the axial direction is about sixfold lower than in-plane resolution. Accordingly, the three-dimensional surface rendered shows steps and jagged regions. After interpolation, the resulting surface has a smoother appearance and allows better identification of image details. To obtain an interpolated surface from two-dimensional snakes, the object is first segmented in each slice by using a conventional snake algorithm. It is possible to progress from slice to slice by using the snake from the previous slice as initialization shape, because differences between slices are generally small. In the next step, new contours are created by interpolating between corresponding vertices of the existing snakes in the axial direction (Figure 6.20). With very low computational effort, a smooth surface can be obtained.

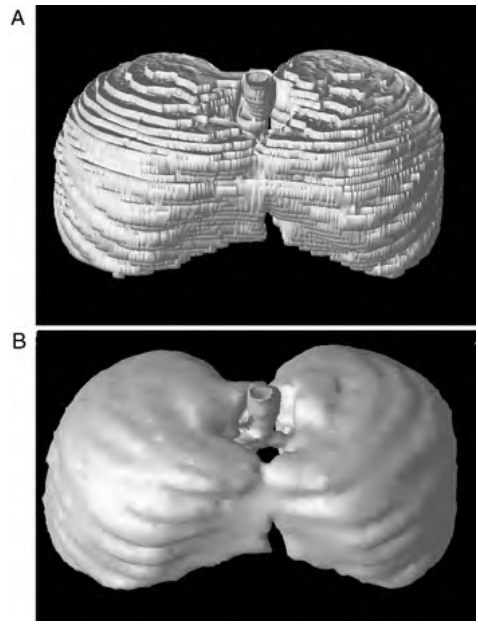


FIGURE 6.19 Demonstration of interpolation with active contours. Three-dimensional renderings such as the segmented lung in this example show jaggedness and steps because of the anisotropic voxel size of most CT and MRI images. Once a parametrized contour is obtained, it is straightforward to create new vertices by interpolation between slices. The resulting surface rendering (B) is smoother and allows better focus on the details.

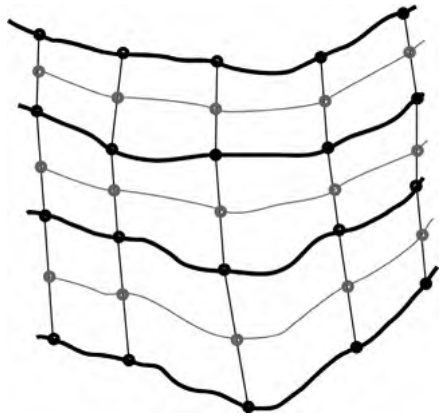


FIGURE 6.20 Creating new contours through interpolation between slices. In each slice, a snake (dark lines) is defined by its vertices (black circles). New vertices can be created by interpolating in the axial direction (gray circles) and by using these vertices to generate new contours (gray lines).

A similar method has been proposed by Raya and Udupa,[22] where an additional step is introduced: The segmented contour is converted into a gray-scale distance map with positive values inside the contour and negative values outside the contour. These gray-value slices are then interpolated and converted to binary by detection of zero crossings. A combination of the live-wire technique with this type of gray-scale interpolation has been used by Schenk et al.[25] for the semiautomatic segmentation and three-dimensional reconstruction of the liver in CT images. A different approach was pursued by Leventon and Gibson.[17] In this approach, two orthogonal scans are segmented by using surface nets, and the nets are linked and relaxed. The resulting surface is considerably smoother than the surface of one single data set.

Because of its closeness to physical models, it is possible to use active contours and active surfaces for the simulation of material elasticity. For example, Bro-Nielsen[4] used elastic active cubes to simulate soft tissue deformation caused by movement of the jaw. Roth et al.[23] used a finite-element approach to simulate changes of facial features under planned cosmetic surgery to be used as a tool for preoperative planning. One step further ahead, deformable models can be integrated into a neurosurgery simulation system[33] capable of simulating the tissue reaction to cutting, pulling, and prodding. A further enhancement of the experiment was achieved through three-dimensional binocular vision and force feedback.

The basic mathematical foundation and computer implementation of deformable models are well established and in widespread use. Nonetheless, the field of deformable models is very extensive and under active research. Three-dimensional active surfaces are of particular interest in the biomedical community for the shape analysis of three-dimensional objects with practical applications in image registration and the relationship between shape and disease.

REFERENCES

1. Amini AA, Tehrani S, Weymouth TE. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints. Proc 2nd Int Conf Comput Vis 1988; 95–99.
2. Ashton EA, Molinelli L, Totterman S, Parker KJ, VirtualScopics LLC, Rochester NY. Evaluation of reproducibility for manual and semi-automated feature extraction in CT and MR images. Proc Int Conf Image Process 2002.
3. Atkins MS, Mackiewich BT. Fully automatic segmentation of the brain in MRI. IEEE Trans Med Imaging 1998; 17(1):98–107.
4. Bro-Nielsen M. Modelling Elasticity in Solids Using Active Cubes: Application to Simulated Operations. Lecture Notes in Computer Science. New York: Springer-Verlag, 1995: 535.
5. Carlbom I, Terzopoulos D, Harris KM. Computer-assisted registration, segmentation, and 3 D reconstruction from images of neuronal tissue sections. IEEE Trans Med Imaging 1994; 13(2):351–363.
6. Caselles V, Catté F, Coll T, Dibos F. A geometric model for active contours in image processing. Numer Math 1993; 66(1):1–31.
7. Cohen LD. On active contour models and balloons. CVGIP: Image Underst 1991; 53(2):211–218.
8. Cohen LD, Cohen I. Deformable models for 3D medical images using finite elements and balloons. Proc CVPR 1992; 2–9.
9. Cohen LD, Cohen I. Finite-element methods for active contour models and balloons for 2-D and 3-D images. IEEE Trans Pattern Anal Mach Intell 1993; 15(11):1131–1147.
10. Davatzikos C. Spatial transformation and registration of brain images using elastically deformable models. Comput Vis Image Underst 1997; 66(2):207–222.
11. Dijkstra EW. A note on two problems in connection with graphs. Num Math 1959; 1:269–271.
12. Falcão AX, Udupa JK, Samarasekera S, Sharma S, Hirsch BE, Lotufo RA. User-steered image segmentation paradigms: live wire and live lane. Graph Models Image Process 1998; 60(4):233–260.
13. Gougoutas AJ, Wheaton AJ, Borthakur A, Shapiro EM, Kneeland JB, Udupa JK, Reddy R. Cartilage volume quantification via live wire segmentation. Acad Radiol 2004; 11(12):1389–1395.
14. Hamarneh G, Yang J, McIntosh C, Langille M. 3D live-wire-based semi-automatic segmentation of medical images. Proc SPIE 2005; 5747:1597–1603.
15. Kang HW. G-wire: A livewire segmentation algorithm based on a generalized graph formulation. Pattern Recogn Lett 2005; 26(13):2042–2051.
16. Kass M, Witkin A, Terzopoulos D. Snakes: active contour models. Int J Comput Vision 1988; 1(4):321–331.
17. Leventon ME, Gibson SFF. Model Generation from Multiple Volumes Using Constrained Elastic Surface Nets. Lecture Notes in Computer Science. New York: Springer-Verlag, 1999; 388–393.
18. Lorigo LM, Faugeras O, Grimson WEL, Keriven R, Kikinis R. Segmentation of Bone in Clinical Knee MRI Using Texture-Based Geodesic Active Contours. Lecture Notes in Computer Science. New York: Springer-Verlag, 1998; 1195–1204.

19. Malladi R, Sethian JA, Vermuri BC. Shape modeling with front propagation: a level set approach. IEEE Trans Pattern Anal Mach Intell 1995; 17(2):158–175.
20. Marx S, Brown CM. Progressive livewire for automatic contour extraction. Proceedings, Western New York Image Processing Workshop, 2004; 2–23.
21. McInerney T, Terzopoulos D. T-snakes: topology adaptive snakes. Med Image Anal 2000; 4(2):73–91.
22. Raya SP, Udupa JK. Shape-based interpolation of multidimensional objects. IEEE Trans Med Imaging 1990; 9(1):32–42.
23. Roth SH, Gross MH, Turello S, Carls FR. A Bernstein–Bézier based approach to soft tissue simulation. Comput Graphics Forum 1998; 17(3):285–294.
24. Salah Z, Orman J, Bartz D. Live-wire revisited. Proc BVM 2005.
25. Schenk A, Prause G, Peitgen HO. Efficient Semiautomatic Segmentation of 3D Objects in Medical Images. Lecture Notes in Computer Science. New York: Springer-Verlag, 2000: 186–195.
26. Sebbahi A, Herment A, De Cesare A, Mousseaux E. Multimodality cardiovascular image segmentation using a deformable contour model. Comput Med Imaging Graph 1997; 21(2):79–89.
27. Staib LH, Duncan JS. Boundary finding with parametrically deformable models. IEEE Trans Pattern Anal Mach Intell 1992; 14(11):1061–1075.
28. Steines D, Cheng C, Wong A, Tsai J, Napel S, Lang P. Segmentation of osteoarthritic femoral cartilage using live wire. ISMRM Eighth Scientific Meeting, Denver, CO, 2000.
29. Székely G, Kelemen A, Brechbühler C, Gerig G. Segmentation of 2-D and 3-D objects from MRI volume data using constrained elastic deformations of flexible Fourier contour and surface models. Med Image Anal 1996; 1(1):19–34.
30. Takanashi I, Muraki S, Doi A, Kaufman A. 3D active net for volume extraction. Proc SPIE 1998; 3298(184):193.
31. Urschler M, Mayer H, Bolter R, Leberl F. The LiveWire Approach for the Segmentation of Left Ventricle Electron-Beam CT Images. 26th Workshop of the Austrian Association for Pattern Recognition (AGM/AAPR), 2002; 319–326.
32. Vaillant M, Davatzikos C. Finding parametric representations of the cortical sulci using an active contour model. Med Image Anal 1997; 1(4):295–315.
33. Wang P, Becker AA, Jones IA, Glover AT, Benford SD, Greenhalgh CM, Vloeberghs M. A virtual reality surgery simulation of cutting and retraction in neurosurgery with force-feedback. Comput Methods Programs Biomed 2006; 84(1):11–18.
34. Williams DJ, Shah M. A fast algorithm for active contours and curvature estimation. CVGIP: Image Underst 1992; 55(1):14–26.
35. Xu C, Prince JL. Snakes, shapes, and gradient vector flow. IEEE Trans Image Process 1998; 7(3):359–369.
36. Yang Y, Tannenbaum A, Giddens D, Coulter W. Knowledge-based 3D segmentation and reconstruction of coronary arteries using CT images. Conf Proc IEEE Eng Med Biol Soc 2004; 3:1664–1666.

7

THE HOUGH TRANSFORM

The Hough transform is a tool to detect and quantify shape primitives in images, particularly in the presence of noise. The Hough transform is a robust tool to extract features (such as straight edges, circles, or ellipses, but also primitives defined by polygons) from images and describe them parametrically. The Hough transform is generally used as one step in a processing chain. First, the image needs to be segmented such that the edge of the shape of interest becomes a prominent feature. Following segmentation, the Hough transform serves as a filter for primitives. With the Hough transform, geometric primitives can be detected and separated from image noise. Implicitly, the primitives can be quantified (e.g., circle radius and position). The strength of the Hough transform to find, extract, and quantify shapes lies in the ability to identify those shapes and features even if the outline is broken, incomplete, or corrupted by noise in the thresholded image.

The original Hough transform, developed by Paul Hough in 1962,[10] was designed to identify straight lines in images. In 1972 the Hough transform was refined by R. Duda and P. Hart,[4] who introduced the line representation in polar coordinates and proposed an extension toward ellipses. A further milestone was the generalization by D. H. Ballard,[2] who not only presented a generalization for analytic curves but also proposed the use of shape templates (see also Merlin and Faber[15] for an alternative generalization).

The idea behind the Hough transform is to transform the shape of interest into its parameter space. For example, a line in a Cartesian (x,y) coordinate system can be

Advanced Biomedical Image Analysis, By Mark A. Haidekker

described by the equation

$$y = mx + n \tag{7.1}$$

with the constants m (the slope of the line) and n (the y intercept). Each line is uniquely characterized by the pair of constants m and n. Therefore, any line can be represented by a point in a coordinate system of m and n. Conversely, any point (x,y) is associated with a set of values for m and n that satisfy Equation (7.1), which can be rewritten as

$$m = \frac{y}{x} - \frac{1}{x}n \tag{7.2}$$

which, for constant x and y, is another line equation in a (m,n) coordinate system. Therefore, each point (x,y) is represented by a line in (m,n)-space. It can be seen from Equation (7.2) that the line equation (7.1) is unsuitable, since values in (m,n)-space become undefined for vertical lines. In the next section, a different line equation will be presented, but the principle of the Hough transform can best be explained with the basic equation (7.1).

When a set of points (y_k,x_k) that lie on a line described by $y = Mx + N$ is transformed into (m,n)-space (called Hough space or *parameter space*), each point is represented by a line in Hough space:

$$m = \frac{y_k}{x_k} - \frac{1}{x_k}n \tag{7.3}$$

All lines meet in one point (M,N), as shown in Figure 7.1. To extract the feature (in this section, the line), a second step is necessary. In Figure 7.1, only six points of the gray line are examined. Each point is now transformed into Hough space, and the traces of the resulting lines are accumulated. Starting with an empty image matrix (i.e., all elements are zero) in Hough space, the lines that correspond to each point

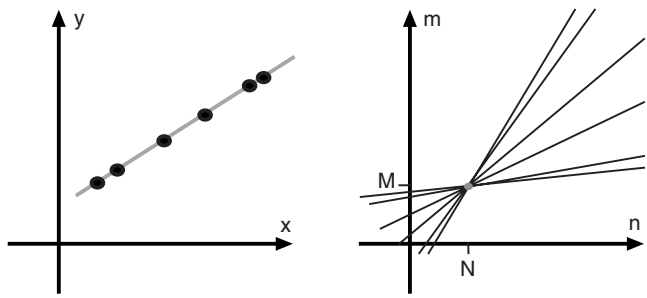


FIGURE 7.1 Principle of the Hough transform. Individual points in (x,y)-space are represented by lines in Hough space. If all points lie on a line $y = Mx + N$, all lines in Hough space will meet at point (M,N).

are traced, and the image values along the trace are incremented by 1. This process is often described as the point (y_k,x_k) casting a vote in Hough space, and the process of accumulating votes is common to all varieties of Hough transforms. In the process, the Hough-space pixel at (M,N) is incremented for each line and therefore accumulates the most votes. Peak detection would now provide a single point, (M,N), and the reconstructed equation for the line is $y = Mx + N$.

7.1. DETECTING LINES AND EDGES WITH THE HOUGH TRANSFORM

To use the Hough transform to extract lines, the equation of a line must be rewritten to prevent undefined values that occur in Equation (7.2) for vertical lines. Instead of using slope and intercept, a line can be described by its distance to the origin, ρ, and its angle with the x-axis, θ (Figure 7.2). With these two parameters, the line is now described by

$$x \cos\theta + y \sin\theta = \rho \tag{7.4}$$

ρ and θ are related to m and n through

$$\begin{aligned} m &= -\frac{\cos\theta}{\sin\theta} \\ n &= \frac{\rho}{\sin\theta} \end{aligned} \tag{7.5}$$

The corresponding Hough space is two-dimensional, with coordinates ρ and θ. Following the same idea as in Equation (7.3), we can ask what type of trace an arbitrary point (y_k,x_k) would create in a coordinate system of ρ and θ. For this purpose we assume x and y to be constant: namely, (x_k,y_k) for the kth point in any set. We then insert

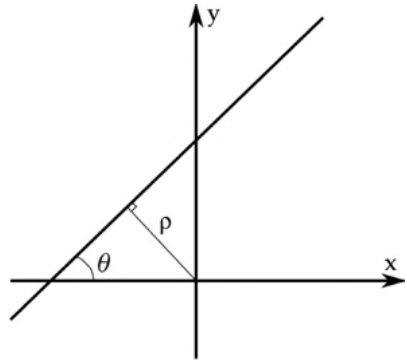


FIGURE 7.2 Definition of a straight line in polar coordinates [Equation (7.4)]. θ is the angle of the line with the horizontal axis, and ρ is the minimum distance to the origin.

(x_k,y_k) in Equation (7.4), combine the terms with θ, and solve for ρ. The resulting equation describes a sine function:

$$\rho = \sqrt{x_k^2 + y_k^2} \cdot \sin(\theta + \varphi) \tag{7.6}$$

where $\tan \varphi = x_k / y_k$. The sine function is always centered on $\rho = 0$, and its amplitude and phase shift are identical to the location of the point (y_k,x_k) in polar coordinates with an additional 90° phase shift.*

For the accumulating step, an arbitrary angular increment $\Delta\theta$ with $\theta_j = j\Delta\theta$ is chosen, and for each point of interest (y_k,x_k) the corresponding set of values $\rho(\theta_j)$ is computed using Equation (7.4). Since Hough space is discrete, the pixel at (ρ, θ_j) in Hough space is increased for the kth point at each angle θ ($0 \leq \theta < 180°$). Each point therefore produces a sinusoidal trace in Hough space following Equation (7.6). The restriction of θ to angles below 180° is possible because of the symmetry of Equation (7.4). In the terminology of the Hough transform, the point (y_k,x_k) casts j votes for all angles θ_j in Hough space. An example is given in Figure 7.3. First, an edge detector extracts the edges of the features (Figure 7.3B). For the actual Hough transform seen in Figure 7.3C, each white pixel from the edge is selected and Equation (7.4) is applied 180 times with θ increasing from 0° to 179° at 1° intervals. The Hough transform image is therefore 180 pixels high, and each row corresponds to the angle θ in degrees. The points where multiple sinusoidal traces meet are brighter than other points. Once the votes for all points (y_k,x_k) have been accumulated in Hough space, a peak-finding algorithm or simple thresholding determines the pairs (ρ, θ) with the highest number of votes, and each of these pairs defines a straight line in Cartesian space. In the case of Figure 7.3C, the brightest pixel is found at the coordinates $\rho = 21$ and $\theta = 29°$. Application of Equation (7.5) yields the values $m = 0.55$ and $n = 43$ for the equation of the line representing the edge.

The following example illustrates the advantages and challenges of the Hough line transform. Figure 7.4 shows an x-ray image of a mouse bone together with its edge image. The edge image was created by applying the Sobel operator, thresholding, and then superimposing the edge over the Sobel image. As a consequence, the actual edges are visible as white lines in the Sobel image. These white lines are composed of the pixels that cast a vote in the Hough transform.

The Hough transform of the edges in Figure 7.4B is shown in Figure 7.5. Two areas of high votes (two local maxima) become evident. The left local maximum is well defined, whereas the right maximum is more spread out along a curve. These areas of local intensity maximum correspond to the upper and lower edges of the bone, respectively. Since the upper edge follows more closely a straight line, its local maximum in the Hough transform is more focused. Conversely, the lower edge is curved. Therefore, its corresponding local maximum is less focused and spread

*The line Hough transform is closely related to the Radon transform in computed tomography. The *sinogram,* the collection of all projections of an object (see Figure 3.13), has the same shape as the line Hough transform because each x-ray absorbing point of the object at (y_k,x_k) creates a sinusoidal trace that is described by Equation (7.6) in the sinogram—hence its name.

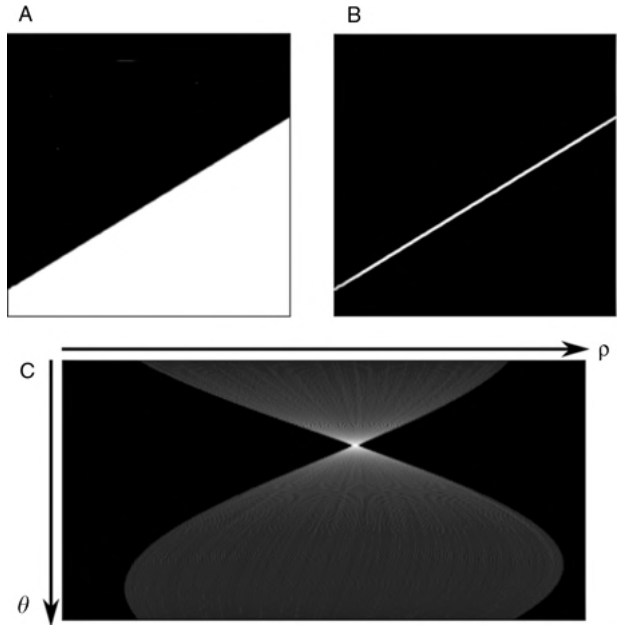


FIGURE 7.3 Example image to demonstrate the Hough transform. Image A contains a feature with a dominant diagonal edge. In the first step, the edge is extracted by an edge detector such as the Sobel operator (B). Each point of image B that lies above a threshold (i.e., is considered as belonging to the edge) is allowed to cast a vote for all angles θ, and each point contributes votes along a sinusoidal curve in (ρ, θ) space (C). The traces meet in a single point, and the coordinates ρ and θ coincide with the distance from the origin and the angle with the x-axis of the edge in image B. Image C was contrast-enhanced with a gamma function to improve visibility of the nonmaxima traces.

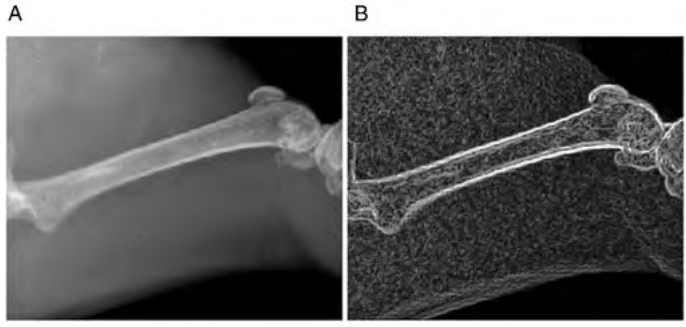


FIGURE 7.4 X-ray image of a mouse femoral bone (A) and the edges detected (B). Image B shows the edges as white lines and line segments superimposed over the Sobel image.

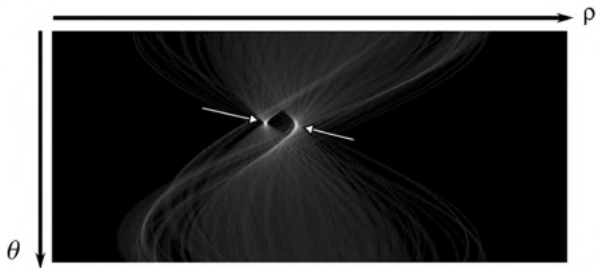


FIGURE 7.5 Hough transform of the edges in Figure 7.4B. Two areas with local intensity maxima (arrows) become evident. These areas correspond to the upper and lower edge of the bone. The image has been contrast-enhanced with a gamma function to improve the visibility of the traces.

over multiple values of ρ and θ. To identify the likely edge candidates, a local maximum filter needs to be applied next. In this example the filter found two pixels with an equal number of votes for the right local maximum (the lower, curved edge). Consequently, the inverse Hough transform contains two closely aligned lines for the lower edge. The inverse Hough transform superimposed over the original x-ray image is shown in Figure 7.6. It can be seen that the straight lines generated by the inverse Hough transform follow the bone outline very closely. The remarkable advantage of the Hough transform is its ability to detect dominant lines even in the presence of noise. Noise is amplified by the edge detection operation (Sobel operator). As a consequence, edges may be fragmented, and fragments of other edges remain in the image. This is particularly clearly visible in Figure 7.4B, where the knee joint (top right) contributes numerous pixels above the threshold (i.e., pixels that cast votes in the Hough transform).

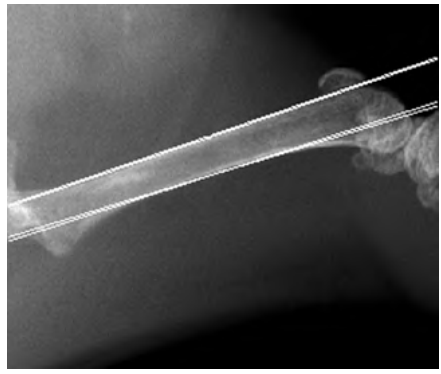


FIGURE 7.6 The inverse Hough transform of the local maxima in Figure 7.5 superimposed on the original x-ray image. The local maximum for the lower edge was ambiguous (two pixels of an identical number of votes), and two closely aligned lines emerge.

One disadvantage of the line Hough transform is that the lines are defined by two parameters only and therefore extend over the entire image. Additional steps are necessary to trim the lines. Examples include limiting the lines by the thresholded version of the original shape, and limiting the lines to a region of interest or by analyzing intersections of multiple lines. Moreover, the examples in Figures 7.5 and 7.6 demonstrate that local maxima can be ambiguous and the line extraction relies strongly on a robust local maximum filter.

The basic algorithm for the line Hough transform is presented in Algorithm 7.1. Each nonzero pixel of the input image is allowed to cast a vote for all possible angles θ over the full 180° angle range. For reasons of symmetry, the angles from 180° to 360° duplicate the votes from 0° to 180°, and computational efficiency can be increased by restricting the angles to 0° to 180°.

```
set deltatheta=10;                  // Choose a suitable Δ Θ
set hysize=180/deltatheta;          // Hough image Y size
set hxsize = max(xm,ym)*1.4142;     // Hough image X size
allocate IH(hxsize,hysize);         // Allocate output image

for (y=0 while y<hysize increment y=y+1) do
   for (x=0 while x<hxsize increment x=x+1) do
      IH(x,y) = 0;                  // Clear the accumulator to zero
   endfor;
endfor;

for (y=0 while y<ym increment y=y+1) do    // double loop over the image
   for (x=0 while x<xm increment x=x+1) do
      if (IM(x,y) > 0) then                // consider only nonzero pixels
         x1 = x-xm/2;
         y1 = y-ym/2;                      // Coordinate transform: image
                                             center is origin

         // Each nonzero pixel is allowed to cast votes over the full range of angles
         for (theta=0 while (theta<180) increment
                              theta=theta+deltatheta)
            rho = x1*cos(theta) + y1*sin(theta);   // Compute rho
            x2 = rho+hxsize/2;                     // coordinate transform for IH
            IH(x2,theta) = IH(x2,theta) + 1;       // Accumulate vote
         endfor;
      endif;
   endfor;
endfor;
```

Algorithm 7.1 Hough transform for lines. The input image `IM` (size: `xm`, `ym`) is assumed to be thresholded, and each nonzero pixel is allowed to cast votes for angles theta from 0 to 180° with the angular increment of `deltatheta`. The output image `IH` contains the accumulated votes with values of ρ along the *x*-axis and values of θ along the *y*-axis. A typical output image is shown in Figure 7.5.

Further improvement in the line detection process using the Hough transform is possible when the direction of the edge is known.[17] Further reduction of the angle range is possible by combining the Hough transform with the edge detector. The Sobel operator, for example, convolves the original image with two matrices, G_x and G_y [Equation (2.20)], to emphasize horizontal and vertical edges, respectively. The convolved images are combined on a pixel-by-pixel basis to provide the edge magnitude. The two convolved intermediate images contain additional information: If the pixel value computed by convolution with G_x is larger than the one obtained by convolution with G_y, the edge is horizontal (more precisely, between -45° and $+45^\circ$ from the x-axis), and the range of values for θ where the pixel casts a vote can be restricted to the range 45° to -45°. In the opposite case (convolution with G_y creates a larger pixel value than convolution with G_x), the range for θ can be restricted to the range 45° to 135°. The votes for each pixel can therefore deviate only by up to $\pm 45^\circ$ from the dominant edge direction. Further refinement is possible when the compass or Kirsch edge-detection operators are used (Figure 2.5). To emphasize edges with compass or Kirsch operators, four convolutions are performed, and the largest pixel value of the four convolution operations is taken as the edge value. In addition, the edge orientation is horizontal ($-22^\circ \leq \theta \leq 22^\circ$), upward diagonal ($23^\circ \leq \theta \leq 67^\circ$), vertical ($68^\circ \leq \theta \leq 112^\circ$), and downward diagonal ($113^\circ \leq \theta \leq 157^\circ$ or $-23^\circ \leq \theta \leq -68^\circ$) when the maximum pixel value results from convolution with K_1, K_2, K_3, and K_4 [Equation (2.24)], respectively. The vote of each pixel is therefore limited to an angular horizontal band of 45° width in the Hough transform. Figure 7.7 shows the Hough transform of the edge image in Figure 7.4B with the limitation of the angle range. The main difference in Figure 7.5 is the restriction that pixels from the knee region are not allowed to cast votes into the horizontal band ($68^\circ \leq \theta \leq 112^\circ$).

There are two advantages of the use of restricted ranges for θ. First, the absence of pixel traces from nondominant edges in the band containing dominant edges reduces the number of local maxima. Dominant edges are therefore easier to find and

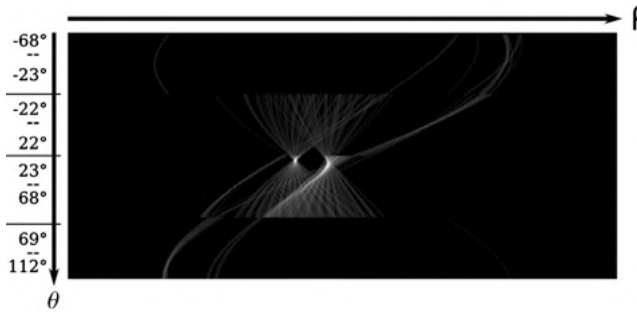


FIGURE 7.7 Hough transform of the edges in Figure 7.4B with angular restriction to $\pm 22^\circ$ from the edge orientation. The angular restriction divides the image into four horizontal sections, and pixels are not allowed to cast votes into sections that differ by more than $\pm 22.5^\circ$ from their main edge orientation.

isolate. Second, for each pixel only one-fourth of the original angle range needs to be computed. Provided that edge detection was required in both cases and that edge detection and Hough transform can be combined into one operation, computational efficiency increases markedly.

7.2. DETECTION OF CIRCLES AND ELLIPSES WITH THE HOUGH TRANSFORM

Many shapes in medical images can be approximated by circles or ellipses. The principal idea of the Hough transform, the accumulation of votes in parameter space, can be applied to both circles and ellipses, since both shapes can be described analytically. A circle is described by three parameters: radius r and the two center coordinates, x_c and y_c:

$$r^2 = (x - x_c)^2 + (y - y_c)^2 \tag{7.7}$$

Therefore, the Hough space for circles is three-dimensional. The projection of an individual pixel (x_k,y_k) into Hough space is a cone: If the pixel coordinates x and y in Equation (7.7) are assumed to be fixed with $x = x_k$ and $y = y_k$, Equation (7.7) is satisfied by an infinite set of concentric circles with center (x_k,y_k). The radius of the circles increases as the circles are translated along the r-axis (Figure 7.8). If a number of pixels that lie on a circle in image space are projected into Hough space, the ensuing cones meet at the point (x_c,y_c,r). Accumulating votes along the cones (analogous to the sinusoidal traces in the line transform) will therefore yield a maximum at (x_c,y_c,r) that can be used to reconstruct the circle.

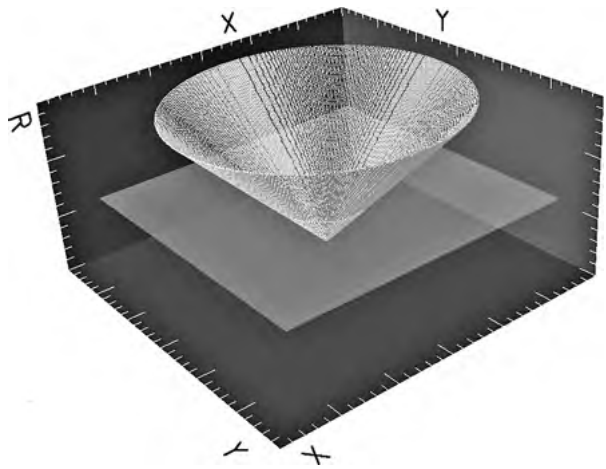


FIGURE 7.8 A pixel is projected as a cone in Hough space since a cone satisfies Equation (7.7) for increasing r. The cone tip rests at (x_k,y_k) for $r = 0$. The light gray plane is located at $r = 50$, and its intersection with the cone is therefore a circle of radius 50.

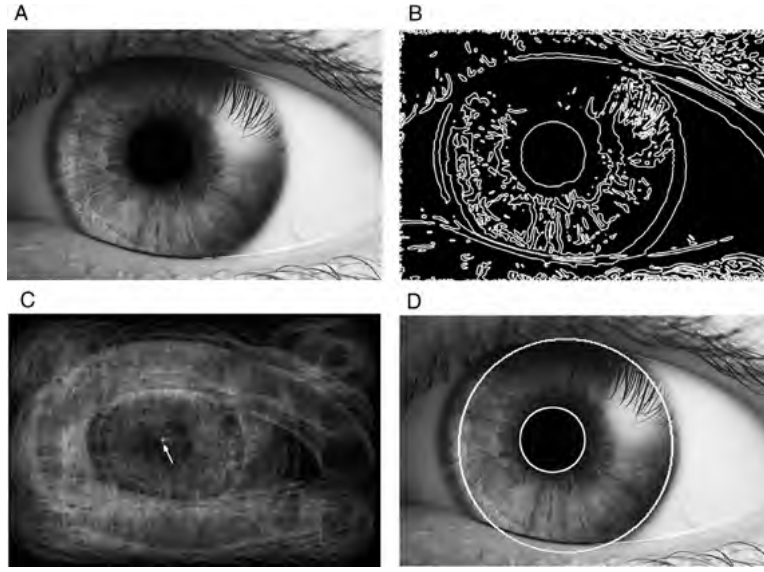


FIGURE 7.9 Detection of the pupil and iris with the Hough transform: (A) the original eye photo and (B) the edge image obtained by filtering, edge detection, and thresholding. (C) One plane in Hough space. In this plane ($r = 60$), the cones from the pupil pixels intersect and form a local maximum (arrow). The inverse Hough transform of this local maximum leads to the smaller circle in (D). Although the smaller circle represents the pupil very well, the larger iris circle is affected by various conflicting edges caused, for example, by the eyelids. As a result, the circle is smaller than the actual iris. [(A) From http://en.wikipedia.org/wiki/Image:Eye_iris.jpg.]

The circle Hough transform is frequently used to quickly determine the pupil diameter of the eye. An example is shown in Figure 7.9. In this example, isolation of the pupil and iris will be attempted. Whereas the pupil is well defined, the iris is partly occluded by the eyelids. In preparation of the Hough transform, image filtering and edge detection are necessary. Since edge detection increases the noise component, the image was first blurred by using a Gaussian convolution filter. Application of the Sobel edge detector was followed by manual thresholding. The resulting edge image is shown in Figure 7.9B. The pupil is represented by an almost perfect ring, while the iris caused multiple edges. Each pixel in Figure 7.9B is projected as a cone in Hough space. One plane in Hough space, similar to the plane in Figure 7.8, is shown in Figure 7.9C. In this plane at $r = 60$, the cones from the pupil pixels intersect and add up to a clear local maximum (indicated by an arrow). Multiple local maxima can be found at larger r values, because there are many edges related to the iris. The eyelids contribute additional edges. Therefore, several significant local maxima exist at large values of r in Hough space. It is not trivial to isolate the meaningful maxima, and a simple maximum-finding algorithm as used to generate Figure 7.9 does not necessarily find the pixel that best represents the iris circle. The two local

maxima that represent pupil and iris were finally subjected to the inverse Hough transform and yielded two circles. The two circles, superimposed over the original image of the eye, are shown in Figure 7.9D. In this case, the edge at the left side of the iris (Figure 7.9A) where light is refracted and causes a dark rim is most dominant, and the maximum-finding algorithm picked a pixel that represents a circle that is too small. More sophisticated algorithms and the use of a priori knowledge, such as the approximate radius, strongly improve the results obtained from Hough transform-based methods.

An additional validation method would be the comparison of the reconstructed circles to the original edge image. Image values from the edge image can be averaged along the reconstructed circles and the average values thresholded to eliminate false positives. Furthermore, if the gradient information from the edge detection step is available, the circle detection algorithm can be further improved by using the first derivative of the circle.[2] Consider the parametric form of the circle,

$$\begin{aligned} x(\phi) &= r\cos\phi \\ y(\phi) &= r\sin\phi \end{aligned} \tag{7.8}$$

where ϕ is the angle of the line connecting the circle's center with a point (x,y) on the circle. From Equation (7.8), the gradient follows:

$$\frac{dx}{dy} = -\frac{\sin\phi}{\cos\phi} = -\tan\phi \tag{7.9}$$

Here $\Phi = 90^\circ - \phi$ is the angle of the circle's tangent. If the angle Φ is available, the additional constraint reduces each circle of the cone to two points, and the point (x,y) with known gradient angle Φ is represented by two slanted lines in Hough space. In practice, however, a larger angular range needs to be considered because of the discrete nature of the edge detectors. With the Sobel edge detector and the compass edge detector, the angle range can be reduced to 90° and 45°, respectively. Each point of the circle therefore casts votes over a cone segment rather than a full cone in Hough space. The addition of edge information is computationally more efficient and reduces the number of local maxima in Hough space, thus making the isolation of meaningful circles easier.

The ellipse needs five parameters to fully describe its position, shape, and rotation. Most frequently, the equation of an ellipse that is oriented parallel to the x-axis is found:

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} = 1 \tag{7.10}$$

Here a and b are the long and short semiaxes, respectively, and x_c and y_c the center coordinates. However, ellipses found in images may have any orientation; therefore, a rotation of the coordinate system at an angle θ must be considered. The general ellipse equation assumes either the stable polynomial form of Equation (7.11) with

the eccentricity $e = b/a$ and the coefficients given in Equation (7.12)[12] or the polar form given in Equation (7.13) with the coefficients r and ϕ defined in Equation (7.14).

$$x^2 + y^2 - U(x^2 - y^2) - 2Vxy - Rx - Sy - T = 0 \tag{7.11}$$

$$\begin{aligned} e &= \frac{b}{a} \\ U &= \frac{1 - e^2}{1 + e^2} \cos 2\theta \\ V &= \frac{1 - e^2}{1 + e^2} \sin 2\theta \\ R &= 2x_c(1 - U) - 2y_c V \\ S &= 2y_c(1 - U) - 2x_c V \\ T &= \frac{2a^2b^2}{a^2 + b^2} - \frac{x_c R}{2} - \frac{y_c S}{2} \end{aligned} \tag{7.12}$$

$$\frac{r^2 \cos(\phi - \theta)}{a^2} + \frac{r^2 \sin(\phi - \theta)}{b^2} = 1 \tag{7.13}$$

$$\begin{aligned} r^2 &= \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \tan\phi &= \frac{x - x_c}{y - y_c} \end{aligned} \tag{7.14}$$

The parameter space is now five-dimensional with the parameters x_c, y_c, a, b, and θ. This not only puts a heavy load on memory use and computational effort, but the accumulated votes will start to spread out in this high-dimensional space and become sparse. It becomes more difficult to detect meaningful local maxima in high-dimensional spaces with sparse votes. Generally, a straightforward approach as seen in the line and circle transforms is no longer possible, and more efficient algorithms are needed.

One early proposal[19] involved the detection of majority votes for the ellipse center only, followed by a search for pixel clusters in the image that satisfied the ellipse equation. With multipass techniques, the five-dimensional space was broken down into low-dimensional subspaces: for example, into one two-dimensional space for (x_c,y_c), another two-dimensional space for (a,b) and finally, a one-dimensional space for the orientation θ.[16] Breaking down the voting process into subspaces, like spreading out the votes into the five-dimensional parameter space, decreases the robustness of the algorithm.

Alternatively, it is possible to reduce the high-dimensional space by providing a one-dimensional space for each parameter. For the ellipse, five independent accumulators would receive the votes for the parameters x_c, y_c, a, b, and θ independently. Each accumulator provides one or more majority votes for its respective parameter. The main advantage of this technique, apart from the memory efficiency, is the increased robustness achieved by concentrating the votes. On the other hand, the solutions are not unique. If each accumulator provides exactly one maximum, the shape is uniquely

defined. However, if each accumulator provides multiple maxima, all shapes that are a possible combination of the maxima are shape candidates. If two ellipses exist in the image, for example, there will be two maxima in each accumulator. By combining these parameters, 32 ellipses can be identified as possible solutions. A subsequent validation step can identify the true shapes out of the candidate pool.

A more advanced approach was presented by Yip et al.[22] Under inclusion of the gradient direction, point pairs on opposite sides of the ellipse are sought. These point pairs have the same tangent direction and are connected by a line through the center of the ellipse. Provided that they are members of the ellipse outline, two of these point pairs are sufficient to compute all five parameters of the ellipse candidate. The accumulation can therefore be reduced to accumulating the x and y coordinates of the two point pairs, and ellipse candidates can be extracted from the histogram of the two-dimensional accumulator array. Similar approaches of reducing the dimensionality by using edge information and well-designed transformations were introduced by other groups.[1,18,20] These advanced algorithms for extracting ellipse parameters are valid for circles as well, since the circle is a special case of the ellipse with $a = b = r$ in Equation (7.10).

7.3. GENERALIZED HOUGH TRANSFORM

The idea of the Hough transform, the accumulation of votes, can be extended to detect arbitrary (i.e., nonanalytical) shapes.[2] For this purpose, the shape template is decomposed into straight edge sections. In the extreme case, each edge pixel may represent one edge section. An arbitrary reference point is chosen. A frequent choice is the centroid of the shape. Next, a vector from the center of each edge section to the reference point is computed and its length and its angle to the edge direction are stored in a table. This table is generally referred to as the *R-table*.

Generation of an R-table and the voting process are illustrated in Figure 7.10. For a simple sample shape such as the polygon in Figure 7.10, the reference point can be placed near the polygon's center. Each edge is connected to the reference point, resulting in a R-table that contains the vectors v1 through v5.

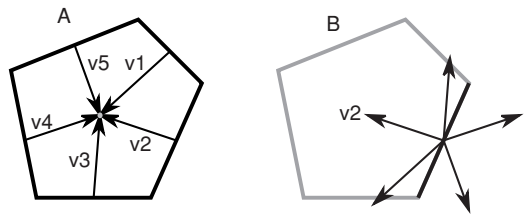


FIGURE 7.10 Generation of the R-table (A) and voting process for one edge of the shape (B). To generate the R-table, each edge is connected with the reference point, and each of the resulting vectors v1 through v5 is entered in the table. For the voting process (B), each edge is displaced successively by all vectors and a vote for the endpoint coordinates of each vector is cast. At the end of vector v2, votes from the other edges accumulate.

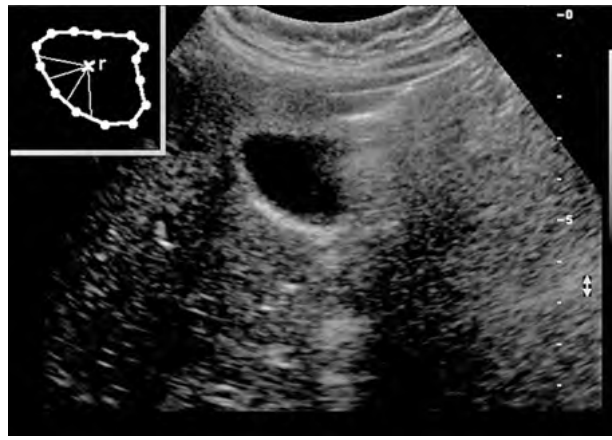


FIGURE 7.11 Ultrasound image of a gallbladder. The inset shows the approximate outline, composed of small line segments. A reference point r is defined, and the lines (four of 14 lines shown) connecting each edge segment with the reference point are entered in the R-table.

When a shape is analyzed, each edge is displaced by all vectors in the R-table in sequence. In the example in Figure 7.10B, this process yields four points for each edge. At each of these points, the accumulator array is incremented. If the shape indeed matches the template, a majority vote (for the polygon in Figure 7.10, five votes) is found in the center, and the location of the square is known.

The generalized Hough transform is illustrated further in a more complex example in Figure 7.11. The example shows an ultrasound image of a gallbladder together with a line segment approximation of its shape. An arbitrary reference point r is chosen, and each line segment is connected to point r. In this example, the shape is approximated by 14 line segments, and 14 vectors would be entered into the R-table.

If a shape exists in the image that matches the template, its reference point would be reached from any edge point by displacing that point by one of the vectors in the R-table. This notion leads to the algorithm to accumulate the votes. A two-dimensional array (the Hough transform image) needs to be provided. Furthermore, an edge image needs to be provided where each candidate edge is identified: In the simplest case, each edge is represented by a pixel or a vertex with known x and y coordinates. Consistent with the notation in previous sections, these would correspond to the points (x_k,y_k). Now each edge point or vertex is displaced successively by all of the vectors in the template's R-table. The coordinate of the endpoint of the displacement is incremented by one; thus, votes are accumulated. If a shape matching the template exists in the image, a majority vote will be found at the coordinate of the reference point in the Hough transform image. The shape can now be reconstructed by using the R-table again: Each vector in the R-table is now inverted and points away from the reference point that was found. At the end of each vector is one edge segment of the shape template.

From Figure 7.12 it becomes clear that an exact template match is hardly ever found in medical images. In such a case, the votes do not accumulate in one pixel of the accumulator array but rather in a blurred region near the ideal reference point,

FIGURE 7.12 Ultrasound image of Figure 7.11 after segmentation of the gallbladder with the outline of the segmented shape superimposed over the image. The segmentation makes use of the nonechogenic property of the gallfluid. Region growing from an arbitrary point inside the gallbladder area with a low threshold fills the gallbladder area, and the outline is obtained by applying the Sobel operator.

and strategies must be devised to identify the majority vote. The simplest strategy is to introduce binning (i.e., the combination of adjoining pixels). The votes would now fall into a single bin rather than be spread out over neighboring pixels, and the bin containing a local maximum can be identified. However, binning implies a trade-off between detection reliability (needs large bins) and exact localization (needs small bins). Furthermore, selecting bins that are too large may introduce incorrect detection in any of these ways: detection of false-positive maxima, detection of a local vote maximum for shapes that only approximate the template, or the joining of the maxima from closely spaced matching shapes. Clustering techniques, such as *k-means clustering*, may provide a suitable alternative to assign spread-out votes to one cluster. The centroid of a majority cluster would then be used to identify presence and location of a template-matching shape.

Extensions of the algorithm make it possible to account for shapes that are rotated and scaled with respect to the template. First the R-table is extended by the template's edge direction (angle θ_1) and the relative orientation of the displacement vector (the angle between the edge direction and the displacement vector $\Delta\theta$). Second, in the shape under examination, the absolute angle θ_2 of each edge point is determined. For each edge point and for each displacement vector in the R-table, the vector needs to be rotated by $\theta_2 - \theta_1$ prior to displacing the edge for the vote. Rotating the vector aligns it with the overall rotation of the shape relative to the orientation of the template. To obtain the overall rotation angle of the shape under examination (provided that a majority vote indicated its existence), each individual rotation angle can be recorded and averaged. In the generalized Hough transform, the edge rotation range obtained by the Sobel or compass operators is not sufficiently accurate to account for shape rotation. Rather, it is necessary to follow the contour of the edge detected and determine the edge direction from neighboring points.

To account for different sizes of the shape (a different scale), the parameter space can be extended to include a scale dimension s. The accumulator array is now three-dimensional (x,y,s), and the voting process is repeated with different values of s, with each displacement vector being multiplied by s before displacing the edge. Along the s-axis, the votes converge to the appropriate scale of the shape under examination.

7.4. RANDOMIZED HOUGH TRANSFORM

The inherent disadvantages of the Hough transform, that is, large memory space use and the challenge of finding meaningful local maxima in Hough space, led to a different approach to curve parametrization: the randomized Hough transform.[21] Strictly, the randomized Hough transform is not a transform since the original space cannot be fully reconstructed, due to the stochastic nature of the randomized Hough transform. The randomized Hough transform has in common with the standard Hough transform the principle of using a parameter space and the notion of accumulating votes. The difference is the selection of groups of pixels in a random fashion.

Let us introduce the randomized Hough transform by using the line for simplicity. The starting point is an edge image that has been thresholded. Therefore, a pixel is either set (edge) or not set (background). Assuming the line described in Equation (7.1) by its parameters m and n, any two edge pixels (x_1,y_1) and (x_2,y_2) lie on a line that can be calculated by solving the linear equation system for m and n:

$$\begin{aligned} y_1 &= mx_1 + n \\ y_2 &= mx_2 + n \end{aligned} \tag{7.15}$$

A similar approach can be used to obtain ρ and θ in the polar form of Equation (7.4). For the randomized Hough transform, pixel pairs with (x_1,y_1) $\neq$ (x_2,y_2) are selected randomly and the corresponding point in parameter space [either (m,n) or (ρ,θ)] is incremented by 1, thus accomplishing the accumulation step. Any pixel pair must be selected only once, but individual pixels may be part of more than one pair. If there is a dominant line in the image, the probability of incrementing the corresponding accumulator in Hough space is higher than the accumulators for random lines defined by arbitrary pixel pairs. Through the detection of local maxima, dominant lines can be isolated and transformed into regular space using the conventional inverse Hough transform.

The straightforward approach of solving an equation system with n parameters using n randomly selected points is only possible in special cases, such as any shape that is described by a linear equation in parameter space:

$$z_0 + a_1 z_1 + a_2 z_2 + \cdots + a_n z_n = 0 \tag{7.16}$$

where a_1 through a_n are the parameters (coordinates in Hough space) and z_0 through z_n are functions of x and y. For n randomly chosen edge points (x_k,y_k), a linear equation system based on Equation (7.16) yields the parameters a_1 through a_n. A notable exception is the circle, which can be solved from equation (7.7) in two steps.[21] Three points that lie on the circle are needed to solve Equation (7.7) for the unknown parameters x_c,y_c, and r. The idea behind the first step is to compute the

midperpendicular [Equation (7.17)] of the line segment between two pairs of points. The midperpendiculars meet in the center of the circle. For example, the point pairs $i = 1, j = 2$ and $i = 2, j = 3$ could be chosen, leading to

$$y_c - \frac{y_i + y_j}{2} = \frac{y_i - y_j}{x_i - x_j}\left(x_c - \frac{x_i + x_j}{2}\right) \tag{7.17}$$

which can be solved for x_c and y_c. In the second step, any of the three points can be used to obtain r from Equation (7.7).

One additional consideration is the stop condition for the selection of random point groups. It is possible to limit the number of iterations combined with a limit on the highest number of votes in a cell. The random selection of pixel groups and the accumulation of votes in parameter space can be stopped, for example, if either 1000 or more point groups have been processed or 50 or more votes have accumulated in one single cell.

The most important advantage of the randomized Hough transform over the conventional Hough transform is the option to organize votes in a dynamic data structure rather than using the n-dimensional Hough space. By further following the example of the line transform, let us assume that the parameters (m,n) are stored as a two-dimensional searchable list, which at the start of the process is empty. Whenever a pixel pair yields a parameter pair (m,n), the list is searched if any element k exists for which

$$\sqrt{(m - m_k)^2 + (n - n_k)^2} \leq \delta \tag{7.18}$$

If so, the vote counter of the kth element is increased by 1. Otherwise, a new element with the new parameter pair (m,n) is created with a vote number of 1. The new distance threshold, δ, can be used to balance resolution and memory requirement. If the pixels in the edge image are sparse, a large δ allows us to accumulate a larger number of votes with an effect similar to binning in the conventional Hough transform.

Algorithm 7.2 suggests how to implement the randomized Hough transform for the circle. Analogous to the conventional Hough transform, Algorithm 7.2 makes use of a three-dimensional parameter space for voting instead of using a linked list. List management involves some complexity, and the principle is easier to demonstrate by using regular parameter space. For the conversion to linked lists, freely available libraries advertise themselves.

For nonlinear shapes, such as the ellipse, the approach needs to be adapted. No straightforward linear scheme exists to extract the five ellipse parameters from five randomly selected points. In this case, a statistical approach may prove beneficial. By sampling more than five points, the ellipse parameters can be extracted using the least-squares method.[6] In this case, and in the case of any other higher-dimensional Hough space, the superiority of linked lists to store the accumulated votes becomes obvious. In addition, image preprocessing becomes highly important. Whereas the iterative Hough transform is robust against incomplete shapes, the presence of noise may confound the maxima found in the Hough transform. Since the Hough transform implies the application of an edge detector step, noise suppression is a key step in image preprocessing.

```
set rmax=50;                    // Choose a suitable maximum radius
set dcnt=0;                     // counter for the number of random point triplets processed
set vmax=0;                     // Reset the highest vote count memory
allocate IH(xh, yh, rmax);      // Allocate output image

for (y=0 while y<yhsize increment y=y+1) do
   for (x=0 while x<xhsize increment x=x+1) do
      for (r=0 while r<rmax increment r=r+1) do
         IH(x,y,r) = 0;                 // Clear the accumulator to zero
      endfor;
   endfor;
endfor;

while ((dcnt<5000) and (vmax<50)) do
   repeat                               // Make sure we don't pick the same point twice
      repeat
         x1=random(1)*xm; y1=random(1)*ym;
      until IM(x1,y1)>0;                          // Select a nonzero pixel
      repeat
         x2=random(1)*xm; y2=random(1)*ym;
      until IM(x2,y2)>0;                          // Select another nonzero pixel
      repeat
         x3=random(1)*xm; y3=random(1)*ym;
      until IM(x3,y3)>0;                          // Select the third nonzero pixel
   until (  ((x1!=x2) or (y1!=y2)) and
            ((x1!=x3) or (y1!=y3)) and
            ((x2!=x3) or (y2!=y3)) );

   a1=(y2-y1)/2; b1=(y2-y1)/(x2-x1); c1=(x2-x1)/2;      // Start computation
                                                        // of xc,yc
   a2=(y3-y1)/2; b2=(y3-y1)/(x3-x1); c2=(x3-x1)/2;
   xc = (a1-a2+b2*c2-b1*c1)/(b2-b1);                    // center coordinate xc
   yc = (b1*b2*(c2-c1)+a1*b2-a2*b1)/(b2-b1);            // center coordinate yc
   r = sqrt( sqr(x1-xc) + sqr(y1-yc));                  // radius

   if (r<rmax) then IH(xc,yc,r)=IH(xc,yc,r)+1;          // Accumulate vote
   if (vmax < IH(xc,yc,r)) then vmax = IH(xc,yc,r);     // Maximum votes
                                                        // accumulated?
endwhile;
```

Algorithm 7.2 Randomized Hough transform for circles. The input image `IM` (size: `xm, ym`) is assumed to be thresholded, and each nonzero pixel represents an edge pixel. This algorithm represents the conventional form with a three-dimensional parameter space for the accumulation of votes. The output image, IH, contains the accumulated votes. To make the code simpler, the condition that any group of 3 pixels may be selected only once is not considered.

A new technique with superior immunity against image noise has been presented by Lu et al.[14] The idea behind this technique, termed *iterative randomized Hough transform* (IRHT), is repeatedly to perform shape detection with the randomized Hough transform, followed by a reduction of the region of interest to more narrowly enclose the most probable location of the shape. With each iteration, sections of the image (containing only noise) are excluded, and the remaining votes define the actual shape better with each vote. A demonstration of the principle can be seen in Figure 7.13, and the algorithm is described in Algorithm 7.3.

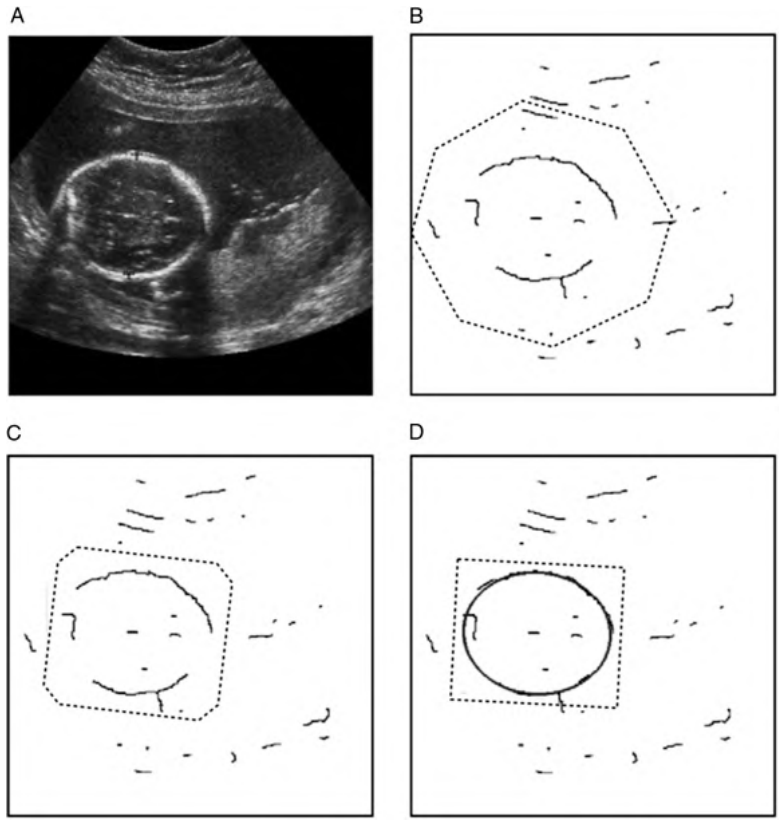


FIGURE 7.13 Application of the IRHT in the quantitative determination of the size of a fetal head. The starting point is an ultrasound image (A), which shows the strongly echogenic parts of the skull. After segmentation and skeletonization (B), an initial octagonal region of interest is placed around the likely candidate (dashed line). Iterative application of the randomized Hough transform allows to reduce the region of interest out of which votes are taken (C), and less noise is included from each iteration to the next. The algorithm converges (D) when the region of interest changes only minimally between iterations. In general, the region of interest aligns with the main axes of the ellipse and is only slightly larger than the ellipse. In part D the parametrized ellipse is shown superimposed over the edge image, and it can be seen that the broken shape (B) is completed.

```
Set ROI to image size
set k=200                                  // number of sets of 5 pixels in RHT
set kf=2500                                // number of polls before failure
set invalid=0; converged=0;                // exit conditions

Allocate memory for five independent accumulators

repeat
   i=0; j=0;
   repeat
      randomly select 5 segmented pixels from inside ROI
      i=i+1
      find ellipse parameters from these 5 pixels: successful?
         successful when:
            1) Solution for Equation 7.11 exists
            2) Ellipse lies within image area
            3) Ellipse center lies within ROI
      if (successful) then
         increase the scores in the accumulators
         j=j+1; i=0;
      else
         if (i>kf) then
            invalid=1;                     // Give up after kf tries
         endif
      endif
   until (invalid or j>k)                  // Iterate until k samples processed

   if (not invalid) then
      find maximum value in each accumulator
      determine ellipse from accumulator maxima
      new_ROI = (slightly larger than bounding box of ellipse)
      if (new_ROI == ROI) then converged=1;
      ROI = new_ROI;
   endif

until (converged or invalid)
```

Algorithm 7.3 Pseudocode for the computation of the iterative randomized Hough transform for ellipses. The input image IM (size: xm, ym) is assumed to be thresholded, and each nonzero pixel represents an edge pixel. This code assumes that only one ellipse exists and takes the single maxima of separated accumulators to determine the ellipse parameters. Due to the complexity of the algorithm and possible variations in the ellipse representation, the pseudocode is kept more general.

Algorithm 7.3 assumes that only one ellipse exists in the image and that for this special case it is possible to separate the Hough space into five individual one-dimensional accumulators, one for each parameter of the ellipse. Maximum detection takes place for each accumulator separately, and the ellipse is determined from the location of these five maxima. The algorithm is strongly dependent on the ellipse equation used. The randomized Hough transform can generally be solved only if the curve equation is linear with respect to the parameters. Therefore, the ellipse representation in Equation (7.11) is ideally suited. A variation of this algorithm would involve the random selection of more than five pixels and the determination of the ellipse parameters by using a least-squares approach.

Multiple ellipses can be detected with the IRHT as well. If multiple ellipses are expected in an image, the IRHT can be applied repeatedly. After each detection of one ellipse, the set of pixels that belong to the detected ellipse is subtracted form the image and the IRHT repeated. This iterative process is stopped after no valid ellipses have been found. Users implementing this algorithm should be aware that an image with pure noise generally identifies an apparent ellipse, centered with respect to the image and angled 45° or 135°.[14]

7.5. BIOMEDICAL EXAMPLES

Although development of the Hough transform is driven primarily by computer vision applications, the Hough transform is frequently used in biomedical image analysis. Medical images pose additional challenges to image analysis, primarily noise and natural shape variation. For this reason, most medical applications of the Hough transform include additional image processing steps or variations of the established Hough transform algorithms. For example, the Hough line transform has been used in the detection of the Cobb angle of scoliosis in spinal radiographs.[24] In digitized radiographs, the most severely tilted vertebrae were identified and outlined manually (Figure 7.14). Images were denoised by using anisotropic diffusion, a step that emphasized the actual vertebral edges in the subsequent edge detection process. The edges of vertebrae are not exact straight lines, and the conventional Hough transform would detect multiple lines at each edge. In this study, a variation of the Hough transform termed the *fuzzy Hough transform*[8] was used, where closely spaced lines contributed to the accumulator more or less depending on the distance from the ideal detected line. In this implementation, the accumulator was convolved along the ρ-axis with a Gaussian blurring function. With this step, closely spaced local maxima were eliminated and a single line emerged. Furthermore, it was necessary to apply a priori knowledge of vertebral geometry: approximately parallel endplates and approximately perpendicular sides, which led to individual accumulator maxima. These maxima had to follow certain distance constraints in Hough space before they were used to compute the exact tilt angle of the vertebra.

In a different study, ultrasound imaging was used to determine the orientation of muscle fibers.[25] The basis for the automated detection of fiber orientation was once again the line Hough transform. Similar to the scoliosis study, the conventional

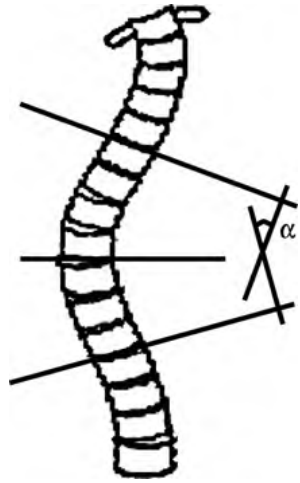


FIGURE 7.14 Definition of Cobb's angle. From a central vertebra that is not tilted, the vertebrae with the largest tilt above and below are found. Cobb's angle α is the angle at which these two vertebrae are tilted toward each other, usually found with the help of two perpendicular auxiliary lines.

Hough transform detected multiple closely spaced lines. The key improvement in this study over the conventional Hough transform was a re-voting mechanism. Iteratively, the most dominant line was determined in Hough space and returned as a valid line. Subsequently, all pixels in the edge image along this line were removed and the Hough transform repeated. By removing pixels along the dominant line, the votes for closely spaced lines were also removed. This algorithm is demonstrated in Figure 7.15.

Ultrasound images are a particular challenge due to the inherent noise. The routine measurement of fetal head size, instrumental in determining fetal growth and maturity, relies almost exclusively on ultrasound imaging. As shown in one example in Figure 7.13, the skull outline is an approximate ellipse with missing segments. For this reason, the ellipse Hough transform appears to be a very attractive approach. In three published studies[13,14,23] images were prepared by edge detection followed by thresholding and skeletonization. Parametrization of the ellipse was then performed using the IRHT.

Another good example where the Hough transform provides an accurate parametrization of the shape is the pupil of the eye. Applications include tracking of the eye movement and quantitative measurement of the pupillary reflex. Unlike ultrasound imaging, images of the eye are obtained with either a regular camera or a video camera. Consequently, the images are characterized by a lower noise component and higher resolution and contrast. Furthermore, segmentation is a relatively straightforward process, because the dark or even black interior of the pupil can be segmented with threshold-based methods. On the other hand, any illumination may introduce areas of bright specular reflection. These areas may lead to incomplete

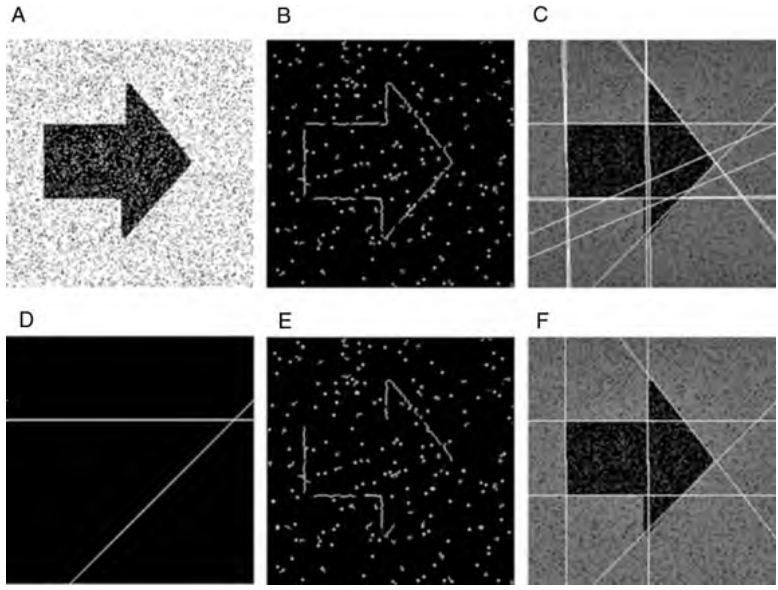


FIGURE 7.15 Iterative Hough transform with re-voting mechanism. The example image (A) is a shape with shot noise superimposed. (B) shows the gradient image after thresholding and skeletonization. The straight lines are distorted, and multiple closely spaced local maxima appear in the Hough transform, evidenced by multiple, almost-identical lines in the inverse Hough transform (C). For the iterative Hough transform with re-voting, one dominant line (or as shown in image D, two lines with identically high votes) is extracted. This line is subtracted from the edge image (B), leading to the new edge image (E). If image E is subjected to the Hough transform, the local maxima that led to the lines in image D no longer appear, and a different line will be the dominant line of the next iteration. As can be seen in image F, the iterative Hough transform with re-voting becomes more robust.

circles, indentations, or other shape distortions in the segmented pupil area. The IRHT has been shown to reconstruct the pupil ellipse accurately in the presence of areas with specular reflection.[13] Noninvasive shape determination of the nucleus of the human lens was performed in images acquired with a Scheimpflug camera, a camera where the lens plane is not parallel to the detector plane. Such a camera is capable of acquiring a focused image of the entire corneal area. The outline of the lens nucleus was approximated by connected parabolic curve segments. This method made it possible to quantitatively analyze the nucleus volume, cross-sectional area, curvature, and thickness as a response to accommodation stimuli.[9]

An other area in which the Hough transform plays an important role is in the determination of the elasticity of the carotid artery in ultrasound motion images.[7] In this study, a combination of line and circle Hough transforms made it possible to extract the diameter of the carotid artery in a sequence of images. A virtual robot for automated endoscopy of the cochlear spiral in micro-CT images was presented.[5] A complex sensing and steering system guides the software robot through the

three-dimensional cochlear image. A cylinder Hough transform is used to estimate the cochlear radius at the robot's position. Finally, the accumulation approach that is the basis of the Hough transform has been used in the registration of images.[3,11]

REFERENCES

1. Aguado AS, Nixon MS. A New Hough Transform Mapping for Ellipse Detection. Technical Report. Department of Electronics and Computer Science, University of Southampton, Southampton, UK, 1995.
2. Ballard DH. Generalizing the Hough transform to detect arbitrary shapes. Pattern Recogn 1981; 13(2):111–122.
3. Chen QS, Defrise M, Deconinck F. Three-dimensional multimodality medical image registration using a parameter accumulation approach. Med Phys 1996; 23(6):877–885.
4. Duda RO, Hart PE. Use of the Hough transform to detect lines and curves in pictures. Commun Assoc Comput Mach 1972; 15:1–15.
5. Ferrarini L, Verbist BM, Olofsen H, Vanpoucke F, Frijns JH, Reiber JH, Admiraal-Behloul F. Autonomous virtual mobile robot for three-dimensional medical image exploration: application to micro-CT cochlear images. Artif Intell Med 2008; 43(1):1–15.
6. Fitzgibbon AW, Pilu M, Fisher RB. Direct least-squares fitting of ellipses. IEEE Trans Pattern Anal Mach Intell 1999; 21:476–480.
7. Golemati S, Stoitsis J, Sifakis EG, Balkizas T, Nikita KS. Using the Hough transform to segment ultrasound images of longitudinal and transverse sections of the carotid artery. Ultrasound Med Biol 2007; 33(12):1918–1932.
8. Han JH, Kóczy LT, Poston T. Fuzzy Hough transform. Pattern Recogn Lett 1994; 15: 649–658.
9. Hermans E, Dubbelman M, van der HR, Heethaar R. The shape of the human lens nucleus with accommodation. J Vis 2007; 7(10):16–10.
10. Hough PVC. Method and means for recognizing complex patterns. US patent 3,069,654. 1962.
11. Jiang CF, Lu TC, Sun SP. Interactive image registration tool for positioning verification in head and neck radiotherapy. Comput Biol Med 2008; 38(1):90–100.
12. Leavers VF. The dynamic generalized Hough transform: its relationship to the probabilistic Hough transforms and an application to the concurrent detection of circles and ellipses. CVGIP: Image Underst 1992; 56:381–398.
13. Lu W, Tan J, Floyd R. Automated fetal head detection and measurement in ultrasound images by iterative randomized Hough transform. Ultrasound Med Biol 2005; 31(7):929–936.
14. Lu W, Tan J, Zhang K, Lei B. Computerized mouse pupil size measurement for pupillary light reflex analysis. Comput Methods Programs Biomed 2008; 90(3):202–209.
15. Merlin PM, Faber DJ. A parallel mechanism for detecting curves in pictures. IEEE Trans Comput 1975; 24:96–98.
16. Muammar H, Nixon M. Approaches to extending the Hough transform. Proc ICASSP 1989; 89:1556–1559.
17. O'Gorman F, Clowes MB. Finding picture edges through collinearity of feature points. IEEE Trans Comput 1976; 25(4):449–456.

18. Pao D, Li HF, Jayakumar R. A decomposable parameter space for the detection of ellipses. Pattern Recogn Lett 14, 951–958. 1993.
19. Tsuji S, Matsumoto F. Detection of ellipses by a modified hough transformation. IEEE Trans Comput 1978; 27(8):777–781.
20. Wu W, Wang MJ. Elliptical object detection by using its geometrical properties. Pattern Recogn 1993; 26(10):1499–1509.
21. Xu L, Oja E, Kultanen P. A new curve detection method: randomized Hough transform (RHT). Pattern Recogn Lett 1990; 11:331–338.
22. Yip RKK, Tam PKS, Leung DNK. Modification of Hough transform for circles and ellipses detection using a 2-dimensional array. Pattern Recogn 1992; 25(9):1007–1022.
23. Yu J, Wang Y, Chen P, Shen Y. Fetal abdominal contour extraction and measurement in ultrasound images. Ultrasound Med Biol 2008; 34(2):169–182.
24. Zhang J, Lou E, Le LH, Hill DL, Raso JV, Wang Y. Automatic Cobb measurement of scoliosis based on fuzzy hough transform with vertebral shape prior. J Digit Imaging 2008 (Epub).
25. Zhou Y, Zheng YP. Estimation of muscle fiber orientation in ultrasound images using revoting Hough transform (RVHT). Ultrasound Med Biol 2008; 34:1474–1481.

8

TEXTURE ANALYSIS

Each image feature contains three major elements of information. For each feature, this is the average image value (e.g., intensity, exposure, density), the shape, and any *systematic local variation* of the image values. This local variation is called *texture*. Some examples that can be found in the UIUC texture database[1,26] are shown in Figure 8.1. The textures differ in character. From top to bottom, increased regularity can be observed. The top two textures have a high degree of randomness and some self-similar properties. The middle two textures are more homogeneous in their feature size, but with irregular distribution, whereas the bottom two of the textures show spatial regularities. In medical images, texture carries important information as well. While some of the texture is caused by image noise, image texture often carries information from the tissue being imaged. An example is shown in Figure 8.2: an abdominal ultrasound image showing the inferior vena cava and a section of the liver. For the human eye, the texture difference between the liver and the surrounding tissue is easily detected despite the dominant ultrasound noise. Many textures are composed of a repeated pattern of texture elements (often referred to as *texels*). Such a texture (e.g., the knit texture in Figure 8.1) would be described completely by the texel's size and orientation. In medical images, periodically arranged texels are the exception. Medical images would best be described as a random arrangement of clusters of different intensity and size.

Texture analysis can serve three primary purposes. First, texture analysis may help classify pixels or image features and therefore assign them to a specific region or object. Second, the classification of regions or pixels by their texture

Advanced Biomedical Image Analysis, By Mark A. Haidekker

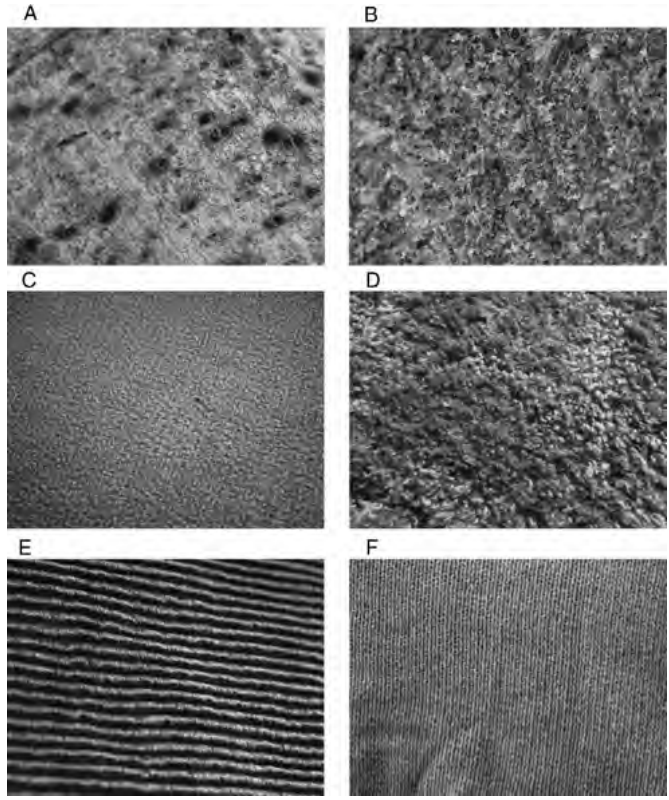


FIGURE 8.1 Some common texture samples.[1,26] From top left to bottom right, the images show tree bark (A), polished granite (B), glass (C), carpet (D), corduroy (E), and knit (F). From top to bottom, the textures become more regular. (From refs. 1 and 26.)

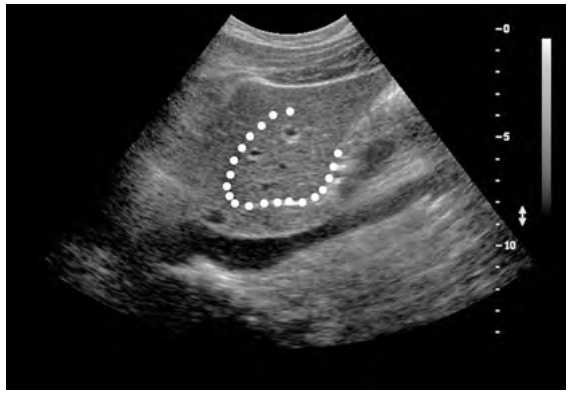


FIGURE 8.2 Ultrasound image of the inferior vena cava (dark region in the lower third of the image). A section of the liver, enclosed by white dots for easy identification, shows a texture different from that of the surrounding tissue.

can aid segmentation. Regions with approximately identical or overlapping intensity values, for example, can potentially be separated by the texture properties. Two-dimensional thresholding with intensity and texture as orthogonal criteria is even more powerful for segmentation and pixel classification. Third, texture analysis may help differentiate between normal and diseased states by their texture in medical images.

Texture analysis is a multistep process. The first step is usually an enhancement step to emphasize the texture and reduce the influence of unwanted processes. The enhancement step includes noise reduction, background removal, and possibly, histogram equalization. The presence of noise may create a pseudotexture that has the potential to dominate and confound texture classification. Background inhomogeneities may affect any intensity-based operations, such as thresholding, and they may also confound some texture metrics. Some texture metrics are more robust against background inhomogeneities. These include first-order statistics, the co-occurrence matrix, Law's texture energy, and the autocorrelation function. Conversely, the run-length method is sensitive to background inhomogeneities, and a highpass filtering step or local histogram equalization is recommended. In the next step, regions of interest need to be delineated over which quantitative texture features will be computed unless the computation of texture features occurs in a local neighborhood on a pixel-by-pixel basis. In the next step the actual classification takes place. Each pixel or region gets assigned to one or several quantitative texture feature metrics. To provide one example of a pixel metric, the local variance in a 7×7 neighborhood can be computed for each pixel. A pixel embedded in a rough texture would then have a higher local variance value assigned to it than would a pixel embedded in a smooth region. A second example would be a local anisotropy metric, where a pixel embedded in a directional texture would get a value different from that of a pixel embedded in a isotropic texture. Depending on the goal, it may be necessary to collect multiple texture feature metrics to fully classify a pixel or region. The final step in this process is evaluation of the information collected in the classification step. Usually, this final step involves determining a category into which the pixel or region belongs. Within a defined region, its texture can be categorized as healthy or diseased. Alternatively, the classification information may be used to segment regions, for example, to separate the liver region in Figure 8.2 from the surrounding tissue. The complexity of this last step can vary greatly from value-based thresholding or multidimensional thresholding to Bayesian classification, clustering, and even artificial-intelligence-based methods such as artificial neural networks. In this chapter, key methods to extract texture features from textured image regions, that is, to assign different numerical values to different types of texture, are presented.

8.1. STATISTICAL TEXTURE CLASSIFICATION

The intensity distribution of pixels in an image or a region of an image is characterized in the intensity histogram. The number of pixels is plotted as a function of their intensity, providing a discrete function $n(I)$. For the purpose of texture classification,

the histogram needs to be normalized so that

$$\frac{1}{N}\sum_{I=I_{\min}}^{I_{\max}} n(I) = 1 \tag{8.1}$$

with the factor N being the total number of pixels. The normalized values $P(I) = n(I)/N$ are the probabilities of a pixel intensity or intensity range. On the basis of this value, we define the moments of the histogram:

$$\mu_k = \sum_{I=I_{\min}}^{I_{\max}} I^k P(I) \tag{8.2}$$

The first moment, μ_1, is the image intensity average. The higher moments can be expressed as the *k-th moment about the mean* by subtracting the average image value μ_1. The moments about the mean are referred to as *central moments of the histogram*,

$$M_k = \frac{1}{S_k}\sum_{I=I_{\min}}^{I_{\max}} (I - \mu_1)^k P(I) \tag{8.3}$$

where the normalization factor s_k is 1 for $k = 2$, and σ^k for $k = 3$ and $k = 4$, with the standard deviation $\sigma = \sqrt{M_2}$. Under this definition, the first central moment M_1 is always zero. The central moments for $k = 2$, 3, and 4 are known as variance, skew, and kurtosis. The second moment, the *variance*, is related to the intensity spread. A high variation in image values is associated with a high variance, and vice versa. Therefore, the variance contains information about the image contrast. The third moment, the *skew*, is related to the asymmetry of the intensity distribution. A perfect Gaussian distribution has a skew of zero. A positive skew indicates a longer tail to the right of the histogram, and a negative skew indicates a longer tail to the left of the histogram. In other words, when the mean value lies to the right of the mode value (the *mode value* is the intensity value with the highest probability), the histogram has a positive skew, and vice versa. In fact, skew is closely related to the difference between the mean and the mode. The fourth moment, the *kurtosis*, provides information about the accumulation of the values around the mean value. A histogram with positive kurtosis has a higher probability of values being distributed near the mean value than does a normally distributed histogram, and little of the variance is caused by infrequent extreme deviations. A negative kurtosis indicates a histogram that has more pronounced tails than does a normally distributed histogram, and indicates a higher probability of extreme intensity values than that for a normally distributed histogram.

Table 8.1 lists the statistical moments of the images shown in Figure 8.1. All images except the corduroy texture show a mean value around 128, which is the mean gray value in an evenly distributed 8-bit image. Glass and granite have a fairly narrow distribution (small variance) that is immediately evident in the glass image. The granite image exhibits mostly average gray areas with a few dark and bright spots. The high intensity variance in the corduroy image is caused by a clear bimodal

TABLE 8.1 Statistical Moments for the Images in Figure 8.1

Texture	μ_1 (Mean)	M_2 (Variance)	M_3 (Skew)	M_4 (Kurtosis)
Bark	146	1979	−0.489	0.114
Granite	119	1669	−0.03	−0.26
Glass	120	1949	0.166	−0.11
Carpet	110	2254	0.263	−0.413
Corduroy	84	3551	0.585	−0.855
Knit	130	2328	−0.163	−0.508

distribution stemming from the ridges and valleys. This distribution is also reflected in the large positive skew (long tail caused by the second mode drawing the mean value to the right of the first mode) and the large negative kurtosis, which indicates a distribution dominated by extreme values. The largest negative kurtosis is found in the corduroy texture, as a consequence of the two modes.

Two more histogram-related metrics can be defined, the energy E and the entropy H. The energy [Equation (8.4)] assumes its maximum value close to 1 only if the image has a very narrow histogram, dominated by a single value. Broader intensity variations [and therefore lower individual $P(I)$ values] cause the energy to decrease. The entropy [Equation (8.5)] is a measure of information content and is in a qualitative manner inversely related to the energy. A predominantly random distribution has a high entropy. Highly correlated or uniform distributions have a low entropy.

$$E = \sum_{I=I_{\min}}^{I_{\max}} P^2(I) \tag{8.4}$$

$$H = -\sum_{I=I_{\min}}^{I_{\max}} P(I)\log_2 P(I) \tag{8.5}$$

Table 8.2 lists the energy and entropy values for the textures in Figure 8.1. The overall variation of the values between textures is low, with the exception of the glass texture, which has a high energy and low entropy. This indicates a fairly uniform distribution of the intensity values in the glass texture.

TABLE 8.2 Statistical Energy and Entropy Values for the Images in Figure 8.1

Texture	Energy	Entropy
Bark	0.019	6.01
Granite	0.018	5.99
Glass	0.029	5.54
Carpet	0.016	6.15
Corduroy	0.018	6.13
Knit	0.017	6.14

It is possible to compute the statistical description values locally, that is, in a small moving window around the central pixel. In this way an image can be filtered to represent, for example, the local variance. Application of a local mean filter corresponds to a convolution-based lowpass (blurring) filter. Most notably, the local variance can be computed in a square window of side length $2l + 1$ using

$$\sigma^2(x,y) = \frac{1}{N}\left\{\sum_{i=x-l}^{x+l}\sum_{j=y-l}^{y+l} I^2(i,j) - \frac{1}{N}\left[\sum_{i=x-l}^{x+l}\sum_{j=y-l}^{y+l} I(i,j)\right]^2\right\} \tag{8.6}$$

where the $I(i,j)$ are the image intensity values in a window centered on x,y and N is the number of pixels in the local window, $N = (2l + 1)$.[2] The effect of the local variance filter can be seen in Figure 8.3, where part A shows four square patches of Gaussian noise inside a uniformly filled area. The local image mean is 127, and the noise standard deviations increase from left to right, top to bottom ($\sigma = 8$, 16, 32, and 48, respectively). The 7×7 local variance filter reveals the standard deviation, and the noisy areas in Figure 8.3B are considerably smoothed. The noisy areas have mostly nonoverlapping values in the filtered image and could be segmented with dual thresholds or hysteresis thresholding.

A local variance filter is applicable only when the texture elements are smaller than the local window size. In the example of Figure 8.3, the noise affects the image on the pixel level. In Figure 8.1 and in the ultrasound image (Figure 8.2), the texture elements are relatively large, which has a negative impact on the local variance filter and requires the use of a larger window. Algorithm 8.1 computes the local variance of a gray-scale image. If the expression (sxx-sx*sx/cnt)/cnt were replaced by sqrt((sxx-sx*sx/cnt)/cnt), the resulting image would contain the local standard deviation, and if it were replaced by cnt*(sxx/(sx*sx)-1), the resulting image

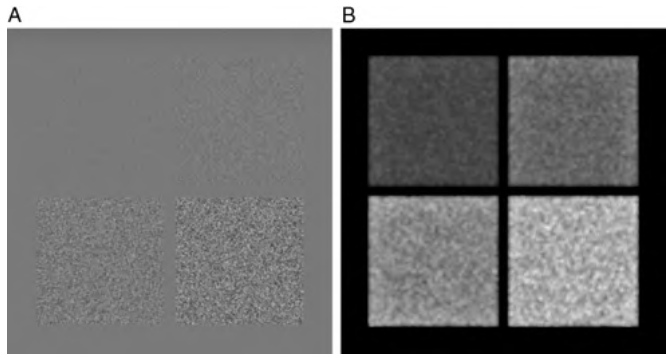


FIGURE 8.3 Application of a local variance filter. Image A contains four square areas of Gaussian noise. The uniform image value is 127, as is the mean value of the noisy areas. The standard deviation is, from top left to bottom right, 8, 16, 32, and 48. The intensity values of the filtered image (B) reflect the standard deviation. In addition, image B is somewhat smoothed.

```
set L=3;                                  // Choose a suitable neighborhood size
allocate IMV(xmax,ymax);                  // Allocate output image


for (y=0 while y<ym increment y=y+1) do    // double loop over the image
   for (x=0 while x<xm increment x=x+1) do

      sx=0; sxx=0; cnt=0;                                  // Clear accumulators
      for (j=y-L while j<=y+L increment j=j+1) do          // double loop over
         for (i=x-L while i<x+L increment i=i+1) do        // the local neighborhood
            if ((i>=0) and (j>=0) and (i<xm) and (j<ym)) // Maintain img edges
               sx = sx+IM(i,j);
               sxx = sxx+IM(i,j)*IM(i,j);
               cnt=cnt+1;
            endif;
         endfor;
      endfor;

      IMV(x,y) = (sxx-sx*sx/cnt)/cnt;                      // local variance
   endfor;
endfor;
```

Algorithm 8.1 Local variance filter. This filter returns the local variance of input image IM(x,y) of size xmax, ymax in the local neighborhood of size 2L+1. The output image is IMV(x,y).

would contain the local coefficient of variation. Using equations analogous to Equation (8.6), adaptations of Algorithm 8.1 to compute local skew [Equation (8.7)] and local kurtosis [Equation (8.8)] can easily be implemented.

$$\text{skew}(x,y) = \frac{1}{N\sigma_{\text{loc}}^3(x,y)} \left\{ \sum_{i=x-l}^{x+l} \sum_{j=y-l}^{y+l} \left[I(i,j) - \overline{I}(x,y) \right]^3 \right\} \tag{8.7}$$

$$\text{kurtosis}(x,y) = \frac{1}{N\sigma_{\text{loc}}^3(x,y)} \left\{ \sum_{i=x-l}^{x+l} \sum_{j=y-l}^{y+l} \left[I(i,j) - \overline{I}(x,y) \right]^4 \right\} - 3 \tag{8.8}$$

In Equations (8.7) and (8.8), $\sigma_{\text{loc}}(x,y)$ denotes the local standard deviation at coordinate x,y, and $\overline{I}(x,y)$ denotes the local mean value.

8.2. TEXTURE CLASSIFICATION WITH LOCAL NEIGHBORHOOD METHODS

8.2.1. Texture Classification Based on the Co-occurrence Matrix

Many texture classification operations require computation of the co-occurrence matrix. These methods were pioneered in 1973 by Haralick et al.,[15,16] who formulated

14 texture metrics that build on the co-occurrence matrix. The gray-scale co-occurrence matrix C_d^θ (i,j) is a two-dimensional histogram of gray values i and j where C_d^θ (i,j) specifies the probability that a pixel with gray level j is located at a distance d and direction θ from a pixel with gray level i. Therefore, the co-occurrence matrix of an image with 256 gray levels has the size of 256×256 elements. The element $C(0,1)$ of a co-occurrence matrix with $d = 1$ and $\theta = 0$ contains the probability that the right neighbor of a pixel with value 0 ($i = 0$) is a pixel with value 1 ($j = 1$). The diagonal elements of a co-occurrence matrix contain the probabilities that two neighbors have the same gray value. For the choice of $d = 1$, four directions of θ are possible: 0°, 45°, 90°, and 135°.

Uncorrelated pixels have a broad co-occurrence matrix. Conversely, a coarse texture with respect to the displacement vector d tends to have a narrow distribution along the diagonal. Such a distribution indicates that the probability of having a pixel with a similar intensity value in its neighborhood is very high. This behavior is demonstrated in Figure 8.4. The co-occurrence matrix of uncorrelated noise with a box probability distribution is very broad, indicating that the neighborhood of any value pair i, j is equally probable. The co-occurrence matrix of Gaussian noise strongly reflects the isotropic Gaussian distribution. Finally, Perlin noise can be interpreted as a coarse texture with highly correlated pixels: The probability of neighboring pixels having the same or nearly the same intensity value is much higher than the probability of pixel pairs with strongly differing intensities. Consequently, the co-occurrence matrix shows values of high probability clustered around the diagonal. Figure 8.5 shows several examples of co-occurrence matrices corresponding to the texture examples in Figure 8.1. A longer displacement vector ($d = 4$) was chosen, and pairs of co-occurrence matrices with perpendicular displacements are shown ($\theta = 0^\circ$ and $\theta = 90^\circ$). The co-occurrence matrices of the bark texture show a slightly skewed distribution, which is widely direction-independent and, being narrower than the distribution in Figure 8.4F, shows a higher degree of correlation than the knit texture. Corduroy, with its highly anisotropic texture, shows very different co-occurrence matrices with 0° and 90° displacement vectors. In fact, the distinct shape seen in Figure 8.5D indicates a repeating pattern: a high probability of a bright pixel followed by a dark one, and vice versa. The knit texture shows a broad but anisotropic distribution.

Haralick et al.[15,16] have defined 14 global metrics to classify the texture by quantitatively describing the shape of the co-occurrence matrix. Many of these metrics are mutually dependent, and only some key metrics will be introduced here. Let $P(i,j)$ describe the compound probability of a pixel pair at displacement d and displacement direction θ; then some of the feature metrics can be described by the equations in Table 8.3.

The average values in the i and j directions can be computed as

$$\begin{aligned} \mu_i &= \sum_i \sum_j iP(i,j) \\ \mu_j &= \sum_j \sum_i jP(i,j) \end{aligned} \tag{8.18}$$

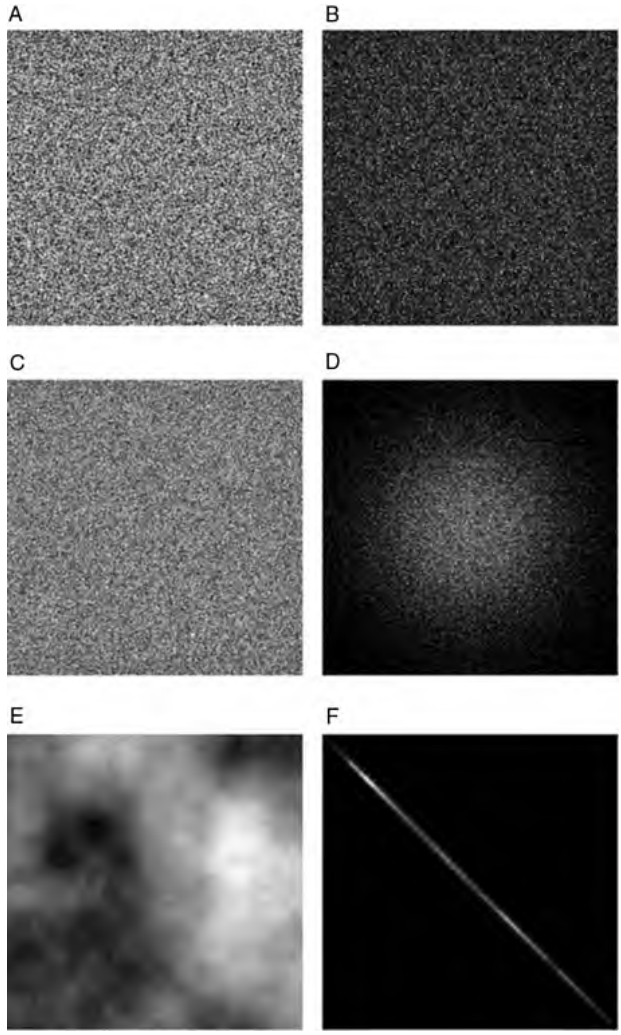


FIGURE 8.4 Co-occurrence matrices for different types of noise ($d = 1$, $\theta = 0$). From top to bottom, uncorrelated, equally distributed noise (A) has a very broad co-occurrence matrix (B); Gaussian noise (C) has a symmetrical co-occurrence matrix (D) that reflects the Gaussian distribution; Perlin noise (E) represents a coarse texture, where the co-occurrence matrix (F) is widely limited to the diagonal.

and the corresponding standard deviations σ_i and σ_j required to compute the contrast can be computed as

$$\begin{aligned} \sigma_i &= \sum_i \sum_j (i - \mu_i)^2 P(i,j) \\ \sigma_j &= \sum_j \sum_i (j - \mu_j)^2 P(i,j) \end{aligned} \tag{8.19}$$

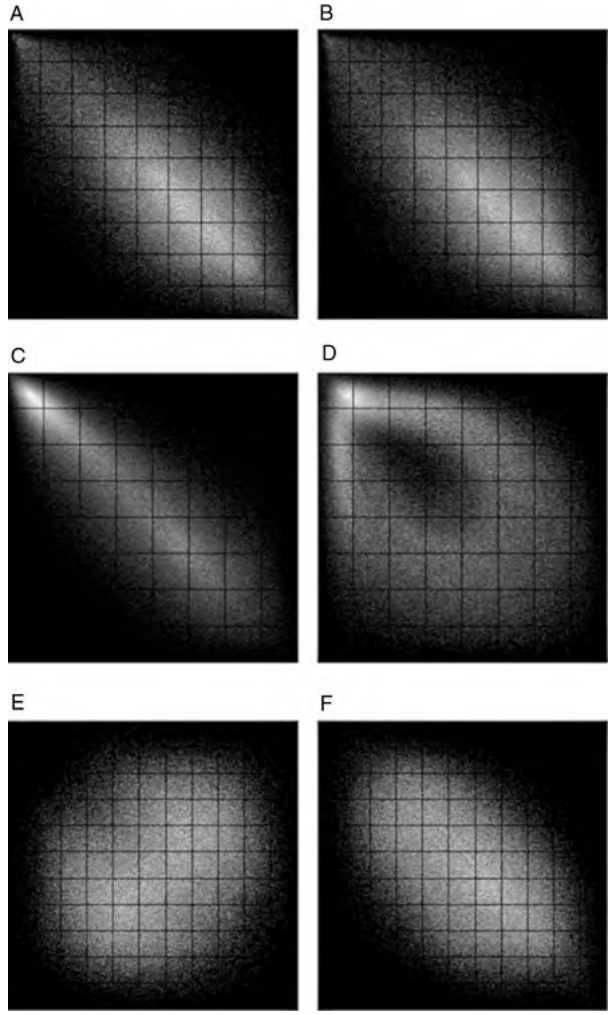


FIGURE 8.5 Co-occurrence matrices for the bark, corduroy, and knit textures in Figure 8.1. In all cases, $d = 4$ pixels. From top to bottom, the co-occurrence matrix for the bark texture at $\theta = 0^\circ$ (A) and $\theta = 90^\circ$ (B), for the corduroy texture at $\theta = 0^\circ$ (C) and $\theta = 90^\circ$ (D), and for the knit texture at $\theta = 0^\circ$ (E) and $\theta = 90^\circ$ (F) are shown. All co-occurrence matrices have been intensity-normalized for better visualization.

Numerical examples for the sample textures in this chapter (Figures 8.1, 8.2, and 8.4) are given in Table 8.4. The purpose of computing several different texture classification metrics is to obtain a multidimensional texture vector. Although it is sometimes possible that a single quantitative descriptor differentiates two textures (e.g., the liver and muscle regions in the ultrasound image in Figure 8.2), more often an image contains multiple textures with potentially overlapping values of

TABLE 8.3 Haralick's Texture Classification Metrics

Texture Feature	Description	Formula	
Energy	Defined in a manner analogous to Equation (8.4).	$\sum_i \sum_j P^2(i,j)$	(8.9)
Entropy	Information content, defined in a manner analogous to Equation (8.5).	$-\sum_i \sum_j P(i,j) \log_2 P(i,j)$	(8.10)
Contrast	Weighs probabilities by their distance to the diagonal.	$\sum_i \sum_j \lvert i-j \rvert P(i,j)$	(8.11)
Inertia	An exaggeration of the contrast metric, as it weighs probabilities by the square of the distance to the diagonal.	$\sum_i \sum_j (i-j)^2 P(i,j)$	(8.12)
Correlation	Increases in the presence of large homogeneous regions.	$\sum_i \sum_j \frac{ij\,P(i,j) - \mu_i \mu_j}{\sigma_i \sigma_j}$	(8.13)
Texture homogeneity	Weighs the probabilities by proximity to the diagonal and is therefore the complement of contrast.	$\sum_i \sum_j \frac{P(i,j)}{1+\lvert i-j \rvert}$	(8.14)
Inverse difference	Valid only for nondiagonal elements $i \neq j$. Closely related to texture homogeneity.	$\sum_i \sum_j \frac{P(i,j)}{\lvert i-j \rvert}$	(8.15)
Inverse difference moment	The metric complementary to inertia. Probabilities are weighted by proximity to the diagonal.	$\sum_i \sum_j \frac{P(i,j)}{1+(i-j)^2}$	(8.16)
Cluster tendency	Probabilities are weighted by their deviation from the mean values.	$\sum_i \sum_j (i-\mu_i+j-\mu_j)^2 P(i,j)$	(8.17)

some metrics. Multidimensional clustering techniques, that is, finding vectors that are spatially related in the n-dimensional feature space, can be used to better differentiate between different textures. One of the key challenges associated with the texture classification metrics that build on the co-occurrence matrix is the determination of a suitable displacement vector. Generally, some experimentation is necessary. Ideally, a training set of images is available where the feature vector of two tissues to be differentiated (e.g., tissue of interest and background or healthy tissue and diseased tissue) can be computed as a function of d, and a value of d is chosen where the distance between the feature vectors is maximal. In general, a larger value of d broadens the co-occurrence matrix (Figure 8.6) and therefore

TABLE 8.4 Examples of Texture Feature Metrics for Selected Textures Shown in Figures 8.1, 8.2, and 8.4

Texture[a]	Entropy	Contrast	Correlation	Inverse Difference	Inertia
White noise	15.2	85.0	0.002	0.041	10,866
Gaussian noise	14.6	55.6	0.0009	0.056	4,796
Perlin noise	10.9	1.7	0.999	0.521	4.73
Bark	13.7	29.2	0.63	0.095	1,449
Corduroy	13.3	19.2	0.90	0.15	746
$\theta = 0^\circ$ $\theta = 90^\circ$	14.14	49.6	0.46	0.065	3,851
Knit	14.2	64.5	−0.27	0.043	5,933
Liver ultrasound					
Liver region	11.4	11.2	0.78	0.2	216
Muscle region	11.39	8.9	0.89	0.23	142

[a]For the noise textures, $d = 1$ and $\theta = 0^\circ$; for the textures in Figure 8.1, $d = 1$ and $\theta = 0^\circ$ unless otherwise stated; for the textures in Figure 8.2, $d = 2$ and $\theta = 0^\circ$.

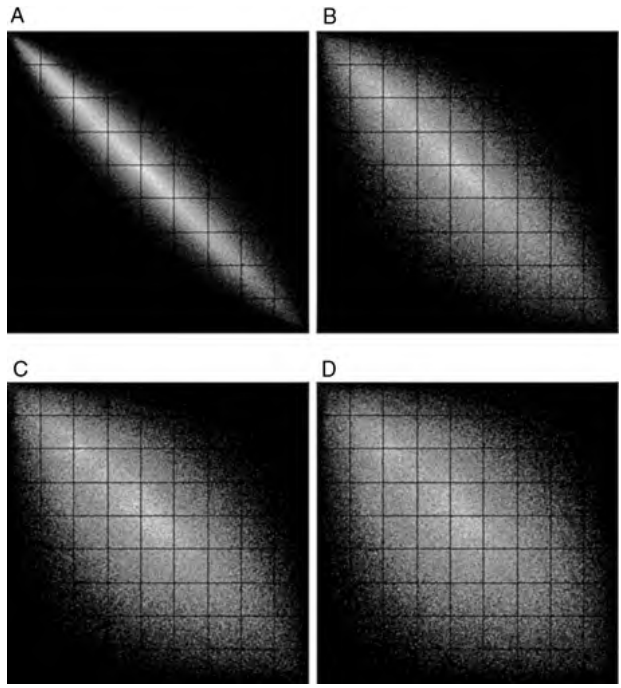


FIGURE 8.6 Changes in the co-occurrence matrix with the choice of the displacement vector d. The underlying texture is the carpet texture in Figure 8.1, and the displacement vector was $d = 1$ (A), $d = 2$ (B), $d = 3$ (C), and $d = 4$ (D), with $\theta = 0$ in all cases.

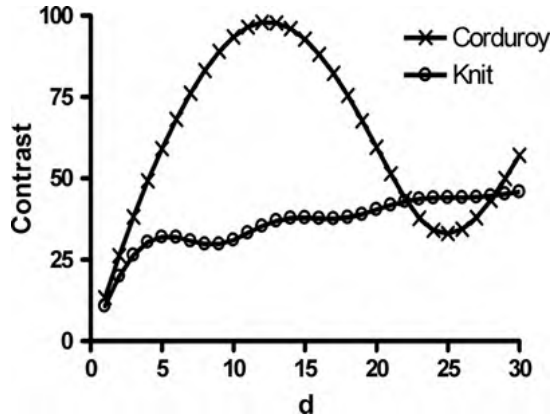


FIGURE 8.7 Influence of the distance d on texture classifiers: in this example, contrast. For $\theta = 90^\circ$, the strong periodicity of the corduroy texture can be seen. The knit texture exhibits weaker periodicity.

increases entropy, contrast, and inertia and decreases energy, correlation, inverse difference, and the clustering tendency.

With a small value for d, noise will dominate the co-occurrence matrix. Generally, d should be chosen to match the size of a texture element. A good example is the corduroy texture in Figure 8.1, where the ridges and valleys in the vertical direction are about 10 pixels apart. Choosing $d = 10$ would therefore provide the largest difference between a horizontal and a vertical displacement. Further increasing d reduces this difference. This effect is demonstrated in Figure 8.7, where the contrast as a function of displacement d with $\theta = 90^\circ$ is shown. A maximum contrast of the corduroy texture is found at $d = 12$, which accurately measures the average periodicity of the texture in the vertical direction. In contrast, the knit texture does not exhibit a similarly strong periodicity, although relative maxima can be seen at $d = 5$ and $d = 15$. The size of a knit texture element is therefore approximately five pixels. For the description of the knit texture, $d = 5$ would therefore be a good choice.

The second flexible parameter is the angle θ. If anisotropic properties of the texture are of no relevance and there is only weak anisotropy, an approach used frequently is to compute the texture feature metrics over four angles ($\theta = 0^\circ$, 45°, 90°, and 135°) and average the resulting four directional values to yield a single value. With this approach, the feature vector becomes rotation-invariant. Alternatively, the maximum or minimum value for the feature metric over all rotations can be determined. This second approach emphasizes anisotropic properties more strongly.

Examination of the feature vector as a function of θ becomes important to examine the anisotropic properties of the texture. For example, the maximum and minimum value of the contrast $c(\theta)$ for different values of θ can be determined, and its ratio ζ

is a metric for the contrast anisotropy.

$$\zeta = \frac{\max_\theta c(\theta)}{\min_\theta c(\theta)} \tag{8.20}$$

If anisotropy is a characteristic attribute of a specific texture, it is reasonable to include one or more anisotropy measures in the feature vector.

Like a histogram, the co-occurrence matrix can be computed within a restricted region of interest. To compute the feature vector of the liver and muscle regions in Figure 8.2, the region can be delineated (segmented) manually and subjected to texture analysis. Furthermore, the computation of the co-occurrence matrix can be restricted to a moving window around a cental pixel. In this case, a local feature vector can be computed per pixel, resulting in an enhanced image where the individual pixels can be classified (and segmented) by their feature vector. This is illustrated in Figure 8.8, an image false-colored with the intensities of the clustering feature in the red channel, the contrast feature in the green channel, and the inertia feature in the blue channel. All local features were computed in a circle with a radius of 15 pixels. To restore the original image detail, the image was converted from a red–green–blue (RGB) composite image to the hue–saturation–value (HSV) model, and the value channel was replaced by the original image in Figure 8.2. The resulting HSV image with a modified value channel was again converted to the RGB model in Figure 8.8. Different hues represent different feature vectors. The walls of the *vena cava* are clearly visible in red. The liver region has a more bluish hue than the surrounding muscle region, which is of a deeper green shade.

Pseudocode to compute the co-occurrence matrix is provided in Algorithm 8.2. The resulting co-occurrence matrix (computed from an arbitrary Cartesian displacement vector `dx`, `dy`) is a square matrix with dimensions determined by the number of bins.

FIGURE 8.8 Composite image of the ultrasound image in Figure 8.2. The red color channel represents clustering, the green channel represents contrast, and the blue channel represents inertia. (*See insert for color representation of the figure*.)

```
set numbins=128;                    // Choose the number of discrete gray levels
set dx=4;                           // dx and dy determine the displacement (Cartesian)
set dy=0;                           // which can also be computed from d and θ (polar)
delta = 256/numbins;                // bin size for 256 gray values
gmax = numbins;                     // size of GLCM matrix
allocate GLCM(gmax,gmax);           // Allocate GLCM output image
set cnt=0;                          // number of pixels counted

for (y=0 while y<gmax increment y=y+1) do // double loop over the glcm image
   for (x=0 while x<gmax increment x=x+1) do
      GLCM(x,y)=0;                        // Reset accumulators
   endfor;
endfor;

for (y=0 while y<ym increment y=y+1) do    // double loop over the input image
   for (x=0 while x<xm increment x=x+1) do

      x1=x+dy; y1=y+dy;                   // shifted coordinate
      if ((x1≥0) and (y1≥0) and (x1<xmax) and (y1<ymax)) then
         bin1 = IM(x,y)/delta;
         bin2 = IM(x1,y1)/delta;
         if ((bin1≥0) and (bin2≥0) and (bin1<gmax) and (bin2<gmax)) then
            GLCM(bin1,bin2) = GLCM(bin1,bin2)+1;
            cnt=cnt+1;
         endif;
      endif;

   endfor;
endfor;

for (y=0 while y<gmax increment y=y+1) do // Normalize GLCM to contain probabilities
   for (x=0 while x<gmax increment x=x+1) do      // and not counts
      GLCM(x,y)=GLCM(x,y)/cnt;
   endfor;
endfor;
```

Algorithm 8.2 Co-occurrence matrix. This algorithm computes the gray-level co-occurrence matrix `GLCM(x,y)` from an input image `IM(x,y)`. IM is assumed to have 256 gray levels and has a size of `xmax`, `ymax`. Binning is possible (`numbins`), and the number of bins determines the final size of `GLCM`.

Binning is reasonable when computing the co-occurrence matrix of small regions. Four gray levels of an 8-bit gray-scale image, for example, could be combined in one bin, and the co-occurrence matrix would reflect 64 gray levels only. The output co-occurrence matrix is normalized to reflect the probability of a gray-scale pair, $P(i,j)$, rather than the number of gray-scale pairs $n(i,j)$. The matrix will therefore satisfy the condition

$$\sum_i \sum_j P(i,j) = 1 \tag{8.21}$$

From the co-occurrence matrix, it is straightforward to compute the feature vector defined by Equations (8.9) through (8.17) by substituting `GLCM(i,j)` for $P(i,j)$.

8.2.2. Laws' Texture Energy Metrics

K. I. Laws proposed a different approach to texture classification[25] by suggesting five different one-dimensional convolution kernels:

$$\begin{aligned} L_5 &= [\ 1 \quad 2 \quad 4 \quad 2 \quad 1] \\ E_5 &= [-1 \quad -2 \quad 0 \quad 2 \quad 1] \\ S_5 &= [-1 \quad 0 \quad 2 \quad 0 \quad -1] \\ R_5 &= [\ 1 \quad -4 \quad 6 \quad -4 \quad 1] \\ W_5 &= [-1 \quad 2 \quad 0 \quad -2 \quad 1] \end{aligned} \tag{8.22}$$

Although the kernel size may be larger or smaller in different implementations, the length 5 kernels are being used most commonly. L_5 is a Gaussian-type blurring kernel that provides the smoothed gray level (L) of the texture. E_5 is a gradient kernel, related to the Sobel and compass edge (E) detectors. S_5 is derived from the Laplacian operator and emphasizes spots (S). R_5 emphasizes pixel-sized ripples (R), and W_5 emphasizes waves (W) that stretch over a few pixels. To obtain two-dimensional convolution kernels, one of the one-dimensional kernels in Equation (8.22) is multiplied by the transpose of another kernel. For example, $L_5^T L_5$ yields the well-known 5×5 Gaussian smoothing kernel. A second example is the combination kernel

$$L_5^T R_5 = \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ 2 & -8 & 12 & -8 & 2 \\ 4 & -16 & 24 & -16 & 4 \\ 2 & -8 & 12 & -8 & 2 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix} \tag{8.23}$$

which is sensitive toward ripples in the horizontal direction and smoothes the texture in the vertical direction. By multiplying pairs of the one-dimensional kernels, a theoretical combination of 25 convolution kernels K_N is possible. Of these, only $L_5^T L_5$ has a nonzero sum of elements and requires normalization with a factor of 1/100. It is possible to obtain up to 25 "feature images" F_N by convolving the image I with each of the 25 combination kernels K_N ($1 \leq N \leq 25$). The convolution step should be preceded by background flattening (e.g., background removal by unsharp masking or Fourier highpass filtering), and the convolved feature images should be further processed by smoothing the absolute values of F_N[25]:

$$E_N(x,y) = \frac{1}{(2l+1)^2} \sum_{j=y-l}^{y+l} \sum_{i=x-l}^{x+l} |F_N(i,j)| \tag{8.24}$$

The resulting E_N are called *energy maps*. Each image pixel $I(x,y)$ is now associated with a vector of energies E_N (x,y) that can be used to classify the pixel by means of multidimensional thresholding, clustering techniques, grow-merge or split-merge methods, or artificial intelligence methods such as neural networks.

Since the energy maps are not rotation-independent [as can be seen in Equation (8.23)], additional features may be devised, for example, through the generation of horizontal and vertical ripple images R_X and R_Y with the kernels $L_5^T R_5$ and $R_5^T L_5$, followed by computing a compound ripple-emphasized image $R(x,y) = \sqrt{R_X^2(x,y) + R_Y^2(x,y)}$ in analogy to the Sobel edge detector. In addition, directionality features may be computed by convolving the image with the kernels $L_5^T R_5$ and $R_5^T L_5$, followed by computing a pixel-wise ratio $R_X\ (x,y)/R_Y\ (x,y)$. On the other hand, not all 25 combinations of the one-dimensional kernels are practical. A kernel obtained from $W_5^T R_5$, for example, will produce an energy image that is difficult to interpret. For this reason, only a limited number of kernels are used in practice, and the feature vector per pixel usually has fewer than 10 elements.

The advantage of Laws' texture energies is their local nature. While methods based on the histogram and the co-occurrence matrix may be restricted locally through definition of a region of interest, the choice of a small region can restrict the number of pixels to a level where they no longer represent the texture statistically. The local nature is inherent in Laws' texture energy maps. However, as with most texture classification methods, the neighborhood size (in this case the size of the kernels) makes the texture energies sensitive to scaling. Texture extraction, the definition of a feature vector for a pixel or a region in the image, is therefore usually one step in a three-process step: image preprocessing, texture extraction, and texture-based segmentation.

8.2.3. Autocorrelation-Based Texture Classification

The autocorrelation function is the convolution of an image $I(x,y)$ with a shifted version of itself:

$$\rho(\Delta x,\Delta y) = \frac{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} I(x,y)I(x+\Delta x, y+\Delta y)}{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} I(x,y)^2} \tag{8.25}$$

The autocorrelation function is point-symmetric around the origin [i.e., $\rho(\Delta x,\Delta y) = \rho(-\Delta x, -\Delta y]$. Therefore, it is sufficient to compute the autocorrelation function for only two quadrants. The maximum value of the autocorrelation function is always in the origin where $\rho(0,0) = 1$. The autocorrelation function is suited particularly for the detection of repeated patterns, but the broadness of the peak near the origin contains information on the coarseness of the texture. Figure 8.9 shows the autocorrelation functions of the sample textures in Figure 8.1. In each case, the center of the image represents the origin of the autocorrelation function $\rho(0,0)$, and Δx increases from left to right, whereas Δy increases from top to bottom. The periodicity of the corduroy and knit patterns becomes particularly obvious, but the glass texture exhibits two local maxima close to the origin that also indicate a certain periodicity. The irregular textures show a dominant cental peak only. However, the broadness of the central peak conveys information on the coarseness of the texture. The broadest peak (indicating a

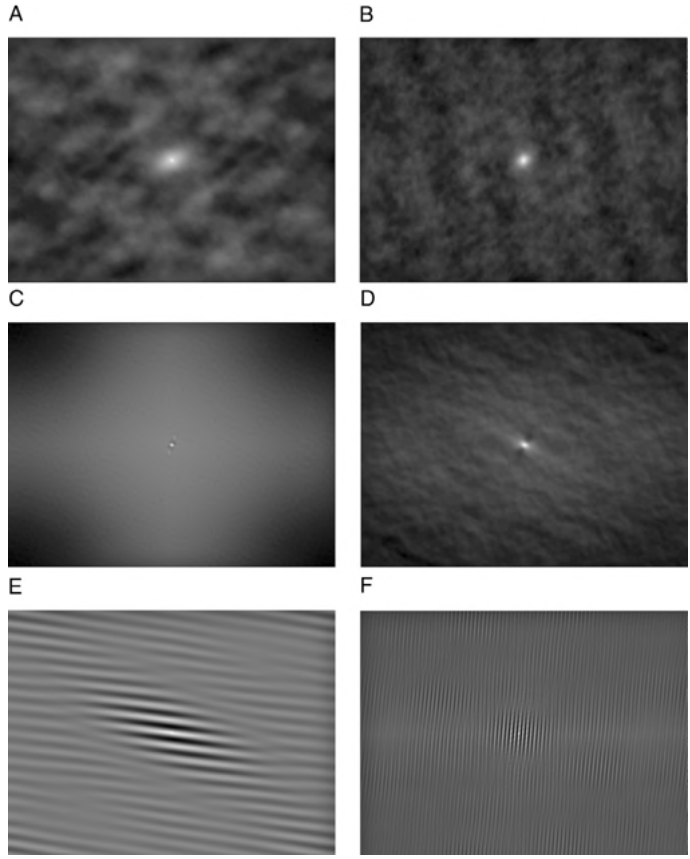


FIGURE 8.9 Autocorrelation functions of the sample textures in Figure 8.1. From top left to bottom right, the images show the autocorrelation images of tree bark (A), polished granite (B), glass (C), carpet (D), corduroy (E), and knit (F). All autocorrelation functions have been contrast-enhanced for better visual representation.

coarse texture) is associated with the bark texture, followed by the granite and carpet textures.

It is difficult to extract single-quantity (scalar) metrics from the autocorrelation image as shown in Figure 8.9, computed for the entire possible range of Δx and Δy. A simple solution is the computation of scalar values for $\rho(0,k)$, $\rho(k,0)$, $\rho(k,k)$, and $\rho(-k,k)$, where k is a parameter linearly related to the size of the texture. These four values may be used as a feature vector. The average of these four values, $\bar{\rho}(k)$, may be used as a single metric or provide a different feature vector for increasing k. Also, the ratio of perpendicular autocorrelation values, for example, $\rho(0,k)/\rho(k,0)$, can be used in a feature vector to represent the anisotropy.

Alternatively, higher-order autocorrelation functions can be defined.[24] Equation (8.25) describes a first-order autocorrelation function. A second-order autocorrelation

function contains a second displaced term:

$$\rho(\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2) = \frac{\sum_{y=0}^{M-1} \sum_{x=0}^{N-1} I(x,y) I(x+\Delta x_1, y+\Delta y_1) I(x+\Delta x_2, y+\Delta y_2)}{\sum_{y=0}^{M-1} \sum_{x=0}^{N-1} I(x,y)^3} \quad (8.26)$$

If the displacement is restricted to a 3 × 3 neighborhood (i.e., $\Delta x \in \{-1, 0, 1\}$ and $\Delta y \in \{-1, 0, 1\}$), 20 different combinations of the two displacements Δx_1,Δy_1 and Δx_2, Δy_2 are possible. Together with the four possible first-order autocorrelation values and the local neighborhood average intensity, a feature vector with 25 elements emerges. Again, it is possible to combine different displacement sizes (i.e., $\Delta x \in \{-k,0,k\}$ and $\Delta y \in \{-k,0,k\}$) to an even higher-order feature vector that contains different scales. Toyoda and Hasegawa[38] propose higher-order autocorrelation values up to the eighth order, a computation that allows for a total 223 combinations of displacements. For such high-order vectors, neural networks are commonly used for further processing.[24]

Analysis of the central peak reveals information about the coarseness of the texture and can also reveal possible self-similar properties. An additional possible texture feature metric is the broadness of the central peak (expressed, for example, as full width at half maximum, the width of the peak where it drops halfway from the highest to the lowest value), or the decay of an exponential function fitted into $\rho(r)$, where r describes a straight line starting in the origin. In the example of Figure 8.9, the decay constants are 0.001 (bark), 0.0015 (granite), and 0.0024 (carpet), providing a suitable metric of the coarseness of the texture. Moreover, if the decay function $\rho(r)$ follows a power-law decay for small values of r [i.e., $\rho(r) \propto r^{-\beta}$], some fractal properties of the texture can be assumed,[20] and the value of β, determined by nonlinear regression, quantifies the scaling behavior. In the examples above, a power-law curve fit into the bark texture gives poor regression, and consequently, the bark texture would not be considered to be fractal. However, power-law curve fits are possible with $r^2 \geq 0.998$ for both granite and carpet, providing values for β of 0.017 and 0.028, respectively. Both textures can therefore be considered fractal with power-law scaling properties in the autocorrelation function, and the value of β may be used to differentiate the textures.

8.3. FREQUENCY-DOMAIN METHODS FOR TEXTURE CLASSIFICATION

In the frequency domain, texture properties such as coarseness, graininess, or repeating patterns can be identified. Of primary interest is the spectrum's magnitude or the squared magnitude, (i.e., the power). Since the Fourier transform does not retain spatial information, only global properties can be examined. To illustrate the relationship between the visual appearance of a texture and its frequency-domain representation, Figure 8.10 shows the logarithmic magnitude of the Fourier transforms of the

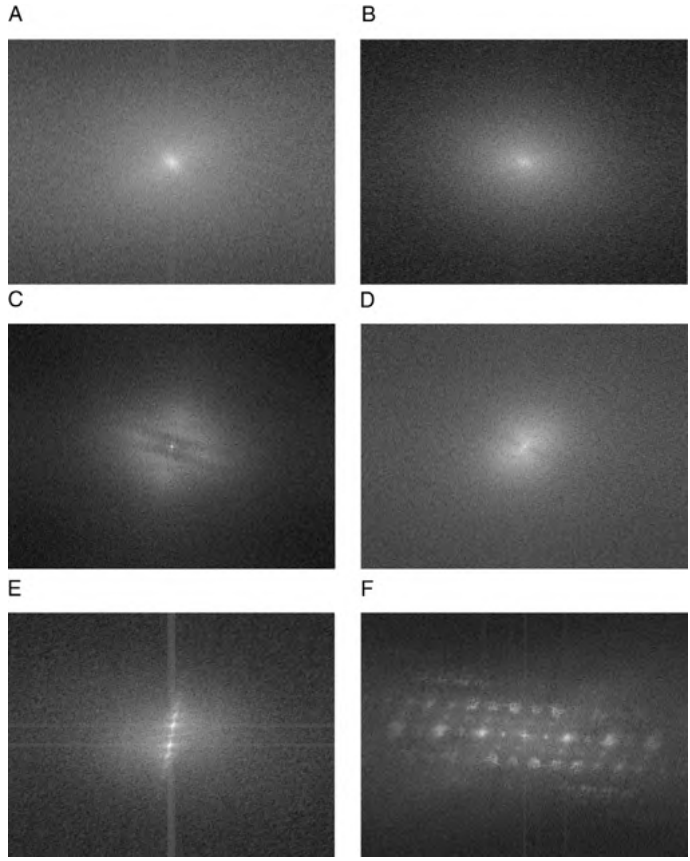


FIGURE 8.10 Magnitude of the Fourier transform of the images in Figure 8.1. From top left to bottom right, the images show the magnitude of the Fourier spectrum of the tree bark (A), polished granite (B), glass (C), carpet (D), corduroy (E), and knit (F) images. Particularly in the bottom row, the periodicity of the pattern becomes visible in the form of distinct multiple peaks.

textures in Figure 8.1. Directionality and periodicity can be particularly well seen in the corduroy and knit examples (the bottom row). Irregular patterns can better be characterized by the decay of the magnitude with increasing frequencies. Examples are shown in Figure 8.11 for the bark, granite, and carpet textures. Frequencies have been sampled along radial lines starting at the origin of the Fourier transform image in 25° intervals from 0° to 175° and averaged. The averaging of Fourier transform data along circles of constant frequency reduces both the noise component and the effects of possibly anisotropic decays. If the texture is dominated by few large and relatively homogeneous features (e.g., bark), low-frequency components dominate the Fourier spectrum, and a rapid drop-off toward higher frequencies can be seen. A robust intensity-invariant quantitative feature can be extracted by averaging several samples at low frequencies and dividing this average by a similar average at higher

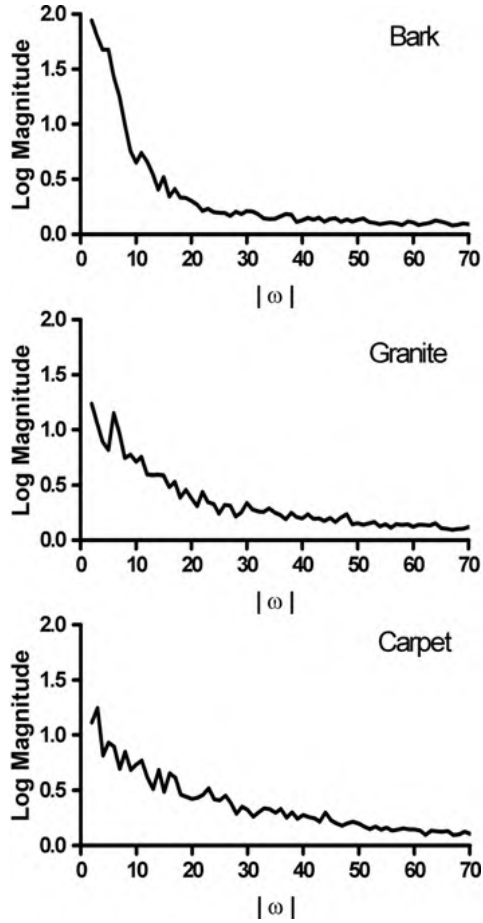


FIGURE 8.11 Different magnitude decay behavior with higher frequencies. A texture that is dominated by few, relatively homogeneous large areas (like the bark texture) has a large magnitude at low frequencies and drops off rapidly toward higher frequencies. The granite and carpet samples both show a more moderate decay of the magnitude at high frequencies, which indicates a more irregular size distribution of the patterns.

frequencies. Very high frequencies should be avoided because those regions are usually dominated by noise. The average frequency from 5 to 10 per pixel, divided by the average frequency from 40 to 45 per pixel, yields the ratios 9.8 (bark), 4.6 (granite), and 3.5 (carpet). This value provides a measure of the dominance of large features in relation to small features. Furthermore, self-similar properties of the texture can easily be identified. If the magnitude $A(\omega)$ decays with increasing frequency ω in a power-law fashion following the equation

$$A(\omega) \propto \omega^{-\beta} \tag{8.27}$$

(i.e., the data points of magnitude over frequency in a double-logarithmic scale lie on a straight line), self-similar behavior of the texture pattern exists, and the Hurst exponent ($H = \beta/2 - 1$) and the fractal dimension ($D = 2 - H$) can be used to quantify this behavior.

The relationship of the fractal dimension to frequency scaling behavior is explained in detail in Section 10.4. In the example in Figure 8.10, the bark texture exhibits a poor regression fit and cannot be considered to be self-similar. In the cases of carpet and granite, β can be determined by linear regression of the log-transformed data with $r^2 > 0.97$, and the slope of the fit is 0.29 in both cases. Finding power-law scaling behavior in the carpet and granite textures but not in the bark texture matches the observation of similar scaling behavior in the autocorrelation function. However, the scaling exponents differed strongly when computed with the autocorrelation function but not with the frequency spectrum. Some experimentation is usually needed to find suitable texture metrics that represent the texture and distinguish it from other textures. This is one of the reasons why high-dimensional feature vectors are extracted and analyzed.

In addition to averaging the magnitude decay over various angles, it is possible to use orthogonal directions to obtain information on the texture anisotropy. The quantity described above, that is, the magnitude at low frequencies divided by magnitude at high frequencies, can be computed in different directions. This ratio R, computed over various angles θ, shows low variability in isotropic textures. Examples are the bark and granite textures. Conversely, corduroy has a strong directional component and would therefore show a large variability of $R(\theta)$. Two possible quantitative metrics are (a) the ratio of the largest to the smallest value of $R(\theta)$, or (b) the coefficient of variation of $R(\theta)$. Very similar to the metrics obtained in the spatial domain, the purpose of quantitatively describing multiple frequency-domain texture properties is to obtain a feature vector that allows classification of the texture.

Although when applied globally, the Fourier transform loses all spatial information, it is possible to subdivide the image into separate regions before performing the Fourier transform. If the region of interest that contains the texture is known (e.g., after segmentation), it is possible to perform the Fourier transform on the bounding box of the feature. Although it seems obvious to pad the image values outside the region of interest with zeros, a possible abrupt intensity change from the feature to the surrounding zero-valued region adds additional frequency components that need to be accounted for. Alternatively, windowing functions as described in Section 3.2 can reduce those artifactual frequency components.

8.4. RUN LENGTHS

The run-length method[13] to classify texture is very popular and widespread in the analysis of biomedical images. A *run* is a sequence, in a straight scan direction, of pixels with identical image value. The associated run length is the length of the run, usually the number of pixels for the horizontal or vertical scan direction, or the number of pixels multiplied by $\sqrt{2}$ for a diagonal direction. The notion of run lengths

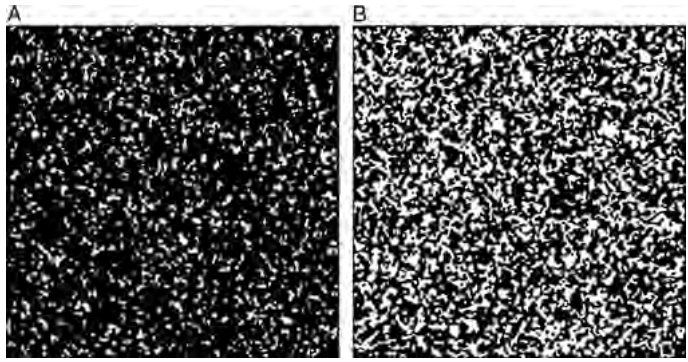


FIGURE 8.12 Test images for demonstration of the run-length method.

is related to the transmission of telefax data. Since black-and-white telefax pages contain large areas of contiguous white, it is more efficient to transfer the information "120 pixels of white" than actually transmitting 120 white pixels. In this example, 120 would be the length of a white run. On a black-and-white image, black and white runs can be found, and their length distribution characterized in two histograms. Usually, the image is scanned in four directions: 0°, 45°, 90°, and 135°. The remaining four directions have identical histograms, due to symmetry.

Consider the test image in Figure 8.12. The pattern was generated using a cellular automaton algorithm, where a square lattice was filled initially with black pixels and where white pixels were randomly seeded with a probability P. Iteratively, each pixel was set to white when the majority of pixels in its neighborhood (i.e., five or more pixels) were white, and set to black otherwise. Iterations are stopped when the pattern converges. The seeding density was $P = 42\%$ in Figure 8.12A and $P = 48\%$ in Figure 8.12B. Consequently, Figure 8.12A is dominated by longer black runs compared to Figure 8.12B. The corresponding run-length histograms, scanned in a horizontal direction, are shown in Figure 8.13.

The most obvious difference is the distribution of black runs in Figure 8.12A that is indicated by the broad distribution of the histogram in Figure 8.13A (black line), as opposed to the narrower distribution of black runs in Figure 8.12B (gray line in Figure 8.13A). The distribution of white runs (histogram in Figure 8.13B) is not fundamentally different between the two images in Figure 8.12A and B, although it becomes evident that a larger overall number of white runs exists in Figure 8.12B. If scalar values are needed to describe the texture, the histograms can be further quantified by using the moments of the histogram or by fitting a nonlinear function into the histogram.

The notion of run lengths can be extended to gray-scale images. In gray-scale images, a run extends over pixels with the same gray value. Since this metric is strongly affected by noise and minor intensity fluctuations, it is necessary to quantize the gray-scale values. A run would now be defined as a contiguous sequence of pixels in the scan direction, with pixel intensities falling inside a certain range. If

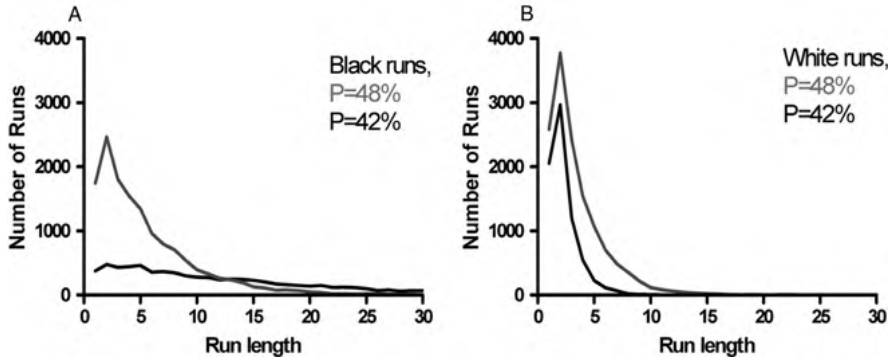


FIGURE 8.13 Run-length histograms for the test images in Figure 8.12: (A) the black runs; (B) the white runs.

an image has 256 gray levels, it could, for example, be quantized into 16 gray-scale bins, ranging in intensity from 0 to 15, 16 to 31, 32 to 47, . . . , 239 to 255. The optimum bin size depends on the gray-scale variation of the texture. Suitable texture preprocessing (contrast enhancement, background flattening, histogram equalization, and noise reduction) can strongly improve the representation of a specific texture by run lengths. In the case of gray-scale images, the run-length histogram for each scan direction becomes two-dimensional, with one axis being the run length, and the other axis the gray value or gray-value bin. The probability of a specific run length is therefore $P(g,l)$, where g is the gray-value bin and l is the length. If the histogram has been normalized, then

$$\sum_{g=0}^{G-1} \sum_{l=1}^{L} P(g,l) = 1 \tag{8.28}$$

where G is the number of gray-scale bins (the number of bins into which the image has been quantized) and L is the longest run. From the two-dimensional run-length histogram, a feature vector containing seven scalar descriptive values can be extracted. These are commonly known as SRE (short-run emphasis), LRE (long-run emphasis), HGRE (high gray-level emphasis), LGRE (low gray-level emphasis), RLNU (run length nonuniformity), GLNU (gray-level nonuniformity), and RPC (run percentage). In addition, pairwise combinations of short- and long-run emphasis with low and high gray-level emphasis result in the combined values of SRLGE (short-run, low gray-level emphasis), SRHGE (short-run, high gray-level emphasis), LRLGE (long-run, low gray-level emphasis), and LRHGE (long-run, high gray-level emphasis). Equations to compute these quantitative metrics are listed in Table 8.5.

As with all other texture extraction methods, the texture is associated with a multidimensional feature vector suitable for multidimensional thresholding or clustering. Each of the 11 metrics will generally be different for the four directions. The additional information may be used to determine texture anisotropy. For example,

TABLE 8.5 Definition of Run-Length Metrics

Description	Equation
Short-run emphasis. This metric increases when short runs dominate, for example, in fine-grained textures.	$\text{SRE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{P(g,l)}{l^2}$ (8.29)
Long-run emphasis. This metric increases when long runs dominate, for example, in textures with large homogeneous areas or coarse textures.	$\text{LRE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}P(g,l)l^2$ (8.30)
Low gray-level emphasis. Emphasis is orthogonal to SRE, and the metric increases when the texture is dominated by many runs of low gray value.	$\text{LGRE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{P(g,l)}{(g+1)^2}$ (8.31)
High gray-level emphasis. Emphasis is orthogonal to LRE, and the metric increases when the texture is dominated by many runs of high gray value.	$\text{HGRE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}P(g,l)(g+1)^2$ (8.32)
Gray-level nonuniformity. This metric increases when gray-level outliers dominate the histogram.	$\text{GLNU} = \sum_{\text{l}=1}^{L}\left[\sum_{g=0}^{G-1}P(g,l)\right]^2$ (8.33)
Run-length nonuniformity. This metric increases when few run-length outliers dominate the histogram.	$\text{RLNU} = \sum_{g=0}^{G-1}\left[\sum_{l=1}^{L}P(g,l)\right]^2$ (8.34)
Run percentage. This metric provides information on the overall homogeneity of the histogram and is maximal when all runs are of unity length irrespective of the gray level.	$\text{RPC} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{1}{P(g,l)l}$ (8.35)
Short run, low gray-level emphasis. This is a diagonal metric that combines SRE and LGRE. The metric increases when the texture is dominated by many short runs of low gray value.	$\text{SRLGE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{P(g,l)}{l^2(g+1)^2}$ (8.36)
Long run, high gray-level emphasis. This is the complementary metric to SRLGE and increases with a combination of long, high-gray value runs.	$\text{LRHGE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}P(g,l)l^2(g+1)^2$ (8.37)
Short-run, high gray-level emphasis. This metric is orthogonal to SRLGE and LRHGE and increases when the texture is dominated by short runs with high intensity levels.	$\text{SRHGE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{P(g,l)(g+1)^2}{l^2}$ (8.38)
Long-run, low gray-level emphasis. Complementary to SRHGE, it increases when the texture is dominated by long runs that have low gray levels.	$\text{LRLGE} = \sum_{g=0}^{G-1}\sum_{l=1}^{L}\frac{P(g,l)l^2}{(g+1)^2}$ (8.39)

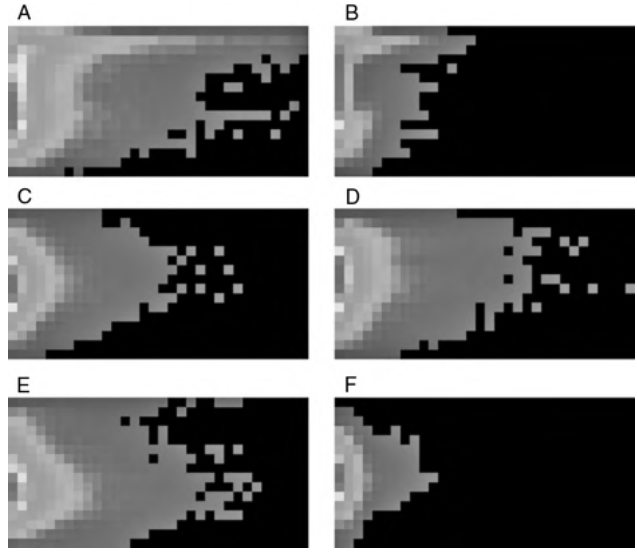


FIGURE 8.14 Run-length histograms of some of the textures in Figure 8.1: (A) corduroy at 0° scan angle, (B) corduroy at 90° scan angle, (C) glass, (D) carpet, (E) bark, and (F) knit, C–F at 0° scan angle. The histograms have been contrast-enhanced and false-colored for better visual representation. Black represents zero; blue low values; green, intermediate values; and red, high values. Each histogram shows increasing gray values (16 bins) from top to bottom and increasing run lengths (32 bins) from left to right. (*See insert for color representation of the figure*.)

in a striated texture (such as the corduroy texture), long runs will be found with a scan direction parallel to the striation, and short runs will dominate the scan direction perpendicular to the striation. In the corduroy example, a higher value for LRE can be expected in the 0° scan direction compared to the 90° scan direction.

Gray-scale run-length histograms of some of the textures in Figure 8.1 are shown in Figure 8.14, and some key run-length metrics are given in Table 8.6. It can be seen that the textures are dominated by short runs (large values for SRE). Increased gray-level bins, background flattening, and smoothing would increase the run lengths. The strong anisotropy of the corduroy texture can also be seen. In the vertical direction,

TABLE 8.6 Run-Length Metrics for the Run-Length Histograms Shown in Figure 8.14

Description	SRE	LRE	LGRE	HGRE	GLNU	RLNU
Corduroy						
0° scan direction	535	0.037	25.5	0.36	0.15	0.35
90° scan direction	3093	0.016	72.0	0.85	1.26	8.50
Glass	1184	0.015	11.2	0.60	0.47	1.30
Carpet	1030	0.017	13.3	0.50	0.35	1.00
Bark	703	0.020	7.11	0.61	0.27	0.54
Knit	2961	0.0091	14.6	1.19	1.86	7.76

short runs dominate and the vertical SRE is almost six times larger than the horizontal SRE, while the horizontal LRE is more than two times larger than the vertical LRE. In addition, the two nonuniformity metrics are much higher in the vertical than in the horizontal direction. The knit texture has values similar to those of corduroy in the vertical direction, but like all other textures, it is more isotropic than corduroy. Bark, carpet, and glass have intermediate values, with bark showing a higher tendency toward longer runs and more uniformity, while glass has shorter runs and more nonuniformity. Each of the rows of Table 8.6 can be seen as a six-dimensional feature vector (i.e., a signature that is specific for the texture).

A suggested algorithm to compute the gray-scale run-length histogram can be found in Algorithm 8.3. This algorithm example scans along horizontal scan lines,

```
set gmax=256;                    // Choose the number of discrete gray levels in image
set gbins=16;                    // number of gray bins
maxrl = xm;                      // maximum possible run length index; changes with direction
allocate RL(gbins,maxrl+1);      // Allocate RL output image (the actual 2D histogram)

for (y=0 while y<gbins increment y=y+1) do       // double loop over the output image
   for (x=0 while x<=maxrl increment x=x+1) do
      RL(x,y)=0;                 // Reset accumulators
   endfor;
endfor;
runcnt=0;                        // total number of runs (for normalization)

for (y=0 while y<ym increment y=y+1) do    // loop: start point for scan line
   boundary=1;                            // Discard boundary runs
   x=0; runlength=0;
   pbin = gbins*IM(x,y)/gmax;             // Initialize first run
   while (x<xm) do                        // horizontal run to right image edge
      gbin = gbins*IM(x,y)/gmax;          // this pixel's gray bin
      if (gbin=pbin) then                 // same gray level, same run
         runlength = runlength+1;         // Update length of this run
      else                                // change of gray level, must store run
         if (boundary=0) then             // Discard runs starting at image edge
            RL(pbin,runlength)=RL(pbin,runlength)+1;
            runcnt=runcnt+1;
         endif;
         boundary=0;
      endif;
      x=x+1;                              // next pixel in scan line
   endwhile;
endfor;
```

Algorithm 8.3 Run-length histogram for gray-scale images. This algorithm scans along horizontal scanlines. The input image `IM(x,y)` is a two-dimensional image of size `xm` and `ym` and with `gmax` gray levels. The algorithm produces a two-dimensional histogram `RL(g,l)`, where `g` corresponds to the gray level and `l` corresponds to the run length. The variable `runcnt` contains the total number of runs and can be used to normalize `RL` to conform to Equation (8.28). The algorithms for 45°, 90°, and 135° are very similar.

and each scan line starts at a pixel along the left image edge. To compute the run-length histograms along 45°, 90°, and 135° scan lines, similar functions need to be implemented that differ from Algorithm 8.3 in three aspects. First, the maximum possible run length must be assigned to `maxrl`. Second, the scan lines start at each pixel of the left and top image edges for 45°, at the top image edge for 90°, and at the top and right image edges for 135°. Third, the `while` loop needs to be adapted for each scan direction. The algorithm is designed to discard any runs that touch the image edges. Furthermore, the algorithm computes a variable, `runcnt`, that contains the total number of runs counted. By dividing each element of the output matrix `RL` by `runcnt`, the histogram is converted from counts to probabilities. After this normalization, `RL` conforms to Equation (8.28) and contains the probability of a specific run. This step is required to use `RL` in the computations of the metrics in Equations (9.29) through (9.39). The implementation of the summations in Equations (9.29) through (9.39) is straightforward by creating a double loop over all elements of `RL` and adding up the individual elements of `RL`, weighted with the appropriate factors and powers.

8.5. OTHER CLASSIFICATION METHODS

In Section 8.3, the loss of spatial information in the Fourier transform was discussed. It is possible to retain some spatial information by applying the windowed Fourier transform, that is, to subdivide the image into smaller square tiles and perform an individual Fourier transform on each of these tiles. When a Gaussian weighing function is applied to each of the tiles prior to the actual Fourier transform, the resulting filter becomes similar to a Gabor filter. More precisely, a Gabor filter with a filter function $h(x,y)$ is a complex sinusoid modulated by a Gaussian function as defined by

$$\begin{aligned} h(x,y) &= g(x,y)\ \exp\left[-2\pi j(Ux+Vy)\right] \\ g(x,y) &= \frac{i}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \end{aligned} \tag{8.40}$$

where U and V scale the sinusoids (U and V are also the center coordinates of the filter in the Fourier domain) and σ is the width of the filter. It can be shown that the frequency response $H(u,v)$ of the filter $h(x,y)$ is a real-valued, orientation-selective bandpass,

$$H(u,v) = \exp\left\{-2\pi^2\sigma^2\left[(u-U)^2+(v-V)^2\right]\right\} \tag{8.41}$$

with the pass frequency (U,V) and a bandwidth determined by σ. Equation (8.40) describes a symmetrical filter, and implementations with different bandwidths in the x and y directions are possible. With its three parameters, U, V and σ, a Gabor filter can be tuned to emphasize specific texture features and sizes when the image is convolved with the filter. For this reason, texture analysis is often performed with a Gabor

filter bank that contains multiple rotated and scaled Gabor filters.[39] Accordingly, the output of the filter bank contains multiple images, each image being the result of the convolution of the original image with one Gabor filter. For the purpose of texture segmentation, only the magnitude of the elements of the convolved images are considered, and each convolved image may again be convolved with a Gaussian blurring function to improve segmentation results.[41] The kth output image m_k (x,y) of the filter bank can therefore be described as

$$m_k(x,y) = g(x,y,\sigma_s) \otimes |h_k(x,y) \otimes I(x,y)| \tag{8.42}$$

where $I(x,y)$ is the input image, h_k (x,y) the kth Gabor filter, and $g(x,y,\sigma_s)$ a Gaussian function with standard deviation σ_s. The k pixels at the same spatial coordinate can be thought of as the feature vector of the corresponding pixel of the input image. The feature space can be further optimized by a suitable filter design.[9,41]

Gabor filters are particularly suitable for textures with periodic patterns, but are not preferred for random textures. Markov random field models are more suitable to analyze irregular or noisy images.[2] It is possible to synthesize (and therefore model) a texture by using Markov random fields.[11] A Markov random field is a graph where the nodes represent discrete variables (in this context, image values) and the edges are probability functions that determine the node values. A Markov random field evolves in discrete time steps, where a node value is modified from one step to the next, depending on the original node value (and in higher-order Markov models on earlier values), the value of its neighbors, and the probability functions. By adapting the probability functions, it is possible to "teach" a Markov random field to produce (i.e., model) a specific output image, such as a specific texture. Texture can therefore approximated by Markov random field models, and the differences between the models used to classify texture.[5]

A multiscale approach to texture analysis is possible by using the wavelet decomposition. In its simplest form, a n-element feature vector can be composed from the histogram energy at each decomposition level.[37] Alternatively, texture classification can be performed by neighborhood methods as described in Section 8.2, for example, by applying Laws' energy measures.[29] Starting at the lowest scale and moving to finer scales, more and more pixels can be classified building on the coarser scales. This approach was shown to lead to an unsupervised segmentation algorithm for images with multiple textures.[29] In a different approach, the wavelet decomposition was used to obtain statistical and co-occurrence signatures from several scales.[40]

Finally, color may be used to obtain additional texture information. Color is rarely used in medical imaging, because most modalities produce gray-scale images. Some examples where color information is available include light microscopy, fluorescent microscopy, and histology. Different color bands may carry different texture information, and additional feature vector components can be gained from analyzing the red, green, and blue components separately, by analyzing the color itself (e.g., by transforming the color image into hue-saturation-value space and examining the scalar hue component), or by converting the color image into a gray-scale image by using principal component analysis (Karhunen–Loève transform) and thus maximizing contrast.

8.6. BIOMEDICAL EXAMPLES

In medical images, texture is generated by the inhomogeneities of the tissue being imaged and by the imaging system itself. Most imaging systems (e.g., CT, MRI, ultrasound) are incapable of mapping the actual tissue microstructure onto the image. Rather, the tissue microstructure is first subjected to a contrast function and then convolved with the point-spread function of the system. The imaging system generally adds noise to the image, which may appear as pseudotexture that is generally unrelated to the underlying tissue. Finally, the imaging system may perform postprocessing steps that can have various effects on the image texture, including intensity remapping, blurring, or detail enhancement (see, e.g., Figure 10.25). Finally, depending on the imaging system, different scales may be imaged. A micro-CT device, for example, is capable of revealing details on a smaller scale than a clinical MR imaging device. The level of detail in ultrasound images depends strongly on the frequency and the type of probe used. In all cases, the image formation process has a fundamental influence on image texture. Therefore, a universal approach at texture analysis is not possible. Even within the same modality, feature vectors for the same tissue may be substantially different, depending on instrument settings and imaging conditions. An example of how first-order statistics and co-occurrence matrix features differ between modalities is given by Chung and Logeswaran.[8] In this study, images from the liver in healthy patients and patients with either fatty liver disease or a liver cyst were examined. Texture features were computed on the basis of first-order statistics and the co-occurrence matrix. Figure 8.15 demonstrates the differences between the modalities in one specific case, the correlation based on the co-occurrence matrix [Equation (8.13)]. Although the correlation is a normalized feature metric, both the

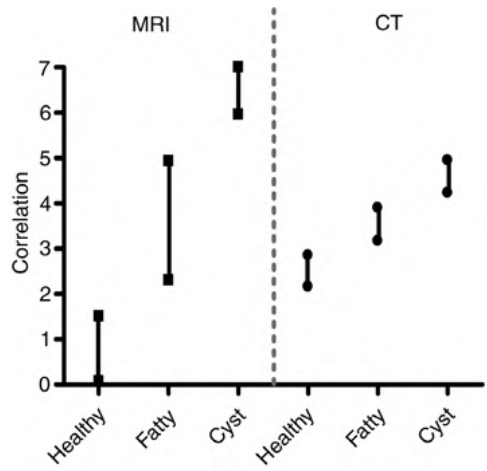


FIGURE 8.15 Intermodality comparison of healthy and diseased liver textures based on the correlation feature between CT and MRI[8]. Shown are the minimum to maximum ranges of 47–52 cases. (From ref. 8.)

ranges and the absolute values differ strongly between the modalities. It is notable, however, that there is no overlap between the values of healthy and diseased cases within one modality. In this special example, a single feature would be sufficient to differentiate the healthy cases from the two diseased cases.

A more comprehensive overview of the performance of different classification schemes in ultrasonic liver images was given by Wu et al.[42] In this example, ultrasound images from patients affected by liver cirrhosis and by a hepatoma were compared with healthy patients. Texture features based on the co-occurrence matrix and on Law's texture energies were used in this study. In addition, Wu et al. extended these concepts toward a multiresolution feature extraction which reveals self-similar properties. The actual feature classification was performed by using Bayesian classification. In this study, longer feature vectors showed better classification than shorter feature vectors, and the three-level multifractal feature scheme performed best in regard to classification. Similar techniques have been applied by Lupşor et al.[30] to validate the concept of a virtual liver biopsy, a noninvasive substitute for a liver biopsy by ultrasound imaging and image analysis. In this study, patients with different progression of fibrosis and cirrhosis as a consequence of hepatitis C were examined. Texture features were again computed from the co-occurrence matrix and first-order statistics, and additional features were computed from a special version of local gray-level box-counting[43] and a gray-level difference matrix. Overall, the feature vector contained 166 elements. The study demonstrated that in different stages of fibrosis and cirrhosis, different elements of the feature vector become relevant for differentiating between stages of the disease. The study emphasizes the need to closely analyze the contribution of the individual elements of the feature vector to differentiate between textures (and their associated disease level).

In an attempt to reduce the number of elements in the feature vector, Smutek et al.[36] examined the ability of 129 feature metrics and low-dimensional combinations of these metrics to distinguish between patients with a chronic inflammation of the thyroid gland and healthy patients on the basis of ultrasound images. Texture feature metrics were based on the gray-level co-occurrence matrix, and for classification, a minimum distance classifier was used. A training set of images was used to provide the spatial cluster information from healthy and diseased patients. After training, several regions were examined in the ultrasound images of the test image set, and a decision (healthy/diseased) was obtained for each region. The final decision was then based on a majority vote. Smutek et al. identified five feature metrics that were most relevant for differentiation between healthy and diseased cases.

These examples show the main shortcoming of texture analysis methods: namely, their relative inflexibility when used in closely related medical applications. A training set of images allows us to develop a texture analysis chain with good sensitivity and specificity for one specific disease, one specific modality, one specific instrument, and even one specific set of instrument parameters. Any deviation from the imaging modality or imaging parameters will invalidate the training. This shortcoming remains an unsolved problem. Although methods to extract texture features are well established, the texture classification is application-critical. For many applications, neural networks have been suggested to improve classification accuracy.

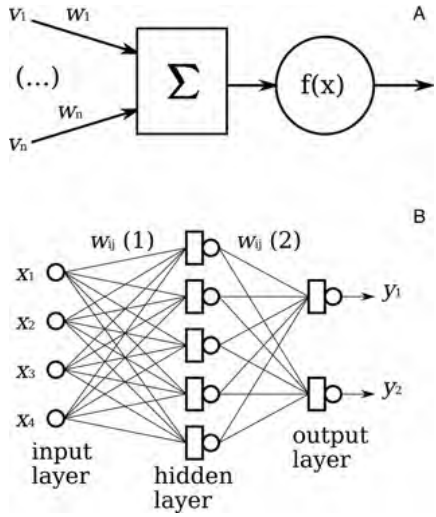


FIGURE 8.16 Artificial neural network. The basic building block is a simulated neuron (A). The neuron creates a weighted sum of its n input values v_i and modifies this sum with an activation function f. A neural network (B) is composed of several layers of neurons, whereby each output of one layer is connected with all neurons of the next layer. In this example, a neural network with one hidden layer is shown. In this case, the input values are fed into the hidden-layer neurons with the first set of weight factors, w_{ij} (1). The output of the hidden layer is fed into the output layer with the second set of weight factors, w_{ij} (2). In each of the square symbols, the summation takes place. Circles represent activation functions.

Figure 8.16 is a sketch of an artificial neural network. The neurons in neural networks are modeled by processing functions that map K input signals v_k onto one single internal output x by weighted summation,

$$x = f\left(\sum_{K=1}^{K} w_k v_k\right) \tag{8.43}$$

where w_k are individual weight factors and f is the neuron's activation function. Each neuron maintains its own weight vector w_k. A common activation function is given by

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8.44}$$

A different activation function is given by

$$f(x) = \begin{cases} 1 & \text{for} \quad x \geq T \\ 0 & \text{for} \quad x < T \end{cases} \tag{8.45}$$

which produces a binary output as opposed to the continuous output in Equation (8.44).

Neural networks most often have several layers. A vector of n input values is fed into the input layer of neurons. Its outputs are fed into a hidden layer of neurons with the first set of weight factors w_{ij} (1) as shown in Figure 8.16. The outputs of the hidden layer are finally fed into the output layer with the second set of weight factors w_{ij} (2). The outputs of the output layer are the values on which a decision is made. Any neural network must be trained. One commonly used training process is called *backpropagation*. For a training set of images, the desired output of the network is assumed to be known. The network's weight factors are initialized to some arbitrary value, and for each training image, the network output and its error E (i.e., the difference to the desired output) are computed. Variations of the weight factors of the output layer neurons cause variations in the error E, and the partial derivative of E toward the individual weight factors is used to update each weight factor.[35] After updating the output layer, the hidden layer is updated in a similar fashion. After the weight factors have been computed by backpropagation training, the network is expected to provide similar output values for similar input data. The popularity of neural networks lies in the fact that the weight factors do not need to have any physical meaning, and that arbitrary decisions can be trained into such a network. In one example,[19] abnormality detection in colonoscopic images was based on four co-occurrence matrix features that were analyzed by a neural network similar to the one described above. In the case of the study by Karkanis et al.,[19] the activation function in Equation (8.44) was used. In this specific example, the feature vector was processed by 16 input neurons (the first layer). The output of the first-layer neurons was fed into a hidden layer (second layer) with 21 neurons and processed in a similar manner. Finally, the outputs of the hidden layer were fed into an output layer consisting of two neurons. The decision (diseased versus healthy) was based on the signals from the two output neurons.

An alternative to neural networks for texture classification was proposed by Chen et al.[6] The study examines the malignancy of breast lesions in ultrasound images, and texture extraction was based on the normalized autocorrelation matrix of the image. The autocorrelation [Equation (8.25)] was computed for autocorrelation distances between 0 and 4, providing a total of 24 feature quantities, since the autocorrelation $\rho(0,0) = 1$. By training, reference vectors were obtained for each element of the map, and the weight factors of each map element, such as neurons [Equation (8.43)], were updated iteratively with the training images. Unlike a neural network, however, the self-organizing map determines the element (or neuron) with the smallest distance to the reference vector and uses this single neuron for the decision.

Neural networks are one classification method emerging from the area of artificial intelligence.[33] Another intensely studied artificial intelligence method is the genetic algorithm. Genetic algorithms simulate a population of classifiers, including inheritance of mutated genes to subsequent generations of classifiers, with simulated selection of the fittest individuals. In the context of texture classification, the genes of the population are mapping algorithms of texture feature metrics to diagnostic outcomes. In one example[31] a feature vector was extracted from shape and texture criteria of digitized x-ray mammograms. The texture features were based on Haralick's definitions. The simulated population maps the feature vector to a diagnostic outcome,

whereby each gene is a weight factor in the summation of the feature vector elements. By simulating breeding, mutation, cross-breeding, and selection of the fittest individuals, optimized genes (i.e., weight vectors) emerge. For the fitness function, a simulated individual's ability to predict the correct outcome can be used. Genetic algorithms have certain aspects in common with neural networks. The training period of a neural network corresponds to the genetic evolution (mutation and selection) of the genetic algorithm. In both cases, the algorithm extrapolates from the training set of images to the test set, and it is expected that similar textures cause a similar response or prediction. Finally, in both cases, the individual weight factors (in both the neurons and the genes) do not necessarily represent any meaningful values that could be related to tangible texture properties.

A different approach to classifying texture was proposed by Petrosian et al.[34] for digitized x-ray images of potentially malign breast tissue. Once again, features were extracted from the co-occurrence matrix. The three features evaluated (sum average, correlation, and energy) were used as input for a decision tree. In a three-value decision tree, the three input features, F1, F2, and F3 and several threshold values, T1, T2a and T2b, T3a through T3d (Figure 8.17) were used to obtain decision outcomes. There are eight possible outcomes (decisions), D1 through D8, to be mapped to the diagnosis (malignant or normal).

Similar to neural networks, a training process is required to map decisions D1 through D8 to a diagnosis and to determine suitable thresholds. Training was performed by searching the entire threshold space for the optimum combination of thresholds. Although this method appears to be less efficient than training of a neural network, its one advantage over neural networks is that supervised training can be applied; that is, threshold values may be selected according to prior knowledge and may relate to texture features in an intuitive, meaningful way.

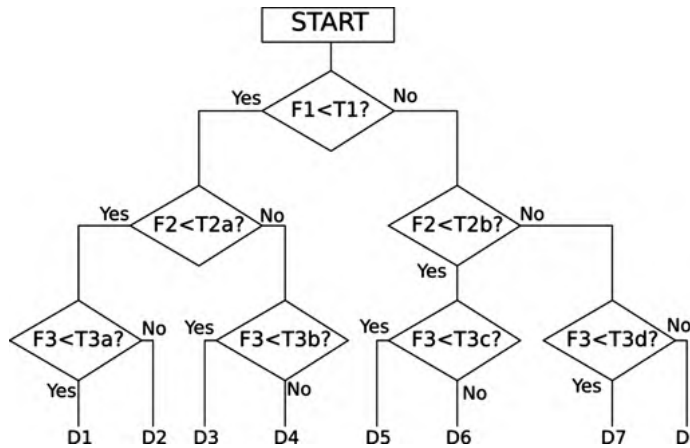


FIGURE 8.17 Three-value decision tree. The features F1, F2, and F3 are compared with seven thresholds T1 through T3d, depending on the path through the tree. The eight possible outcomes need to be mapped to the diagnosis, and the thresholds need to be determined ("trained") for optimum specificity and sensitivity.

Yet another classification approach was used by Pereira et al.[32] to determine the feasibility to distinguish between malign masses, microcalcifications, and healthy cases of digitized x-ray mammograms. Texture analysis was performed by applying the Haralick operators of the co-occurrence matrix. In addition, the energies E_L of the second and third decomposition level of the wavelet transform as defined by

$$E_L = \frac{1}{N_L} \sum_{i=0}^{N_L-1} (\gamma_{i,L})^2 \tag{8.46}$$

where $\gamma_{i,L}$ are the wavelet coefficients of the Lth decomposition level and N_L is the number of wavelet coefficients at the Lth level, were used as texture feature metrics. Classification was performed by using the k-nearest-neighbor (k-NN) method. The k-NN method is a training-based clustering method. Training images are used to create clusters of points (the endpoints of the vectors) in feature space. For each test image, the Euclidean distance to the k nearest neighbors from the training set is calculated. The unknown sample is assigned to the class (i.e., a training set cluster), which contributes the most neighbors to the set of k nearest neighbors.

The influence of the imaging process and imaging instrumentation on the texture was discussed. Several studies focus on examining this influence on texture classification. Kuo et al.[23] used two ultrasound scanners to obtain B-mode images of the breast. A decision tree was built from three co-occurrence matrix features: contrast, co-variance, and dissimilarity. However, Kuo et al. implemented these features in a different manner, obtaining the quantities directly from the image without computing the co-occurrence matrix. In the special case of these three feature metrics, the direct implementation is computationally very efficient. For classification, a decision tree was used. One important conclusion of this study was the necessity to transform the image data to match the resolution of different ultrasound scanners. A similar question was examined in a MRI study by Collewet et al.[10] An earlier multicenter study in which foam samples and MRI[27] were used showed that texture features are not comparable between different scanners. Before extracting texture feature metrics from MR images of cheese samples, Collewet et al. normalized the MR image intensities to either have a matching maximum or mean value (multiplicative intensity scaling), or have a matching histogram by performing a histogram transform. Various texture features, including Haralick's parameters, were extracted, and classification was performed by using the nearest-neighbor method (k-NN with $k = 1$). Classification results were improved with the histogram normalization, and classification became less dependent on MR acquisition protocols.

What is the physical meaning of texture feature metrics? Few studies exist that relate image texture to actual histology. Huber et al.[17] compared metrics derived from the co-occurrence matrix to both the visual appearance of the ultrasound B-mode scan and to histology sections of breast tumor tissue and found that the visual perception of the images corresponded with some features, such as mean gradient and contrast. The dissimilarity metric was identified by Chen et al.[7] to correlate well with the medical diagnosis in B-mode scans of cancerous tissue: stellate lesions,

malignant tissue mixed with fibrous and cellular parts, and circumscribed carcinomas were associated with low, intermediate, and high dissimilarity values. The authors conclude that the cellular and fibrous content of the cancerous region, in combination with the spatial distribution of breast masses, determine the value of the dissimilarity feature. In a later study, the same research group found a strong negative correlation ($r^2 = 0.88$) of the dissimilarity measure with the percentage of fibrosis, and a similarly strong but positive correlation ($r^2 = 0.86$) of the same measure with the percent cellular content in histology.

Up to this point, the co-occurrence matrix was predominantly used in the examples to extract feature values. Other methods, including Fourier-domain methods and the run-length method, enjoy considerable popularity. Fourier-domain analysis of trabecular bone texture is the subject of a study by Gregory et al.[14] Figure 8.18 shows magnified and contrast-enhanced trabecular texture in a projection x-ray image of the femur and its corresponding Fourier transform.

The preferred orientation of the trabeculae can be extracted from the Fourier transform by various means. The magnitude ratio between a low and a high spatial frequency can be computed for various angles, and the angle where the maximum ratio occurs is the direction perpendicular to the preferred orientation angle. Figure 8.19 shows profiles of the magnitude of the Fourier transform parallel and perpendicular to the white dashed line in Figure 8.18B. The corresponding ratios for spatial frequencies of 5 per pixel and 25 per pixel are 7.6 for the parallel direction and 19.5 for the perpendicular direction, indicating a markedly steeper drop-off of the magnitude along the white dashed line (and therefore perpendicular to the preferred trabecular orientation). If self-similar properties are suspected, a rose plot (see Section 10.4) can provide the anisotropy and preferred direction. Gregory et al.[14] used several methods to extract features from three preferred profiles: the parallel profile, the perpendicular profile, and the average profile averaged over all circles of equal spatial frequency. Principal components analysis and fractal analysis provided quantitative measures.

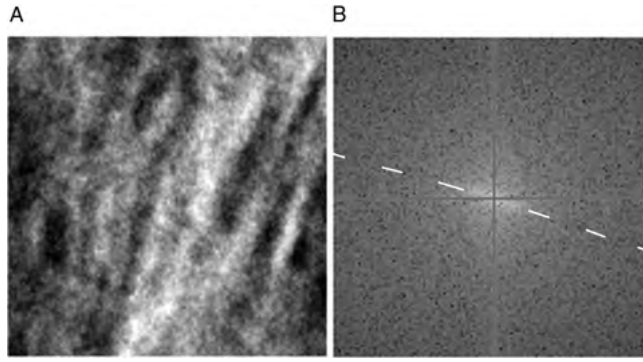


FIGURE 8.18 Magnified section of an x-ray projection image of the femur showing the texture of the trabeculae (A). The logarithmic magnitude of the Fourier transform (B) reveals a preferred orientation about 22° from the horizontal direction, as shown by the dashed white line. (From ref. 14, permission granted through the Creative Commons License.)

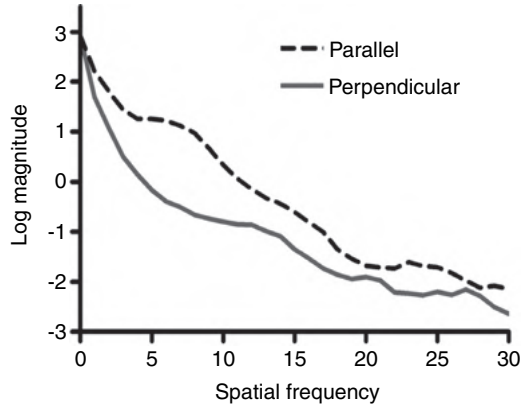


FIGURE 8.19 Profiles of the log-transformed magnitude of the Fourier transform (Figure 8.18B) parallel and perpendicular to the white dashed line.

They found that values obtained by principal component analysis from the perpendicular profile provided the best predictive value for patients with and without hip fractures.

Brown and Frayne[3] present several applications of the S-transform in texture analysis. The S-transform is a localized Fourier transform, where a Gaussian multiplicative window is applied over each single pixel of the image, and a separate Fourier transform is performed for each pixel, providing a complete two-dimensional spectrum for each pixel. This spectrum can be reduced to a one-dimensional spectrum by averaging the frequencies at all angles. The dot product with a reference spectrum can provide a similarity metric. In some biomedical examples, Brown and Frayne showed the ability of the S-transform technique to differentiate between tumor types in MR images and the trabecular pattern in CT images of bone from normal and osteoporotic patients.

The run-length method has achieved considerable popularity in the analysis of cancellous bone and the diagnosis of osteoporosis. In radiography images, CT images and even MR images, it is possible to separate bone structure from other tissue components by its density. Images can therefore easily be segmented into bone and nonbone pixels, and a binary run-length analysis can be applied. If a clear segmentation of the trabecular area is possible in high-resolution images, geometric features are typically computed, such as the mean trabecular width or the mean intratrabecular space. For image modalities with a lower resolution or for projection x-ray imaging, a similar analysis can be performed on gray-level images. A recent study by Lespessailles et al.[28] highlights the usability of texture parameters obtained from the co-occurrence matrix, through fractal analysis, and through run-length analysis in the diagnosis of osteoporosis. This study focused on relatively few parameters: the short-run emphasis [Equation (8.29)], the co-occurrence energy [Equation (8.9)], and a fractional Brownian motion model to obtain the Hurst exponent H (Section 10.1). While the focus of the study by Lespessailles et al.[28] was on assessing the

reproducibility of the texture metrics, other studies found a strong correlation of run-length parameters with histomorphometric values[4,12] and a good predictive value with high specificity for fracture risk.[18]

An interesting extension of the idea of co-occurrence matrices was presented by Kovalev et al.[21] To classify very subtle texture variations in MR images of the human brain, any voxel of the three-dimensional image was considered to be carrying more information than its gray-level intensity. Kovalev et al. proposed to use intensity, gradient magnitude, and gradient orientation as orthogonal elements of information carried by each voxel. Consequently, the co-occurrence matrix becomes six-dimensional. Kovalev et al. have shown the ability of the extended texture analysis method to differentiate between patients with Alzheimer's disease and normal controls,[21] the ability to segment diffuse brain lesions,[21] and to quantify brain asymmetry in subjects of different ages and genders.[22]

As computers become more and more powerful, more sophisticated methods for texture analysis emerge. In fact, texture transformations, such as the computation of the co-occurrence matrix, can amplify subtle texture differences that even the human eye would find difficult to detect. Texture analysis has become a key step in the quantitative and unsupervised analysis of biomedical images.

REFERENCES

1. Anonymous. Texture database. http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/. Accessed Nov. 8, 2008.
2. Besag J. Statistical analysis of dirty pictures. J Appl Stat 1993; 20(5): 63–87.
3. Brown RA, Frayne R. A comparison of texture quantification techniques based on the Fourier and S transforms. Med Phys 2008; 35(11): 4998–5008.
4. Chappard D, Guggenbuhl P, Legrand E, Basle MF, Audran M. Texture analysis of x-ray radiographs is correlated with bone histomorphometry. J Bone Miner Metab 2005; 23(1): 24–29.
5. Chellappa R, Chatterjee S. Classification of textures using Gaussian Markov random fields. IEEE Trans Acoust Speech Signal Process 1985; 33: 959–963.
6. Chen DR, Chang RF, Huang YL. Breast cancer diagnosis using self-organizing map for sonography. Ultrasound Med Biol 2000; 26: 405–411.
7. Chen SJ, Cheng KS, Dai YC, Sun YN, Chen YT, Chang KY, Yu SN, Chang TW, Tsai HM, Hsien CC. Quantitatively characterizing the textural features of sonographic images for breast cancer with histopathologic correlation. J Ultrasound Med 2005; 24(5): 651–661.
8. Chung SH, Logeswaran R. Evaluation of texture analysis techniques for characterization of multimode-based liver images using SGLCM and FOS. Biomed 06, IFMBE Proc 2007; 15: 262–266.
9. Clausi DA, Jernigan ME. Designing Gabor filters for optimal texture separability. Pattern Recogn 2000; 33(11): 1835–1849.
10. Collewet G, Strzelecki M, Mariette F. Influence of MRI acquisition protocols and image intensity normalization methods on texture classification. Magn Reson Imaging 2004; 22(1): 81–91.

11. Cross GC, Jain AK. Markov random field texture models. IEEE Trans Pattern Anal Mach Intell 1983; 5: 25–39.
12. Durand EP, Ruegsegger P. Cancellous bone structure: analysis of high-resolution CT images with the run-length method. J Comput Assist Tomogr 1991; 15(1): 133–139.
13. Galloway MM. Texture analysis using gray level run lengths. Comput Graph Image Process 1975; 4: 172–179.
14. Gregory JS, Stewart A, Undrill PE, Reid DM, Aspden RM. Identification of hip fracture patients from radiographs using Fourier analysis of the trabecular structure: a cross-sectional study. BMC Med Imaging 2004; 4(1):4.
15. Haralick RM. Statistical and structural approaches to texture. Proc IEEE 1979; 67(5): 786–804.
16. Haralick RM, Shanmugam K, Dinstein I. Textural features for image classification. IEEE Trans Syst Man Cybern 1973; 3(6): 610–621.
17. Huber S, Medl M, Vesely M, Czembirek H, Zuna I, Delorme S. Ultrasonographic tissue characterization in monitoring tumor response to neoadjuvant chemotherapy in locally advanced breast cancer (work in progress). J Ultrasound Med 2000; 19(10): 677–686.
18. Ito M, Ohki M, Hayashi K, Yamada M, Uetani M, Nakamura T. Trabecular texture analysis of CT images in the relationship with spinal fracture. Radiology 1995; 194(1): 55–59.
19. Karkanis SA, Magoulas GD, Grigoriadou M, Schurr M. Detecting abnormalities in colonoscopic images by textural description and neural networks. Advance Course in Artificial Intelligence (ACAI) Proceedings, Workshop on Machine Learning in Medical Applications. 1999; 59–62.
20. Knill DC, Field D, Kersten D. Human discrimination of fractal images. J Opt Soc Am A 1990; 7(6): 1113–1123.
21. Kovalev VA, Kruggel F, Gertz HJ, von Cramon DY. Three-dimensional texture analysis of MRI brain datasets. IEEE Trans Med Imaging 2001; 20(5): 424–433.
22. Kovalev VA, Kruggel F, von Cramon DY. Gender and age effects in structural brain asymmetry as measured by MRI texture analysis. Neuroimage 2003; 19(3): 895–905.
23. Kuo WJ, Chang RF, Moon WK, Lee CC, Chen DR. Computer-aided diagnosis of breast tumors with different US systems. Acad Radiol 2002; 9(7): 793–799.
24. Kurita T, Otsu N. Texture classification by higher order local autocorrelation features. Proc Asian Conf Comput Vis 1993; 175–178.
25. Laws KI. Texture energy measures. Proc DARPA Image Underst Workshop 1979; 47–51.
26. Lazebnik S, Schmid C, Ponce J. A sparse texture representation using local affine regions. IEEE Trans Pattern Anal Mach Intell 2005; 27(8): 1265–1278.
27. Lerski RA, Schad LR, Luypaert R, Amorison A, Muller RN, Mascaro L, Ring P, Spisni A, Zhu X, Bruno A. Multicentre magnetic resonance texture analysis trial using reticulated foam test objects. Magn Reson Imaging 1999; 17(7): 1025–1031.
28. Lespessailles E, Gadois C, Lemineur G, Do-Huu JP, Benhamou L. Bone texture analysis on direct digital radiographic images: precision study and relationship with bone mineral density at the os calcis. Calcif Tissue Int 2007; 80(2): 97–102.
29. Lu CS, Chung PC, Chen CF. Unsupervised texture segmentation via wavelet transform. Pattern Recogn 1997; 30(5): 729–742.
30. Lupşor M, Badea R, Nedevschi S, Vicas C, Tripon S, Stefánescu H, Radu C, Grigorescu M. The assessment of liver fibrosis using the computerized analysis of ultrasonic images.

Is the virtual biopsy appearing as an option? 1st Int Conf Adv Med Health Case Through Technol (MediTech) 2007; 245–250.

31. Mu T, Nandi AK, Rangayyan RM. Classification of breast masses using selected shape, edge-sharpness, and texture features with linear and kernel-based classifiers. J Digit Imaging 2008; 21(2): 153–169.
32. Pereira RR Jr, Azevedo Marques PM, Honda MO, Kinoshita SK, Engelmann R, Muramatsu C, Doi K. Usefulness of texture analysis for computerized classification of breast lesions on mammograms. J Digit Imaging 2007; 20(3): 248–255.
33. Perry SW, Wong H-S, Guan L. Adaptive Image Processing: A Computational Intelligence Perspective. Boca Raton, FL: CRC Press, 2002.
34. Petrosian A, Chan HP, Helvie MA, Goodsitt MM, Adler DD. Computer-aided diagnosis in mammography: classification of mass and normal tissue by texture analysis. Phys Med Biol 1994; 39: 2273.
35. Rojas R. Neural Network: A Systematic Introduction. Berlin: Springer-Verlag, 1996.
36. Smutek D, Sara R, Sucharda P, Tjahjadi T, Svec M. Image texture analysis of sonograms in chronic inflammations of thyroid gland. Ultrasound Med Biol 2003; 29(11): 1531–1543.
37. Szczypiñski P, Kociolek M, Materka A, Strzelecki M. Computer program for image texture analysis in PhD students laboratory. Int Conf Signals Electron Syst, Poland 2001; 255–262.
38. Toyoda T, Hasegawa O. Texture classification using higher-order local autocorrelation features. Texture 2005: Proc 4th Int Workshop texture Anal Synth 2005; 131–136.
39. Turner MR. Texture discrimination by Gabor functions. Bio Cybern 1986; 55(2): 71–82.
40. Van de Wouwer G, Scheunders P, Van Dyck D. Statistical texture characterization from discrete wavelet representations. IEEE Trans Image Process 1999; 8(4): 592–598.
41. Weldon TP, Higgins WE, Dunn DF. Efficient Gabor filter design for texture segmentation. Pattern Recogn Lett 1996; 29(12): 2005–2015.
42. Wu CM, Chen YC, Hsieh KS. Texture features for classification of ultrasonic liver images. IEEE Trans Med Imaging 1992; 11(2): 141–152.
43. Xia Y, Feng D, Zhao R. Morphology-based multifractal estimation for texture segmentation. IEEE Trans Image Process 2006; 15(3): 614–623.

9

SHAPE ANALYSIS

Shape analysis of features and texture analysis of features are related operations in image analysis. They allow us to distinguish between classes of features. Shape analysis, however, assumes a binary image, that is, an image partitioned into pixels that belong to a feature and pixels that do not belong to the feature. The goal of shape analysis, similar to texture analysis, is to obtain one or more quantitative metrics that characterize the shape.

Figure 9.1 shows an example where shape analysis and shape-based classification plays an important role. Normal red blood cells show a round or elliptical shape in the microscope image of a blood smear. Some diseases, such as sickle cell disease, affect the shape. Figure 9.1 shows a collage of red blood cells that are predominantly normal, but with some deformed cells. Each deformed cell type (spiculated, half-moon, sickle) occurs twice in the image with a different size and a different rotation. With few exceptions, shape analysis is expected to provide translation-, scaling-, and rotation-invariant metrics. A spiculated cell, for example, should be identified as such irrespective of its size, its position in the image, and its rotation. Consider, for example, Figure 9.2, which shows three different shapes taken from Figure 9.1B rotated and scaled. A human observer would place the shapes in each row in the same class: from top to bottom, round, spiculated, and half-moon, growing in size and rotated clockwise from left to right. Also note the extra protrusion in the leftmost spiculated shape, which does not change the overall visual impression. A computerized shape classifier would typically be expected to provide similar values for each row of shapes.

Advanced Biomedical Image Analysis, By Mark A. Haidekker

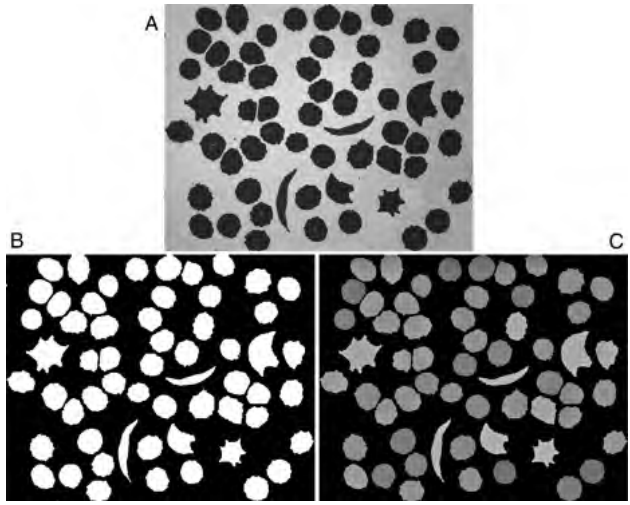


FIGURE 9.1 Example of shape analysis and classification. Image A shows various red blood cells (synthetically generated image) with three types of deformation. Each deformed cell occurs twice in the image, with a different size and rotation. Image B is the segmented (binarized) image, obtained through thresholding, and image C was obtained by quantifying the irregularity of the shape and using a false-color representation of the shape metric. (*See insert for color representation of the figure*.)

In many cases, several segmented features exist in a single image. A single feature is a contiguous region of connected nonbackground pixels. It is possible to distinguish between 4- and 8-connected neighborhoods. In a four-connected neighborhood, only pixels touching a neighboring pixel at 0°, 90°, 180°, or 270° are considered connected. In an 8-neighborhood, the diagonal neighbors are also considered connected.

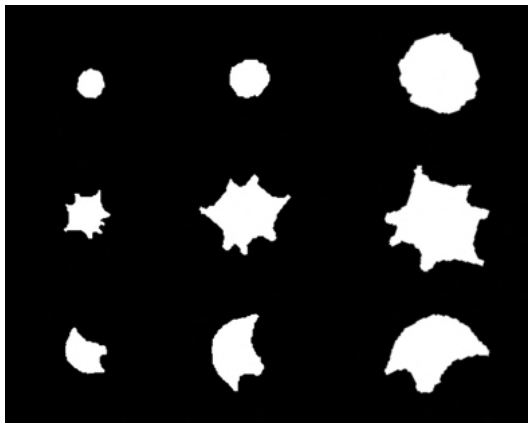


FIGURE 9.2 Three different shapes, magnified and rotated from left to right. Intuitively, the shapes in each row would be considered the same.

In a practical application, however, the definition of connectedness should not play a key role, because the difference between connected and nonconnected features could too easily depend on a single noise pixel. In fact, suitable preprocessing is important in obtaining clear shape representation and separation of neighboring shapes. Preprocessing steps would include filtering to suppress noise, suitable segmentation steps, and additional filtering steps (such as morphological operators) after segmentation.

9.1. CLUSTER LABELING

Most image processing software contain functions to distinguish different features in an image. These are often referred to as *feature labeling*, *particle analysis*, or *cluster analysis*. The most widely applied and most efficient approach to label image features is a two-pass process. The input image to this process is assumed to be strictly binary, with the value 1 representing feature pixels and the value 0 representing background pixels. A counter exists that starts at a value of 2. In the first pass, the image is scanned from left to right and from top to bottom: If a nonzero pixel has a nonzero neighbor to the left or above, it assumes the value of that neighbor; if the scan runs across a nonzero pixel with background pixels to the left and above, it assumes the value of the counter, and the counter is incremented. After the first pass finishes, all features are now assigned different image values. However, some features may contain more than one value (e.g., in U-shaped features). For this reason, a second pass is required in which features that contain connected regions with different image values are renumbered until each feature contains only pixels with one unique value. The process of cluster labeling is illustrated in Figure 9.3. The example is a U-shaped feature, and at the end of the first pass, connected subclusters with more than one value exist. These cases where one feature contains ambiguous numbering need to be resolved in a separate pass.

An alternative and potentially slightly less efficient algorithm can be devised that labels the clusters in one pass. This algorithm scans the binary input image from left to right and from top to bottom, similarly with a counter that starts at a value of 2. Each time the scan encounters a pixel with the value of 1, a region-growing process is initiated, and the fully grown region is filled with the value of the counter. The counter is then incremented. The advantage of the second algorithm lies in its more straightforward implementation. The process of feature labeling and the extraction of three sample shape descriptors (aspect ratio, compactness, and irregularity, as defined in Section 9.2) for each feature is outlined in Algorithms 9.1 and 9.2. Algorithm 9.2 builds on Algorithm 9.1. The input image needs to be thresholded and is assumed to be strictly binary; that is, all background pixels have a pixel value of zero and all feature pixels have a value of 1. Algorithm 9.2 detects unlabeled features, whereas Algorithm 9.1 does the actual labeling of an individual feature by region growing and the determination of its shape descriptors. In the process of region growing, each feature's centroid and boundary can be computed. The centroid is the average of all feature pixel coordinates, and a boundary pixel is any pixel that touches a background pixel. At the end of Algorithm 9.2, the tables for irregularity, compactness, aspect

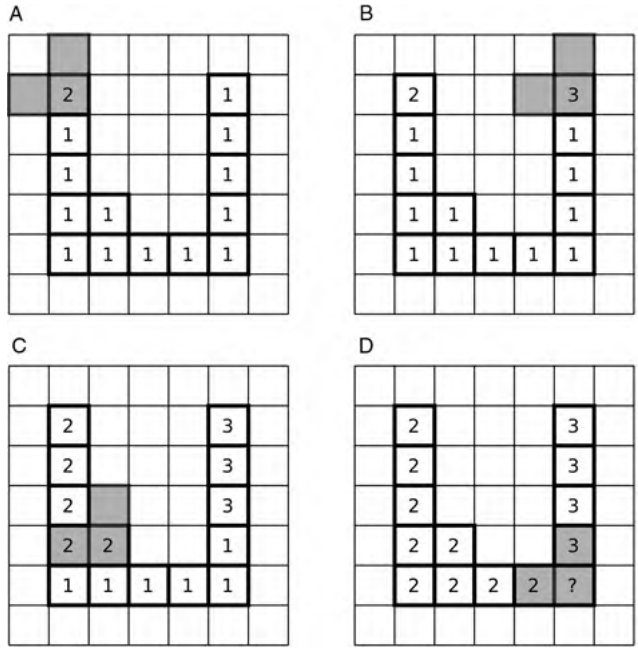


FIGURE 9.3 Cluster labeling with the two-pass method. The image is scanned from left to right and from top to bottom with an inverted-L shape (pixels marked gray). If the corner pixel of the inverted L reaches a new feature (indicated by the value 1), the algorithm begins a new cluster and assigns to it a cluster number greater than 1 (A). The algorithm cannot see that the pixel that was discovered in part B belongs to the same cluster, and both subgroups of connected pixels are treated as separate clusters (C). This leads to an ambiguity in part D. The ambiguity can be resolved by assigning one of the two possible values and merging connecting subclusters in a separate pass.

ratio, and size can be used as lookup tables, or to relabel the clusters according to their features. This algorithm was used to generate Figure 9.1C.

9.2. SPATIAL-DOMAIN SHAPE METRICS

Widely used metrics that describe the shape of a feature are the aspect ratio, the compactness, and the irregularity. All three metrics use the boundary points. Let us assume that a shape has K boundary points and define the distance of the kth boundary point at an angle φ from the x-axis as $r_k(\varphi)$:

$$r_k(\varphi) = \sqrt{(x_k - x_c)^2 + (y_k - y_c)^2} \tag{9.1}$$

where the associated angle φ follows

$$\tan \varphi = \frac{y_k - y_c}{x_k - x_c} \tag{9.2}$$

```
IM(xs,ys)=val;                    // Mark the seed pixel
sx=0; sy=0; area=0;               // to compute the centroid

repeat
   labeled=0;                     // number of pixels relabeled per iteration
   for (y=1 while y<ymax-1 increment y=y+1)
      for (x=1 while x<xmax-1 increment x=x+1)
         if (IM(x,y)=1) then             // Examine unlabeled foreground pixel
            if (IM(x+1,y)=val OR IM(x-1,y)=val
               OR IM(x,y+1)=val OR IM(x,y-1)=val
               OR IM(x-1,y-1)=val) OR IM(x-1,y+1)=val
               OR IM(x+1,y-1)=val) OR IM(x+1,y+1)=val) then
                  IM(x,y)=val;           // Relabel 8-connected pixel
                  labeled=labeled+1;
                  area=area+1;
                  sx=sx+x; sy=sy+y;             // for the centroid
            endif;
         endif;
      endfor;
   endfor;
until (labeled=0);                // Region-growing ends if no more pixels relabeled
xc=sx/area; yc=sy/area;           // the centroid

rmin=xmax+ymax; rmax=0;           // to determine minimum and maximum radius
sx=0; sxx=0;
for (y=1 while y<ymax-1 increment y=y+1)
   for (x=1 while x<xmax-1 increment x=x+1)
      if (IM(x,y)=val) then          // Examine feature pixel
         n=n+1;                      // total number of pixels for this feature
         if (IM(x+1,y)=0 OR IM(x-1,y)=0              // boundary pixel?
            OR IM(x,y+1)=0 OR IM(x,y-1)=0
            OR IM(x-1,y-1)=0) OR IM(x-1,y+1)=0
            OR IM(x+1,y-1)=0) OR IM(x+1,y+1)=0) then
               C = C+1;              // Count boundary pixels
               r = sqrt((x-sx)^2+(y-sy)^2);   // distance from centroid
               if r>rmax then rmax=r;         // Keep track of maximum radius...
               if (r<rmin) then rmin=r;       // ... and minimum radius
               sx=sx+r; sxx=sxx+r*r;           // for the irregularity
         endif;
      endif;
   endfor;
endfor;
A = rmax/rmin; V=sqrt(sxx-sx*sx/C)/sx; C=C*C/area;
```

Algorithm 9.1 Cluster region growing. This algorithm fills a contiguous region in the input image `IM(x,y)` with the value `val`. The seed points is given by `xs` and `ys` and must be part of the feature. The input image is assumed to be strictly binary, with values of 0 for background and 1 for the features. At the end of this algorithm, one cluster has been relabeled to `val`, and the variables `A`, `C`, and `V` contain the shape descriptors of aspect ratio, compactness, and irregularity, respectively.

```
allocate aspect();                                  // Allocate feature metric tables
allocate compact();
allocate irreg();
allocate size();
allocate centx(); allocate centy();
val=1;                                              //Initialize label value

for (y=1 while y<ymax-1 increment y=y+1)
   for (x=1 while x<xmax-1 increment x=x+1)
      if (IM(x,y)=1) then              // Examine unlabeled foreground pixels only
         xs=x; ys=y; val=val+1;        // seed pixel and label value
         label_region;                 // Region-grow this feature (Algorithm 9.1)
         centx(val)=xc; centy(val)=yc;       // Store feature data for this cluster
         aspect(val)=A;
         compact(val)=C;
         irreg(val)=V;
         size(val)=area;
      endif;
   endfor;
endfor;
```

Algorithm 9.2 Cluster labeling. This algorithm builds on Algorithm 9.1, here referred to as `label_region`, to relabel contiguous areas. The input image `IM(x,y)` of size `xmax` and `ymax` is assumed to be strictly binary. In the process of labeling the clusters, tables are generated that contain centroid and shape information. At the end of this algorithm, each contiguous region (i.e., feature) is labeled with the same pixel value. At the end of this algorithm, each feature has a unique pixel value.

This representation allows us to unroll the perimeter and convert it into a one-dimensional function $r(\varphi)$ or, alternatively, into a sequence r_k, $0 \leq r < K$. An example is given in Figure 9.4, where the segmented shape (Figure 9.4A) was unrolled beginning at its topmost point and turning counterclockwise. The unrolled perimeter is shown in Figure 9.4C.

The aspect ratio A is defined as the ratio of the radii of the circumscribed to the inscribed circle, which corresponds to the ratio of the largest to the smallest r_k:

$$A = \frac{\max r_k}{\min r_k} \tag{9.3}$$

The aspect ratio quantifies the eccentricity of a shape. A circle has an aspect ratio of 1. If the shape is more oval or elliptical, or has protrusions, A increases. The aspect ratio is translation-, rotation-, and scaling-invariant within the limits of the discrete pixel resolution of the image. Since A is based on only two salient points, the maximum and the minimum of the sequence of r_k, the aspect ratio does not provide detailed information about the shape. The teardrop-shaped red blood cell and the sickle cell

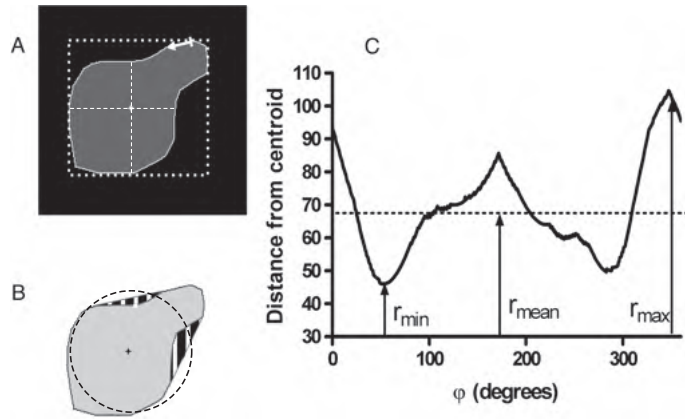


FIGURE 9.4 Sample cluster to illustrate some spatial-domain shape metrics. The cluster is shown in part A after a typical segmentation process with zero-valued background (black) and highlighted perimeter (white). The thick dashed line indicates the bounding box, and the diagonal dashed lines are the long and short axes of an ellipse that approximates the feature. The centroid is indicated at the intersection of the long and short axes. In part B, a circle with radius r_{mean} is displayed. The circle has four intersections with the shape. Furthermore, concave regions have been filled (striped regions) to obtain the convex hull. Part C shows the unrolled radius, with the minimum, mean, and maximum distance of the perimeter from the centroid.

may have the same aspect ratio, yet they belong to two fundamentally different shape classes.

The compactness C of a shape is defined as the squared perimeter length normalized by the area and can be approximated by

$$C = \frac{1}{N}\left[\sum_{k=0}^{K-1}\sqrt{(x_{k+1}-x_k)^2+(y_{k+1}-y_k)^2}\right]^2 \tag{9.4}$$

where N is the number of pixels that belong to the feature (i.e., the area), x_k and y_k are the pixel coordinates of the boundary points, and $x_K = x_0$, $y_K = y_0$ to provide a closed contour. The most compact shape possible is the circle with a theoretical value of $C = 4\pi \approx 12.6$. Since C cannot be smaller than approximately 12.6 under the definition of Equation (9.4), different definitions of compactness can be found, for example,

$$C = \frac{\sqrt{4\pi N}}{\sum_{k=0}^{K-1}\sqrt{(x_{k+1}-x_k)^2+(y_{k+1}-y_k)^2}} \tag{9.5}$$

This alternative value is normalized to 1 for an ideal circular shape and, being the inverse of the definition in Equation (9.4), tends toward zero for irregular shapes. Due to the discrete nature of images, small circles can deviate from the theoretical

value. The compactness is a metric of the deviation from a circular shape, and C [as defined in Equation (9.4)] increases both with eccentricity of the shape and with the irregularity of its outline. In other words, values of C may overlap between highly elliptical shapes with a regular outline and round shapes with an irregular outline. The irregularity of the outline can better be quantified by the coefficient of variation V of the r_k, which can be efficiently approximated with

$$V = \frac{\sqrt{\sum_{k=0}^{K-1} r_k^2 - (1/K)\left(\sum_{k=0}^{K-1} r_k\right)^2}}{\sum_{k=0}^{K-1} r_k} \tag{9.6}$$

Aspect ratio, compactness, and irregularity are shape metrics that are rotation-, scale-, and translation-invariant within the limits of pixel discretization. Each of the metrics, however, can assume similar values for different shapes as demonstrated, for example, in Figure 9.1, where the half-moon and the spicular cells have approximately the same value for irregularity. A multidimensional feature vector may be used to better separate groups of features with a similar shape. An example is given in Figure 9.5, where a two-dimensional feature vector is composed for each shape with the irregularity and compactness as elements. All normal (round) red blood cells can be found inside the box at the lower left corner with $C \leq 13$ and $V \leq 26$. The distance between the cluster centers of the spicularly shaped and half-moon-shaped cells has been increased in two-dimensional space. Assignment of individual shapes to feature classes may be performed automatically by clustering methods such as k-means clustering and fuzzy c-means clustering (Section 2.5).

From the ordered set of boundary points $r(\varphi)$, additional shape descriptors can be derived, but they are to some extent related to compactness, aspect ratio, and irregularity. By fitting an ellipse function into the boundary, the orthogonal long

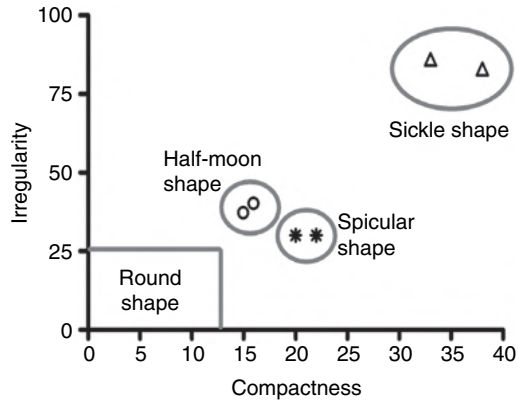


FIGURE 9.5 Separation of features with different shapes by combining two metrics, irregularity and compactness.

and short axes can be determined (Figure 9.4A), and the ratio is referred to as the *elongation*. With the same axes, the bounding rectangle can be defined and the extent of the shape, that is, the ratio of shape area to the area of the bounding rectangle can be determined. The bounding rectangle for this shape metric is aligned with the major and minor axes in Figure 9.4A and does not normally coincide with the bounding box shown in Figure 9.4A. Once the average radius is known, the number of zero crossings of $r(\varphi)$ and the area ratio parameter can be computed. The number of zero crossings is the number of times the shape crosses a circle with radius r_{mean} and with the same centroid as the shape (Figure 9.4B). This metric is related to the irregularity. The area ratio parameter (ARP) is defined as

$$\text{ARP}_p = \frac{1}{N r_{\text{mean}}} \sum_{i=0}^{N-1} \lfloor r_k - r_{\text{mean}} \rfloor^p \tag{9.7}$$

where the operation $\lfloor a \rfloor$ indicates a threshold at zero, that is, a for $a > 0$ and 0 for $a < 0$. A higher value for p causes the area ratio parameter to more strongly emphasize spicular outliers.

A number of descriptors are based on the convex hull of the shape. One method of obtaining the convex hull involves probing the shape in a direction tangential to the edge from every boundary pixel. If a straight line that starts at a boundary pixel tangential to the edge intersects the shape elsewhere, this line bridges a concavity. Two such lines are shown in Figure 9.4B, and the difference between the original shape and the convex shape is highlighted. The convexity metric is the ratio of the convex hull length to the actual perimeter length, and the solidity is the ratio of the shape area to the convex area. A metric known as *Feret's diameter*, defined as the square root of the area multiplied by $4/\pi$, is sometimes used,[22] but neither this metric nor the perimeter/area ratio[11] is scale-invariant. Although these metrics seem to aid in the diagnosis of—in these examples—microvascular shape and malignancy of melanocytic skin lesions, the question of whether size plays a more important role than shape arises immediately. In other publications,[6,24] Feret's minimum and maximum diameters are defined as the minimum and maximum distance of two parallel tangents to the shape outline. Again, this metric is not scale-invariant.

Gupta and Srinath[14] suggested using statistical moments of shape boundaries. Under the definition of the unrolled perimeter r_k, the nth boundary moment m_n is defined by

$$m_n = \frac{1}{N} \sum_{i=0}^{N-1} r_k^n \tag{9.8}$$

and the nth central moment is defined by ($n > 1$)

$$M_n = \frac{1}{N} \sum_{i=0}^{N-1} (r_k - m_1)^n \tag{9.9}$$

The moments m_n and M_n are not scale-invariant. To obtain scale-invariant descriptors, the normalized moments $\bar{m}_n$ and the normalized central moments $\bar{M}_n$ are defined through

$$\begin{aligned} \bar{m}_n &= \frac{m_n}{M_2^{n/2}} = \frac{(1/N)\sum_{i=0}^{N-1} r_k^n}{\left[(1/N)\sum_{i=0}^{N-1}(r_k - m_1)^2\right]^{n/2}} \\ \bar{M}_n &= \frac{M_n}{M_2^{n/2}} = \frac{(1/N)\sum_{i=0}^{N-1}(r_k - m_1)^n}{\left[(1/N)\sum_{i=0}^{N-1}(r_k - m_1)^2\right]^{n/2}} \end{aligned} \tag{9.10}$$

Finally, Gupta and Srinath[14] formulated a feature vector with the scale-, translation-, and rotation-invariant elements F_1, F_2, and F_3, computed from the four lowest-order moments:

$$\begin{aligned} F_1 &= \frac{\sqrt{M_2}}{m_1} \\ F_2 &= \frac{M_3}{(M_2)^{3/2}} \\ F_3 &= \frac{M_4}{(M_2)^2} \end{aligned} \tag{9.11}$$

9.3. STATISTICAL MOMENT INVARIANTS

In analogy to the boundary statistical moments defined in Equations (9.8) to (9.10) and the statistical moments used to describe texture (see Section 8.1), statistical moments can be defined to describe shape. Here the pixel value of the feature is assumed to be unity, and intensity variations do not influence the shape moments. Statistical moments are computed from all pixels of a feature, as opposed to the metrics in the previous section, which are computed form the boundary points only. The kth moment M_k is defined as

$$M_k = \frac{1}{N_F}\sum_{y\in F}\sum_{x\in F}(x - x_c)^k(y - y_c)^k \tag{9.12}$$

where $x,y \in F$ indicates the x- and y-coordinates of all pixels that belong to feature F. Furthermore, N_F is the size (the number of pixels) of feature F, and x_c and y_c are the centroid coordinates. By computing the moment M_k relative to the centroid, the value of the moment becomes translation-invariant. The normalization by N_F makes the M_k scale-invariant. However, under the definition of Equation (9.12), the statistical moments are generally not rotation-invariant. Also, it can be seen that

$M_0 = 1$ and $M_1 = 0$ due to the normalization. Statistical moments tend to become larger in more irregular shapes. A more general approach was introduced by Hu[18] and later extended to invariance under general affine transformations by Flusser and Suk.[12] To obtain the moment invariants (i.e., metrics that are scale-, translation-, and rotation-invariant), Equation (9.12) is generalized to allow different powers $p,q \in \mathbb{N}$ for the two dimensions:

$$M_{p,q} = \sum_{y \in F} \sum_{x \in F} (x - x_C)^p (y - y_C)^q \tag{9.13}$$

Similar to M_1 in Equation (9.12), $M_{1,0} = M_{0,1} = M_{1,1} = 0$. Furthermore, $M_{0,0}$ is the area of the shape (the number of pixels). Therefore, scaling-invariant moments can be computed by normalizing the moment by the area,

$$\begin{aligned} \mu_{p,q} &= \frac{M_{p,q}}{M_{0,0}^{\gamma}} \\ \gamma &= \left[\frac{p+q}{2} \right] + 1 \end{aligned} \tag{9.14}$$

where $p + q \geq 2$, and $[\cdot]$ denotes the truncation of the decimals (integer operation). Hu[18] has defined seven feature metrics, ϕ_1 through ϕ_7, that are rotation-invariant and by merit of the $\mu_{p,q}$, also translation- and scaling-invariant. ϕ_1 through ϕ_7 are defined as

$$\begin{aligned} \phi_1 &= \mu_{2,0} + \mu_{0,2} \\ \phi_2 &= (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2 \\ \phi_3 &= (\mu_{3,0} - 3\mu_{1,2})^2 + (3\mu_{2,1} - \mu_{0,3})^2 \\ \phi_4 &= (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{2,1} + \mu_{0,3})^2 \\ \phi_5 &= (\mu_{3,0} - 3\mu_{1,2})(\mu_{3,0} + \mu_{1,2}) \left[(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2 \right] \\ &\quad + (3\mu_{2,1} - \mu_{0,3})(\mu_{0,3} + \mu_{2,1}) \left[3(\mu_{1,2} + \mu_{3,0})^2 - (\mu_{2,1} + \mu_{0,3})^2 \right] \\ \phi_6 &= (\mu_{2,0} - \mu_{0,2}) \left[(\mu_{1,2} + \mu_{3,0})^2 - (\mu_{2,1} + \mu_{0,3})^2 \right] \\ &\quad + 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{0,3} + \mu_{2,1}) \\ \phi_7 &= (3\mu_{2,1} - \mu_{0,3})(\mu_{1,2} + \mu_{3,0}) \left[(\mu_{1,2} + \mu_{3,0})^2 - 3(\mu_{2,1} + \mu_{0,3})^2 \right] \\ &\quad + (3\mu_{1,2} - \mu_{3,0})(\mu_{2,1} + \mu_{0,3}) \left[3(\mu_{1,2} + \mu_{3,0})^2 - (\mu_{2,1} + \mu_{0,3})^2 \right] \end{aligned} \tag{9.15}$$

Although the definitions in Equation (9.15) are most commonly used, ϕ_2 and ϕ_6 can be simplified by using $\mu_{1,1} = 0$, which leads to[33]

$$\begin{aligned} \phi_2 &= (\mu_{2,0} - \mu_{0,2})^2 \\ \phi_6 &= (\mu_{2,0} - \mu_{0,2}) \left[(\mu_{1,2} + \mu_{3,0})^2 - (\mu_{2,1} + \mu_{0,3})^2 \right] \end{aligned} \tag{9.16}$$

ϕ_1 through ϕ_7 form a feature vector that can be used to classify the shape.

Mertzios and Tsirikolias[25] propose a different definition of the statistical moments. This definition contains a normalization by the standard deviation. The centroid coordinates x_c and y_c are defined

$$x_c = \frac{1}{N_F}\sum_{x\in F} x, \qquad y_c = \frac{1}{N_F}\sum_{y\in F} y \tag{9.17}$$

where $x,y \in F$ indicates the x- and y-coordinates of all pixels that belong to feature F, and N_F is the size (the number of pixels) of feature F. The standard deviation of both coordinates, σ_x and σ_y can be computed with

$$\begin{aligned}\sigma_x &= \sqrt{\frac{1}{N_F}\sum_{x\in F}(x - x_c)^2} \\ \sigma_y &= \sqrt{\frac{1}{N_F}\sum_{x\in F}(y - y_c)^2}\end{aligned} \tag{9.18}$$

The normalized central moments are then defined through

$$m_{p,q} = \sum_{y\in F}\sum_{x\in F}\left(\frac{x - x_c}{\sigma_x}\right)^p \left(\frac{y - y_c}{\sigma_y}\right)^q \tag{9.19}$$

According to Mertzios and Tsirikolias,[25] the normalized moments defined in Equation (9.19) appear to have a better classification performance than the moments as defined in Equation (9.13) and are less sensitive to noise.

To implement the computation of moment invariants, Algorithms 9.1 and 9.2 can be used. The second pass in Algorithm 9.1 (the second double loop) can be extended by performing the summation of each pixel that belongs to the cluster according to Equation (9.13). At the end of Algorithm 9.1, the invariants ϕ_1 through ϕ_7 according to Equation (9.15) would be computed and finally stored in tables similar to size, compactness, aspect ratio, and irregularity in Algorithm 9.2.

9.4. CHAIN CODES

The chain code of a shape outline is the sequence of directions from one discretized boundary point to the next. In an example, starting at an arbitrary boundary point, the nearest boundary point in counterclockwise direction could lie to the right (east), followed by the next point northeast and the next point north, followed by another point to the north, and so on. The chain code would be E–NE–N–N. By replacing the compass directions by numbers as shown in Figure 9.6, the sequence of directions becomes 6–7–0–0. To generate the chain code of a shape, the grid on which the boundary points are discretized does not necessarily coincide with the image pixels.

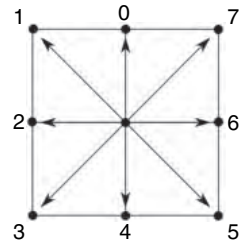


FIGURE 9.6 Enumeration of the possible directions from one grid point to its neighbor.

Rather, a coarser grid is generally desirable to avoid encoding small irregularities of the shape boundary that may be a consequence of image noise or the segmentation process. The example in Figure 9.7 shows the microscopic image of a fluorescently stained chromatid. An 8×8 pixel grid is superimposed (indicated by $+$ signs), and part of the outline is indicated (thick white line starting at S). In this example, the chain code would start with 6–7–6–7–7–0–0–7–0–0–0 $\cdots$.

Chain codes can be computed on an 8-neighborhood (as in the example of Figures 9.6 and 9.7) or on a 4-neighborhood. In that case, only the directions 0, 1, 2, and 3 would exist. The shape needs to be thick enough that its boundary can be circled unambiguously. If two parts of one feature are linked by a single diagonally

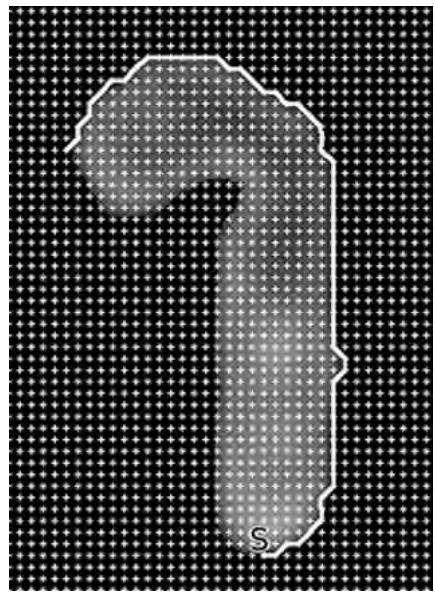


FIGURE 9.7 Generation of the chain code in an example image, a fluorescent microscope image of a chromatid. The chain starts at the lowermost point (labeled "S") and proceeds in a counterclockwise direction. The discrete grid is 8×8 pixels.

connected pixel, this pixel could be traversed diagonally or at a 90° angle. The resulting chain code depends on this decision. Furthermore, the chain code depends on the starting pixel. A suitable choice of a starting pixel, for example, the boundary point with the largest distance to the centroid, is necessary (but not sufficient) to provide a rotation-invariant shape description. In an 8-neighborhood, the chain code varies with the rotational angle of the shape because the diagonal links are longer than the horizontal and vertical links. Russ proposes to subdivide each link into a sequence of smaller links, five for the horizontal and vertical links, and seven for the diagonal links.[34] The rationale is that 7/5 is a good approximation of $\sqrt{2}$ with the consequence that each link, whether diagonal or horizontal/vertical, would have approximately the same length. Under this definition, the first two links in Figure 9.7 would be represented by the sequence 6–6–6–6–6–7–7–7–7–7–7–7. Dhawan describes an adaptive scheme to subdivide the shape boundary.[10] Two vertices are defined on the boundary. Although the initial vertices may be selected arbitrarily, the initial vertices are often placed at the end of the long axis or at points with high curvature. The maximum deviation of the line connecting the two vertices from the shape boundary is now determined (Figure 9.8A). If the line deviates from the curve more than a predetermined threshold distance ε, a new vertex is created at the maximum distance of the curve from the line, and the line is subdivided to touch the new vertex (Figure 9.8B). This divide-and-conquer approach is repeated until the approximation of the curve is satisfactory (i.e., each segment deviates less than ϵ from the curve). The original line can now be used in the opposite direction to approximate the other half of the shape. At the end of this process, the shape is approximated by a polygon with sides that are not necessarily at 0° and 45°. In a last step, the

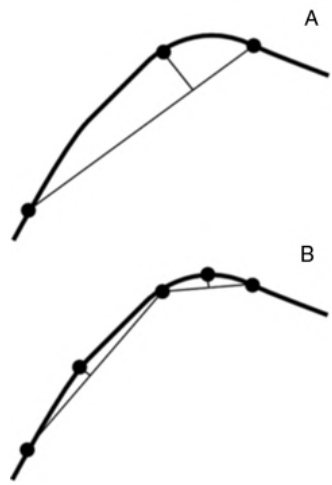


FIGURE 9.8 Approximation of a shape outline by line segments. The maximum distance of a line segment from the curve is determined (A), and if it exceeds a threshold distance, a new vertex is created and the line is split to touch the new vertex (B).

sides are subdivided into links of the chain with a horizontal, vertical, or diagonal direction and of equal length by means of a suitable algorithm, such as the Bresenham algorithm.[5]

The idea of chain codes can now be improved further by considering the orientation differences between successive segments in a counterclockwise direction, that is, the angle between two consecutive chain segments in increments of 45°. If the difference is positive in counterclockwise direction, positive differences indicate a convex section, and negative differences indicate a concave section of the shape boundary. Furthermore, symmetries of the shape are represented by symmetric sections of the differential chain code.

To extract quantitative metrics that describe the shape, the histogram of the differential chain code can be considered. In the differential chain code histogram, a strongly elongated shape would be dominated by the value zero. Irregular shapes would have higher histogram values for larger angular changes. A method that provides even more information about the shape was presented by Iivarinen et al.[19] Starting with the polygonal approximation of the shape, each segment of the polygon is in turn used as a reference line. For all segments except the one used as a reference, the angle θ to the reference line, its minimum and maximum distances, $d_{\min}$ and $d_{\max}$, respectively, are computed. A two-dimensional histogram of the distance and the angle, termed a *piecewise geometric histogram*, is filled by incrementing all histogram bins with angle θ from the smallest to the largest distance, as demonstrated in Figure 9.9. Like all multidimensional histograms, it requires a large number of data to be meaningful. Iivarinen et al.[19] chose a large bin size to limit the size of the histogram to 8×8 bins, and a 16-element feature vector was extracted from the conditional probabilities over each row and column.

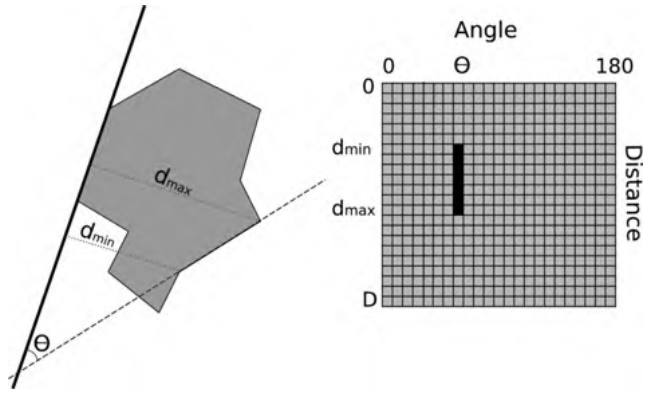


FIGURE 9.9 Construction of a pairwise geometric histogram. One line of the polygon that approximates the shape is selected as reference line (thick line). For each of the remaining sides of the polygon, the angle θ to the reference line (dashed line) and the minimum and maximum distances (dotted lines) are computed. In the pairwise geometric histogram, all bins for angle θ and for all distance values between $d_{\min}$ and $d_{\max}$ are incremented.

9.5. FOURIER DESCRIPTORS

The boundary contour of a shape is a cyclic function and can be subjected to the Fourier transform. The Fourier coefficients contain information on the frequency-domain behavior of the shape outline. Fourier descriptors were introduced by Zahn and Roskies,[42] and difference measures to compare the similarity of shapes were proposed by Persoon and Fu.[28] Several different but related formulations of the Fourier transform of a boundary are possible. First, the distance of the boundary pixels from the centroid is a cyclic sequence r_k, introduced in Equation (9.1). The Fourier coefficients c_n and d_n ($0 \leq n \leq K/2$) can be computed through the discrete Fourier transform:

$$\begin{aligned} c_n &= \frac{1}{K}\sum_{k=0}^{K-1} r_k \cos\left(-\frac{2\pi nk}{K}\right) \\ d_n &= \frac{1}{K}\sum_{k=0}^{K-1} r_k \sin\left(-\frac{2\pi nk}{K}\right) \end{aligned} \tag{9.20}$$

By using the complex notation, the complex Fourier coefficients $C_n = c_n + jd_n$ can be computed through

$$C_n = \frac{1}{K}\sum_{k=0}^{K-1} r_k \exp\left(-\frac{2\pi jnk}{K}\right) \tag{9.21}$$

The coefficients of the discrete Fourier transform are periodic with a period of K and $C_n = C_{n+zK}$, where z is any integer number, positive or negative. The Fourier coefficients are translation-invariant by merit of the r_k, but they are neither scaling- nor rotation-invariant. It can be seen that c_0 is the mean value of the sequence r_k, that is, the average radius of the shape. Since $d_0 = 0$, a normalization by $|C_0|$ or c_0 provides scaling invariance. We can therefore define scaling- and translation-invariant coefficients ζ_n for $n > 0$:

$$\zeta_n = \frac{C_n}{|C_0|} \tag{9.22}$$

The Fourier translation theorem states that a phase shift (corresponding to a rotation of the shape by the angle θ) causes a multiplicative exponential term, $\exp(j\theta)$, in the Fourier transform. This term does not change the magnitude of the Fourier coefficients, and the magnitude of ζ_n is therefore also rotation-invariant.

Alternatively, it is possible to interpret the coordinates of a boundary pixel as a complex number, $p_k = x_k + jy_k$, where the discrete Fourier transform now becomes

$$G_n = \frac{1}{K}\sum_{k=0}^{K-1} p_k \exp\left(-\frac{2\pi jnk}{K}\right) \tag{9.23}$$

Although the principle of the Fourier transform as defined in Equations (9.22) and (9.23) is the same, there are notable differences: The coefficients C_n describe the frequency decomposition of the unrolled perimeter, interpreted as a time series. Consequently, C_0 is the radius of a circle that best represents the shape. In other words, C_0 contains the shape's size information. In contrast, G_0 contains the centroid of the shape. G_1 describes an ellipse that represents the shape, and G_1 contains the shape's size information. Furthermore, the G_n are translation-invariant for $n > 0$. When Equation (9.23) is used to obtain the Fourier coefficients, other normalizations need to be used to obtain rotation-, scaling-, and translation-invariant metrics: for example, the normalization proposed originally for handprint character recognition[13]:

$$\Gamma_n = \frac{G_{1+n}G_{1-n}}{G_1^2} \tag{9.24}$$

or the computation of one single shape factor F from all coefficients for the classification of suspicious breast masses,[32]

$$F = \frac{\sum_{n=-N/2+1}^{N/2} |\mathrm{NFD}_n|/|n|}{\sum_{n=-N/2+1}^{N/2} |\mathrm{NFD}_n|} \tag{9.25}$$

where the NFD_n are normalized Fourier descriptors defined as

$$\mathrm{NFD}_n = \begin{cases} 0 & \text{for } n = 0 \\ \dfrac{G(n)}{G(1)} & \text{for } 1 \le n \le N/2 \\ \dfrac{G(n+N)}{G(1)} & \text{for } -N/2+1 \le n \le -1 \end{cases} \tag{9.26}$$

In Figure 9.10, the significance of the Fourier coefficients is shown. A circular shape with small deviations from the best-fit circle will have low values for all Fourier coefficients in the same way that the Fourier transform of a time function $f(t)$ that is almost constant will have very low or no periodic coefficients. The presence of noise produces a broadly distributed offset since noise is a signal with a wide spectral bandwidth. An elliptical shape unrolls into a sinusoidal wave, where one period fits the unrolled angle range from 0 to 2π; therefore, an ellipse would have a dominant coefficient C_1. Any variations of the outline with higher frequencies increase the values of the higher-order coefficients.

A special aspect to be considered when using Fourier descriptors is the actual tracing of the contour. Segmentation noise generally makes the contour irregular, even if the underlying object is smooth. To obtain an ordered sequence of r_k, it is necessary to follow the contour to sample the coordinates of the boundary pixels. The necessity of this process is contrary to other shape descriptors, such as compactness, aspect ratio, and irregularity, where the set of r_k does not need to be processed in

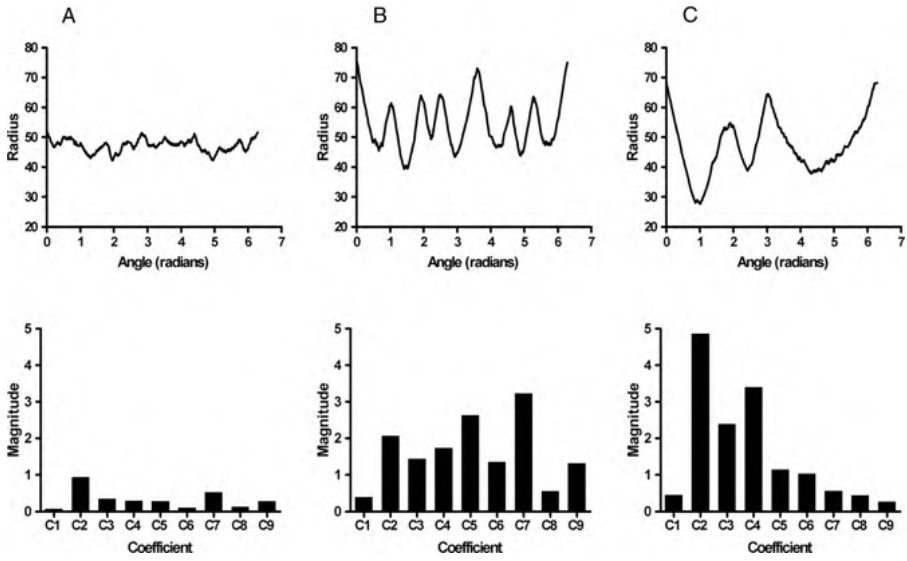


FIGURE 9.10 Unrolled radius r_k (φ), $0 \leq \varphi < 2\pi$, of the round (A), spicular (B), and half-moon (C) shapes in Figure 9.2 with the corresponding Fourier coefficients C_1 through C_9. A round shape with small high-frequency variations of the radius exhibits low, somewhat equally distributed coefficients. Large variations, as in the spicular example, show a broad distribution of coefficients, whereas large low-frequency variations (half-moon shape) are associated with high values in the lower coefficients with a fast decay toward the higher coefficients.

any specific order. Two approaches to contour tracing are possible: curve fitting and pixel-by-pixel boundary tracking.

It is possible to approximate the contour by fitting a parametric curve. One such approach is to use deformable models and active contours (Chapter 6), or the fitting of splines into the known boundary pixels. The Fourier decomposition itself [Equation (9.23)] can be used to provide a parametric representation of the contour, since the exact shape can be reconstructed from the inverse Fourier transform:

$$p_k = \sum_{n=0}^{K-1} G_k \cdot \exp\left(\frac{2\pi jnk}{K}\right) \tag{9.27}$$

Optimization methods can be used to find a set of G_n that minimizes the squared distance from the boundary pixels of the shape to the nearest reconstructed point p_k from Equation (9.27). Although the curve-fitting approach may not exactly represent the boundary pixel by pixel, it avoids the pitfalls of a pixel-based tracing algorithm. Conversely, a pixel-by-pixel contour tracing algorithm is more accurate for describing small shapes and is generally more computationally efficient.

Pixel-by-pixel boundary tracking is based on tracing a contour by moving from one boundary pixel to a connected boundary pixel. In the ideal case, the boundary is the topological equivalent of a circle, but in practical cases, this idealization hardly

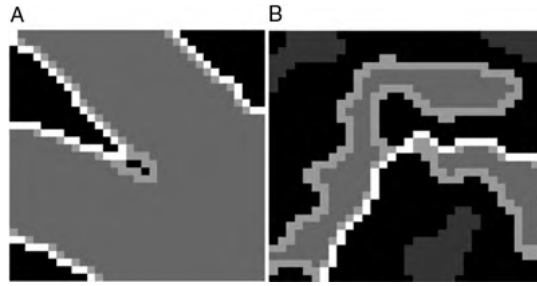


FIGURE 9.11 Ambiguous configurations of boundary pixels for tracing of a contour. Light gray pixels are boundary pixels, and white pixels are marked by a simple contour-tracing algorithm. In part A the algorithm misses a loop and takes a shortcut over an 8-connected pixel, whereas in part B, a right turn is missed because the shape is only 2 pixels wide.

ever occurs. Frequently, ambiguous cases exist where the boundary exhibits short branches or loops, or where the width of the shape is no more than one or two pixels. Two such examples are shown in Figure 9.11. In these examples, boundary pixels (light gray) are defined as pixels that are 8-connected with the background (black). A contour-tracing algorithm can easily miss boundary pixels when the boundary forms an 8-connected loop (Figure 9.11A) or when the shape is less than 3 pixels wide (Figure 9.11B). To avoid unpredictable behavior and incomplete contour tracking, the contour-tracking algorithm needs to keep track of pixels already visited. Furthermore, the algorithm needs to consider ambiguous cases by following all possibilities of ambiguity. In the case of Figure 9.11A, the algorithm needs to follow both the 4-connected loop and the 8-connected shortcut. To reconcile multiple paths, the algorithm needs to be able to backtrack along already visited pixels in case it gets stuck (such a case is unavoidable in Figure 9.11B, where the return path along the narrow section is blocked by already visited pixels) and needs to realize when two possible paths meet and decide which path is the more likely boundary path. Similar algorithms are often used in artificial-intelligence projects to guide a robot through a labyrinth.

A straightforward and robust algorithm for tracing a contour was presented by Sonka et al.[38] Assume that we have enumerated the directions similar to Figure 9.6, but with direction zero being East. With two tables, `xdisp` and `ydisp`, it is possible to revert the enumerated direction to a step offset. The algorithm assumes that `xs` and `ys` point at the leftmost pixel of the topmost row of the feature, a pixel that can typically be found by a scan from top to bottom and for each row, from left to right. The algorithm, designed for an 8-connected boundary, is outlined in Algorithm 9.3. It touches each boundary pixel and allows us to store them in an array (indexed by `k`) for further processing, such as the Fourier transformation or chain code analysis. For a 4-connected neighborhood, the initial direction in the fourth line would be 3 and the displacement tables would be [1,0,−1,0] and [0,−1,0,1], respectively. Finally, computation of the new direction would no longer distinguish between diagonal and horizontal/vertical cases but would simply be updated as `(dir+3) mod 4`.

```
allocate xdisp[8]=(1,1,0,-1,-1,-1,0,1);      // Allocate tables to convert enumerated...
allocate ydisp[8]=(0,-1,-1,-1,0,1,1,1);      // ...directions back into displacements
x=xs; y=ys;                                  // Initialize search coordinates
dir=7; k=0;                                  // initial direction and step count

repeat                                       // Enter loop to follow closed contour
   repeat                                    // Search for next boundary pixel
      if (dir AND 1) then                    // diagonal direction
         dir = (dir+6) mod 8;
      else                                   // hor/vert direction
         dir = (dir+7) mod 8;
      endif;
      xnew=x+xdisp[dir];                     // Probe the new direction
      ynew=y+ydisp[dir];
      if (IM(xnew,ynew)!=0) then             // valid boundary pixel found
         x=xnew; y=ynew;                     // Take the step, also a condition...
      endif;                                 // ...to exit this loop
   until ((x=xnew) and (y=ynew));
                  // Right here, we would record x and y as functions of k
   k=k+1;                                    // Count the number of steps
until ((k>2) and (abs(x-xs)<2) and (abs(y-ys)<2));
```

Algorithm 9.3 Contour tracing for an 8-connected boundary. The input image is `IM(x,y)` and is assumed to be binary; that is, background pixels have a value of zero and foreground pixels have a nonzero value. The start point is `xs`, `ys` and must be the top left boundary pixel of the shape. At the end of the inner loop, the boundary pixels x,y as a function of the sequence index `k` are available for processing or storage.

Algorithm 9.3 does not produce a rotation-invariant sequence. To obtain a rotation-invariant sequence, the algorithm may be followed by cyclical shifting of the boundary pixel sequence so that the starting point is a special landmark point: for example, the boundary pixel with the largest distance to the centroid.

9.6. TOPOLOGICAL ANALYSIS

Topology is a branch of mathematics that deals with the connectivity of shapes under certain transformations. A shape is said to be *topologically invariant* even if it is stretched and bent in rubber band fashion. A shape can be thought to be drawn on a sheet of rubber, and the shape would remain the same in terms of topology if the rubber sheet is stretched and distorted but not torn. A trail around a lake is the topological equivalent of the letter O, but if a bridge crosses the lake in the middle, thus connecting two sides of the trail, it becomes a topologically different shape, now the equivalent of the letter B. For shape analysis, the description in terms of topology is a description of connectivity. This notion leads to graph theory, where a *graph* is defined as a set of nodes (or vertices) that are connected by edges. In two

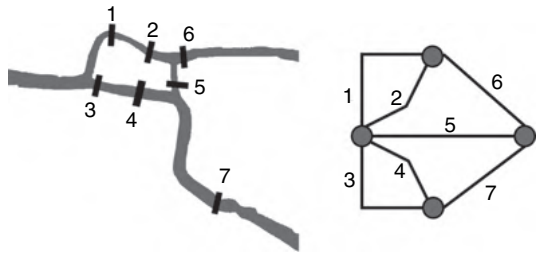


FIGURE 9.12 The seven bridges of Königsberg (only five of them exist today) and the associated graph, where each island is a vertex and each bridge is an edge connecting two vertices. The notion of island includes the right and left banks as idealized infinite islands.

dimensions, edges cannot cross each other, or they would create a new vertex (and with it, a topologically different structure). The mathematician L. Euler proved in 1735 that there is no walk that crosses the seven bridges of Königsberg once and only once (Figure 9.12). This proof initiated the field of topology. For the purpose of the topological description of a shape, the shape and length of the edges (here, the bridges) are not relevant. The graph in Figure 9.12 has four vertices (the two islands and the right and left banks) and seven edges (the bridges). Furthermore, the graph has four holes or loops. The *Euler number* e, also known as the Euler characteristic, is defined as

$$e = v - n + l \tag{9.28}$$

where v is the number of vertices, n the number of edges, and l the number of loops. For the graph in Figure 9.12, $e = 1$. In any graphlike structure, the values of v, n, l, and e are invariant with respect to rotation, scaling, and translation. Furthermore, they are also invariant under affine transformations. These values are therefore particularly suited to describe a meshlike structure.

For long branched or intertwined shapes, the underlying graph of vertices and connections can be extracted through *skeletonization*, a thinning process that reduces shapes to their medial axis. Mathematically, the *medial axis* can be defined as the set of centers of inscribed disks of maximum diameter that touch the shape boundary and are tangential to the boundary without intersecting it.[9] A different but related definition uses the Euclidean distance from a pixel to the nearest background pixel: For each feature pixel, the distance to all background pixels is computed, and the background pixel with the minimum distance is determined. If two background pixels with the same minimum distance exist, the corresponding feature pixel belongs to the medial axis.[4] Probably the most widely used algorithm to compute a skeleton is the iterative algorithm of Zhang and Suen,[43] which, however, has the disadvantage of being very noise-sensitive. The skeletonization process is shown in Figure 9.13, where a fingerprint is thresholded to provide the binary image required for skeletonization and then thinned using the Zhang–Suen algorithm. A magnified section is shown in Figure 9.14. A *vertex* is any pixel where three or more edges are joined. Any edge that connects two vertices is called a *link*, any edge that is connected to only one

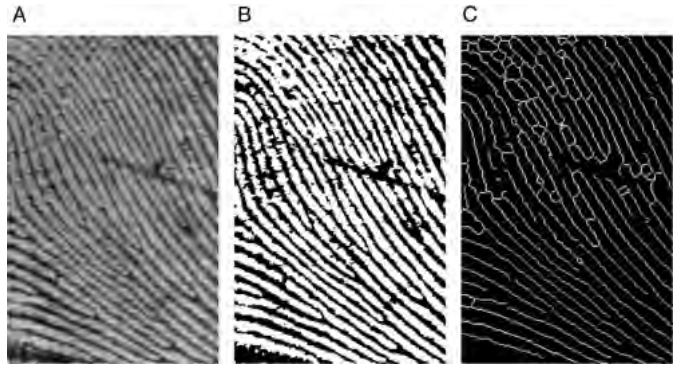


FIGURE 9.13 Example of a skeletonization process. A fingerprint (A, ridges are lighter than valleys) is thresholded (B), and the features—white ridges—are thinned by skeletonization.

vertex is called a *branch*, and the other end of the branch is termed an *endpoint*. A closed sequence of links and vertices is called a *loop*. The numbers of vertices, links, branches, and loops are the topological invariants. They describe the connectivity of the network independent of its position, scale, or rotation, and the metrics are theoretically invariant under affine transformations. However, in practice, even minor changes of the binary images can cause major changes in the network. An example is given in Figure 9.15. The skeleton of a rectangle is a line (although under alternative definitions the medial axis transform would contain four diagonal branches). If the rectangle contains single black pixels, the skeleton loops around those pixels. In addition, even the slightest irregularity in the outline of the rectangle would create additional links. Although this behavior is topologically correct (the second rectangle in Figure 9.15 contains three holes, albeit small ones, so the graph contains three loops), it implies that even small changes in the preprocessing or segmentation step may cause large changes in the topological invariants. This sensitivity needs to be taken into account when a skeleton is being used for shape characterization.

FIGURE 9.14 Magnified section of the fingerprint skeleton in Figure 9.13.

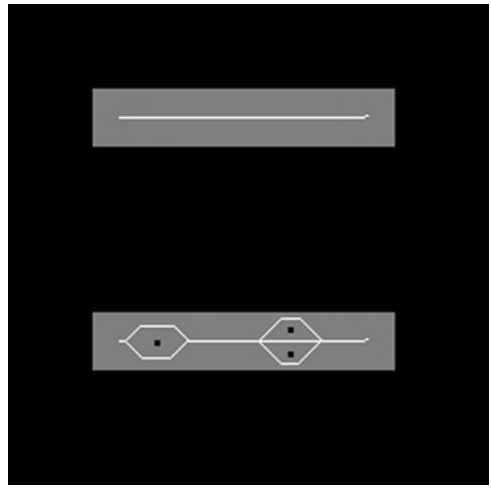


FIGURE 9.15 Skeletonization of a solid rectangle and a rectangle with some random pixels in it.

The skeletonization algorithm by Zhang and Suen produces a skeleton by iterative conditional erosion of the feature. A pixel may not be deleted in the erosion process if it is an endpoint (i.e., it has fewer than two 8-connected neighbors) or if deleting this pixel would split an 8-connected feature into two. Zhang and Suen listed a number of neighborhood patterns that define a boundary pixel that may be eroded. The most efficient implementation is to use *fate tables*, encoded neighborhood pixel patterns that determine whether or not a pixel may be deleted. This is particularly convenient, as each neighbor may be encoded with 1 bit of a byte, and eight neighbors allow for $2^8 = 256$ possible neighborhood patterns (combinations). The neighborhood and its associated code values are explained in Figure 9.16. The code values are powers of 2,

8 *128*	1 *1*	2 *2*
7 *64*	0	3 *4*
6 *32*	5 *16*	4 *8*

FIGURE 9.16 Definition of the neighborhood indices from 1 to 8, clockwise, and the associated code values (in italics).

and adding them together produces any value from zero to 255, depending on the bit pattern. If the top and both horizontal neighbors, for example, are set (i.e., neighbors numbered 1, 3, and 7), the resulting code would be $1 + 4 + 64 = 69$. This index can be used to look up instructions on how to handle the central pixel (the fate of the pixel) in a table. The actual skeletonization is iterative; that is, any boundary points get eroded, if permitted by the fate table, until no more points have been eroded. The permissions are slightly different in alternating iterations, and the fate table reflects this difference. If the fate table holds a zero for a specific bit pattern, the pixel cannot be removed. If it holds a 1, the pixel may be removed in odd-numbered iterations; if it holds a 2, the pixel may be removed in even-numbered iterations; if it holds a 3, the pixel may be removed in any iteration. The sixty-ninth fate table entry, to stay with the example above, is zero. If such a bit pattern occurs, the central pixel may not be removed. The iterative skeletonization process that builds on this scheme is shown in Algorithm 9.4. This algorithm was used to generate Figure 9.13C from Figure 9.13B.

From the skeleton, the connectivity metrics (i.e., the topological invariants) can be determined, such as the number of nodes, links, branches, and loops. For this purpose the skeleton needs to be analyzed, and the following steps can be applied to obtain these metrics. In the first pass, the neighborhood connectivity of each pixel can be analyzed. Any pixel with exactly one neighbor is an endpoint and marks the start of a branch. A vertex is any pixel with more than two neighbors that do not touch other neighbors. Isolated dots (pixels with no neighbors) are also possible, but these are generally the consequence of noise. In the next pass over the image, the endpoints can be iteratively eroded until a vertex or another endpoint is reached. As a result, all branches are now marked. If, in this process, a pixel is eroded from two sides, it indicates the presence of a loop. Alternatively, the number of loops l can be found by subtracting the number of vertices from the number of links plus one. Finally, the Euler number can be computed through Equation (9.28). After this pass, the only structures remaining are isolated loops that have no vertices and one branch. If these were to be counted as loops, they would also contribute one branch each. In addition to these topological metrics, noninvariant metrics, such as the average length of links, branches, and loops, can be computed. Any of these metrics can be combined to provide a feature vector for shape description. In addition, it is possible to further process the skeleton to gain additional descriptors. The skeleton can also be examined for self-similar properties by estimating its box-counting or Minkowsky dimension (see Section 10.2). If a skeleton is characterized by few dominant links, their length and curvature may be examined.

The skeletonization algorithm of Zhang and Suen does not remove all four-connected pixels, as demonstrated in Figure 9.17. If a strictly 8-connected skeleton is required, an additional processing step needs to be added where 4-connected pixels are identified and removed, as in Figure 9.17B. The sensitivity of the skeletonization process toward noise and irregular shape outlines usually leads to numerous small branches, such as the 2-pixel branch coming off the downward link at the right side of Figure 9.17. The skeleton can be pruned to remove those branches. A simple pruning step would involve iterative erosion of endpoints with a limited number of iterations and under the constraint that erosion is strictly limited to branches (i.e., vertices and

```
allocate F(xmax,ymax);                    // fate array
allocate fatetbl[256] = {
   0,0,0,1,0,3,1,1,0,0,0,0,2,2,3,3,0,0,0,0,3,0,0,0,2,0,0,0,2,0,3,2,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,3,0,0,0,3,0,3,2,
   0,3,0,1,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,
   2,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,3,0,0,0,2,0,2,0,
   0,1,0,3,0,1,0,3,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   1,1,0,3,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   3,3,0,3,0,0,0,1,0,0,0,0,0,0,0,0,3,1,0,1,0,0,0,0,3,1,0,0,2,0,0,0
};

iter=0;                                   // Count number of iterations
repeat
   n=0;                                   // Count number of pixels removed
   for (y=1 while y<ymax-1 increment y=y+1) do // Go over the entire image
      for (x=1 while x<xmax-1 increment x=x+1) do
         F(x,y)=0;
         if (IM(x,y)>0) then              // Consider only foreground pixels
            p=0;                  // Use p to prepare the bit code (entry into fatetbl, Figure 9.16)
            if (IM(x,y-1) > 0) then p=p+1;
            if (IM(x+1,y-1) > 0) then p=p+2;
            if (IM(x+1,y) > 0) then p=p+4;
            if (IM(x+1,y+1) > 0) then p=p+8;
            if (IM(x,y+1) > 0) then p=p+16;
            if (IM(x-1,y+1) > 0) then p=p+32;
            if (IM(x-1,y) > 0) then p=p+64;
            if (IM(x-1,y+1) > 0) then p=p+128;

            if ( (fatetbl[p]=3) or
                  (fatetbl[p]=2 and (iter AND 1) = 0) or    // even iteration
                  (fatetbl[p]=1 and (iter AND 1) = 1) then    // odd iteration
               F(x,y)=1;                      // Mark pixel for deletion
            endif;
         endif;
      endfor;
   endfor;          // Done marking; now actually delete the pixels

   for (y=1 while y<ymax-1 increment y=y+1) do  // Go over the entire image again
      for (x=1 while x<xmax-1 increment x=x+1) do
         if (F(x,y)>0) then
            IM(x,y)=0;                    // Actually delete pixels
            n=n+1;                        // Count number of deleted pixels
         endif;
      endfor;
   endfor;

   iter=iter+1;
until (n=0);                                    // Converged if no more pixels deleted
delete (F);
```

Algorithm 9.4 Skeletonization of a binary image. The input image is `IM(x,y)` with size `xmax` and `ymax` and is assumed to be binary (a background pixel is zero, any nonzero pixels are foreground pixels). The algorithm applies the rules in the fate table repeatedly until no more pixels have been removed. Note that the `AND` operator represents a bit-wise logical operation.

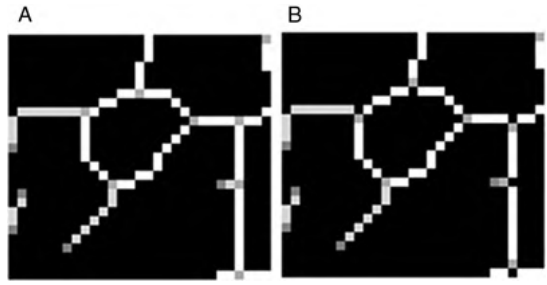


FIGURE 9.17 The skeletonization process by Zhang and Suen leaves several 4-connected pixels in the network (A). Those pixels need to be identified and removed in a separate step if a strictly 8-connected skeleton is desired (B).

links cannot be eroded). At the end of this step, any branch that has fewer pixels than the number of iterations is removed completely. To restore the original, but pruned, skeleton, the erosion step is followed by iterative dilation of the endpoints under the constraint that only pixels that belonged to the original skeleton can be added by dilation. After this step, all branches that have not been pruned are restored to their original lengths.

9.7. BIOMEDICAL EXAMPLES

The methods described in this chapter are widely used in the analysis of biomedical images and computer-aided radiology. Even simple shape metrics that are obtained relatively easily are helpful for diagnosis. Pohlmann et al.[29] analyzed x-ray mammograms and used several metrics, including roughness, compactness [Equation (9.4)], circularity [inverse irregularity, Equation (9.6)], the boundary moments [a metric related but not identical to Equation (9.10)], and the fractal dimension of the boundary, to distinguish between benign and malignant lesions. ROC analysis showed the boundary roughness metric to be most successful, followed by a metric of margin fluctuation: The distance between the shape outline and a filtered outline (obtained, for example, by lowpass-filtering the unrolled boundary in Figure 9.10A and by restoring the shape from the smoothed outline function). Boundary moments and fractal dimension showed approximately the same discriminatory power, followed by irregularity and compactness. Pohlmann et al.[29] also demonstrated that the spatial resolution of the digitized x-ray images had a strong impact on the ROC curve. A 0.2-mm pixel size lead to poor discrimination, and increasing the resolution to 0.05 mm significantly increased the area under the curve. Further increasing the resolution to 0.012 mm did not significantly increase the area under the curve.

The influence of resolution on the classification of mammographic lesions was also examined by Bruce and Kallergi,[7] in addition to the influence of the segmentation method. In this study, both spatial resolution and pixel discretization were changed, and x-rays digitized at 0.22 mm resolution and 8 bits/pixel were compared with x-rays

digitized at 0.18 mm resolution and 16 bits/pixel. Segmentation involved adaptive thresholding, Markov random field classification of the pixels, and a binary decision tree. Shape features included compactness, normalized mean deviation from the average radius [Equation (9.7)], and a roughness index. Furthermore, the sequence of r_k was decomposed using the wavelet transform and the energy at each decomposition level computed, leading to a 10-element feature vector. Although the wavelet-based energy metric outperformed the other classification metrics, it was noted that the lower-resolution decomposition provided better classification results with the lower-resolution images, an unexpected result attributed to the design of the segmentation process. This study concludes that any classification depends strongly on the type of data set for which it was developed.

A detailed description of the image analysis steps to quantify morphological changes in microscope images of cell lines can be found in a publication by Metzler et al.[26] Phase-contrast images were first subjected to a thresholding segmentation, where both the window size and the threshold level were adaptive. The window size was determined by background homogeneity, and the threshold was computed by Otsu's method within the window. A multiscale morphological opening filter, acting on the binarized image, was responsible for separating the cells. Finally, the compactness, here formulated similar to equation (9.5), characterized the shape. Ethanol and toxic polymers were used to change the shape of fibroblasts, and the compactness metric increased significantly with increased levels of exposure to toxic substances, indicating a more circular shape after exposure.

Often, spatial-domain boundary descriptors and Fourier descriptors are used jointly in the classification process. Shen et al.[37] compared the shape's compactness [Equation (9.4)], statistical moments [Equation (9.13)], boundary moments [Equation (9.11)], and the Fourier-based form factor [Equation (9.25)] in their ability to distinguish between different synthesized shapes and mammographic lesions. Multidimensional feature vectors proved to be the best classifiers, but the authors of this study concluded that inclusion of different x-ray projections and additional parameters, such as size and texture, should be included in the mammographic analysis system. In fact, shape and texture parameters are often combined for classification. Kim et al.[21] define a jag counter (*jag points* are points of high curvature along the boundary) and combine the jag count with compactness and acutance to differentiate benign and malignant masses in breast ultrasound images. *Acutance* is a measure of the gradient strength normal to the shape boundary. For each boundary pixel j, a gradient strength d_j is computed by sampling N_j points normal to the contour in pixel j,

$$d_j = \sum_{i=1}^{N_j} \frac{f(i) - b(i)}{2i} \tag{9.29}$$

where the $f(i)$ are pixels inside the tumor boundary and the $b(i)$ are pixels outside the boundary. The acutance A is computed by averaging the d_j along the boundary:

$$A = \frac{1}{d_{\max}} \sqrt{\frac{1}{N} \sum_{j=0}^{N-1} \frac{d_j^2}{Nj}} \tag{9.30}$$

Therefore, the acutance is a hybrid metric that, unlike all metrics introduced in this chapter, includes intensity variations along the boundary. In this study, the jag count showed the highest sensitivity and specificity, followed by acutance and compactness.

A similar approach was pursued by Rangayyan et al.[32] Image analysis was performed in x-ray mammograms with the goal of classifying the lesions. The feature vector was composed of the acutance, compactness, the Fourier-domain form factor [Equation (9.25)], and another hybrid metric, a modification of the statistical invariants presented in Equation (9.13). Here, the invariants included intensity deviations from the mean intensity I_{mean} as defined by

$$m_{p,q} = \sum_{y \in F} \sum_{x \in F} (x - x_c)^p (y - y_c)^q \, |I(x,y) - I_{\text{mean}}| \tag{9.31}$$

In this study, compactness performed best among the single features, and a combination feature vector further improved the classification.

Adams et al.[1] found that a combination of a compactness measure and a spatial frequency analysis of the lesion boundary makes it possible to distinguish between fibroadenomas, cysts, and carcinomas, but to allow statistical separation, three MR sequences (T1 contrast, T2 contrast, and a fat-suppressing sequence) were needed. More recently, Nie et al.[27] combined shape features (including compactness and radial-length entropy) with volume and gray-level metrics (entropy, homogeneity, and sum average). A neural network was trained to classify the lesion and distinguish between benign and malignant cases.

Fourier descriptors, apart from describing shape features, can be useful for various image processing steps. Wang et al.[39] suggested using Fourier coefficients to interpolate the contour of the prostrate gland in magnetic resonance images. Interpolation was performed by zero padding, adding a number of higher-frequency coefficients with complex-zero value. The inverse Fourier transform now has more sample points along the contour. With this technique, contour points from coronal and axial MR scans were joined, and a sufficient number of vertices on the three-dimensional contour was provided for three-dimensional visualization and determination of centroid and size. Lee et al.[23] used Fourier coefficients to obtain a similarity measure between outlines of vertebrae in x-ray projection images. In the x-ray image, the vertebral contour was sampled manually, and an iterative curve evolution scheme was used to reduce the number of sample points in their dependence on the local curvature. Based on the remaining sample points, a polygonal interpolation was used to represent the shape. This polygon was then transformed into the Fourier domain, and the root-mean-squared distance between the Fourier coefficients was used to compare contours and provide a similarity metric. Such a similarity metric can also be useful in image retrieval, where similar shapes may occur in different rotational positions or may be scaled. Sánchez-Marín[35] used Fourier descriptors to identify cell shapes in segmented edge images of confluent cell layers. The performance of the Fourier descriptors was compared to that of a curvature descriptor. Cell outlines were first filtered to create a smoothed outline and then rotated so that the sequence r_k begins

with the largest value of r_k. In this way, a certain measure of rotation invariance was achieved.

Chain codes have been used frequently to characterize shape features. Shelly et al.[36] introduced image processing steps to classify cell nuclei of Pap smear cells as normal or abnormal and to identify extracervical cells. Light microscope images were subjected to edge detection and erosion in preparation for computation of the chain code. The chain code length (in fact, the boundary length) was used to distinguish nuclei from other features in the image. Other features obtained from the chain code were the minimum diameter, the maximum diameter, and the ellipticity. In the same context, that is, images of Pap-stained cervical cells, Bengtsson et al.[2] used chain code metrics to detect overlapping cell nuclei. The difference chain code was used primarily to identify pairs of concavities that indicate overlapping convex shapes: in this case, overlapping nuclei. The concavities were then characterized with a number of metrics, such as the spacing along the contour relative to the length of the contour and the minimum distance of the concavities relative to the length of the contour. A decision tree was used to identify the most relevant concavities and therefore to reduce the false positives.

Chain codes were also used for the segmentation of the left ventricular boundaries in cardiac ultrasound images. Zheng et al.[44] presented an image processing chain that started with adaptive median filtering for noise reduction and thresholded histogram equalization for contrast enhancement. Edge detection was based on region growing of the low-echogenic center of the ventricle in combination with a gradient threshold to identify edge pixels. The resulting shape was lowpass-filtered, and a polynomial interpolation provided a smoothed contour. Finally, the chain code was obtained and used to determine roundness, area, and wall thickness frame by frame as a function of time.

The skeleton of a shape is used when the features in the image are elongated rather than convex and when the features form an interconnected network. A very popular application of connectivity analysis through skeletonization is the analysis of the trabecular network in spongy bone. Spongy bone is a complex three-dimensional network of strutlike structures, the trabeculae. It is hypothesized that bone microarchitecture deteriorates with diseases such as osteoporosis. X-ray projection images, micro-CT images, and high-resolution MR images provide a three-dimensional representation of the trabecular network. An example is given in Figure 9.18, which shows digitized x-ray images of bovine vertebral bone. Figure 9.18A and B show approximately the same region of the same bone, but before the x-ray image in Figure 9.18B was taken, the bone was exposed to nitric acid for 30 min to simulate the microarchitectural deterioration observed in osteoporosis.[3] Since x-ray absorption by bone minerals is high, simple thresholding is usually sufficient to separate bone regions from background. Figure 9.18C and D show the thresholded region (gray) and the skeleton (white) of the projected trabecular structure. Visual inspection reveals the loss of structural detail and the loss of connectivity. Skeleton metrics further demonstrate how the connectivity has been reduced by exposure to the acid: There are more isolated elements (33 after acid exposure and 11 before acid exposure), fewer links and holes (2240 links and 725 holes after acid exposure; 2853 links and 958 holes

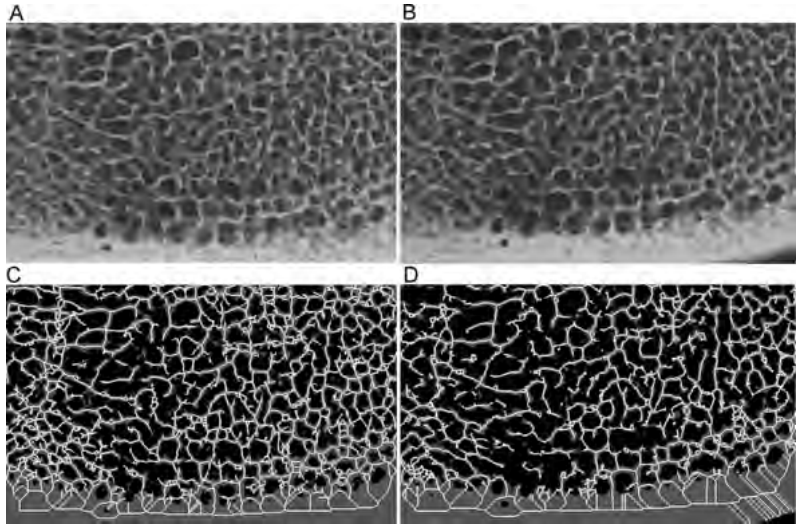


FIGURE 9.18 Microradiographs of a slice of bovine bone (A) and the same bone after 30 minutes of exposure to nitric acid (B). Both radiographs were thresholded at the same level (gray areas in parts C and D), and the skeleton, dilated once for better visualization and pruned to a minimum branch length of 10 pixels, superimposed. Although the changes caused by the acid exposure are not easy to detect in the original radiographs, they are well represented by the connectivity metrics of the skeleton.

before acid exposure), and fewer vertices (1516 vertices after exposure; 1896 vertices before). Conversely, there is almost no change in scaling-dependent metrics such as the average side length. Figure 9.18 also demonstrates the sensitivity of the skeleton toward noise and minor variations in the shape outline. The lower right corner of Figure 9.18D contains a relatively broad area of compact bone. Skeletonization leads to artifactual diagonal loops. These do not occur in Figure 9.18C because the section of compact bone touches the image border. The area outside the image is considered to be background, and consequently, there exists a straight boundary.

In a comparative study[15] where the relationship of topological metrics and other shape and texture parameters to bone mineral density, age, and diagnosed vertebral fractures were examined, topological parameters exhibited poor relationship with the clinical condition of the patients. Typical texture metrics, such as the run length and fractal dimension (see Chapters 8 and 10, respectively), showed better performance. This study seemingly contradicts the successful measurement of topological parameters in x-ray images of bone biopsies[8,17] and high-resolution magnetic resonance images.[40] The main difference between the studies is spatial resolution. It appears that a meaningful application of topological analysis to trabecular bone requires a high enough resolution to resolve individual trabeculae, whereas lower-resolution methods such as computed tomography call for different classification approaches.[15] Given high enough resolution, the topological parameters allow us to characterize

the bone in several respects. Kabel et al.[20] used slice-by-slice micrographs of human bone biopsies with a spatial resolution of 25 μm to reconstruct the three-dimensional network of trabeculae. The Euler number, normalized by the bone volume, was compared with the elastic properties of the bone as determined by finite-element simulations. A weak negative correlation was found between the normalized Euler number (used as a metric of connectivity) and the elastic modulus. These results were corroborated in a study by Yeni et al.,[41] who examined micro-CT images of bone biopsies and also found a weak negative correlation between the volume-adjusted Euler number and the elastic modulus. A different notion of the Euler number was introduced by Portero et al.[30] and defined as the number of holes minus the number of trabeculae connected around the holes. This modified Euler number was found to correlate well with the elastic modulus of calcaneus specimens determined by actual biomechanical strength tests.[31]

Connectivity was found to be a relevant parameter when a question arose as to whether the presence of gold nanoparticles would influence the cross-linking behavior of collagen fibrils.[16] Figure 9.19 shows a representative scanning electron microscopy image of collagen fibrils together with the thresholded and skeletonized binary images. A higher level of cross-linking was related to more curved fibrils, which under visual observation had more self-intersections. Correspondingly, the skeleton showed a higher number of links and a shorter average length of the links. The overall number of holes did not differ between the nanoparticles and the control

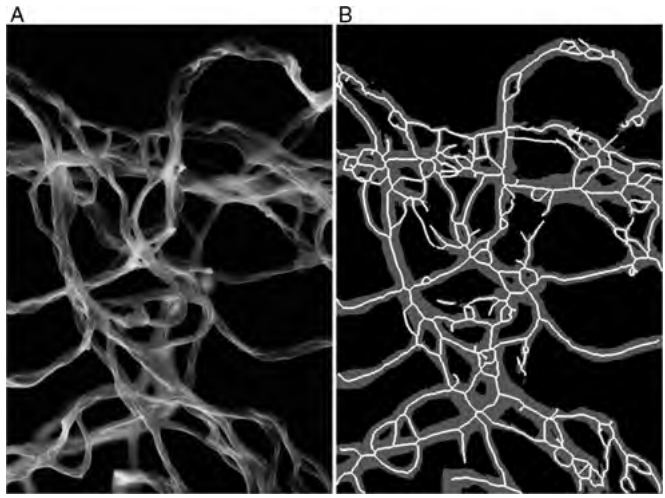


FIGURE 9.19 Scanning electron microscope image of collagen fibrils (A). After threshold segmentation, the skeleton can be created (B), where the gray region is the thresholded fibril region, and white lines represent the skeleton. Connectivity, represented by the number of edges and the number of nodes, was higher in the presence of gold nanoparticles. The connectivity reflected the tendency of the fibrils to curl and self-intersect when gold nanoparticles were present.

group, giving rise to a normalized metric of the number of branches relative to the number of holes. In the presence of nanoparticles, this number was found to be 2.8 times larger than in the control group. This ratiometric number was found to be an objective metric of the degree of fibril cross-linking that is independent of the number of fibrils and the microscope magnification and field of view by merit of the normalization.

REFERENCES

1. Adams AH, Brookeman JR, Merickel MB. Breast lesion discrimination using statistical analysis and shape measures on magnetic resonance imagery. Comput Med Imaging Graph 1991; 15(5):339–349.
2. Bengtsson E, Eriksson O, Holmquist J, Jarkrans T, Nordin B, Stenkvist B. Segmentation of cervical cells: detection of overlapping cell nuclei. Computer Graph Image Process 1981; 16:382–394.
3. Berry JL, Towers JD, Webber RL, Pope TL, Davidai G, Zimmerman M. Change in trabecular architecture as measured by fractal dimension. J Biomech 1996; 29(6):819–822.
4. Blum H. Biological shape and visual science. I. J Theor Biol 1973; 38(2):205–287.
5. Bresenham JE. Algorithm for computer control of a digital plotter. In: Seminal Graphics: Pioneering Efforts That Shaped the Field. New York: ACM, 1993:1–6.
6. Briguet A, Courdier-Fruh I, Foster M, Meier T, Magyar JP. Histological parameters for the quantitative assessment of muscular dystrophy in the mdx-mouse. Neuromuscul Disord 2004; 14(10):675–682.
7. Bruce LM, Kallergi M. Effects of image resolution and segmentation method on automated mammographic mass shape classification. Proc SPIE 1999; 3661:940.
8. Chappard D, Legrand E, Haettich B, Chales G, Auvinet B, Eschard JP, Hamelin JP, Basle MF, Audran M. Fractal dimension of trabecular bone: comparison of three histomorphometric computed techniques for measuring the architectural two-dimensional complexity. J Pathol 2001; 195(4):515–521.
9. Choi HI, Choi SW, Moon HP. Mathematical theory of medial axis transform. Pac J Math 1997; 181(1):57–88.
10. Dhawan AP. Medical Image Analysis. Piscataway, NJ: IEEE Press, 2003.
11. Fikrle T, Pizinger K. Digital computer analysis of dermatoscopical images of 260 melanocytic skin lesions; perimeter/area ratio for the differentiation between malignant melanomas and melanocytic nevi. J Eur Acad Dermatol Venereol 2007; 21(1):48–55.
12. Flusser J, Suk T. Pattern recognition by affine moment invariants. Pattern Recogn 1993; 23(1):167–174.
13. Granlund GH. Fourier preprocessing for hand print character recognition. IEEE Trans Comput 1972; 21(2):195–201.
14. Gupta L, Srinath MD. Contour sequence moments for the classification of closed planar shapes. Pattern Recogn 1987; 20(3):267–272.
15. Haidekker MA, Bidesi A, Radmer S, Andresen R. Texturpaprameter zur Bestimmung osteoporotisch bedingter Strukturveränderungen im CT-Bild der Wirbelkörperspongiosa: eine Vergleichsstudie [Texture parameters to quantify osteoporosis-related structural

changes in CT images of the vertebral spongiosa: a comparative study]. Osteologie 2006; 15:120–130.

16. Haidekker MA, Boettcher LW, Suter JD, Rone R, Grant SA. Influence of gold nanoparticles on collagen fibril morphology quantified using transmission electron microscopy and image analysis. BMC Med Imaging 2006; 6:4.
17. Hanyu T, Arai K, Takahashi HE. Structural mechanisms of bone loss in iliac biopsies: comparison between rheumatoid arthritis and postmenopausal osteoporosis. Rheumatol Int 1999; 18(5–6):193–200.
18. Hu MK. Visual pattern recognition by moment invariants. IEEE Trans Inf Theory 1962; 8(2):179–187.
19. Iivarinen J, Peura M, Särelä J, Visa A. Comparison of combined shape descriptors for irregular objects. Proc 8th Br Machine Vis Conf 1997; 2:430–439.
20. Kabel J, Odgaard A, van Rietbergen B, Huiskes R. Connectivity and the elastic properties of cancellous bone. Bone 1999; 24(2):115–120.
21. Kim KG, Kim JH, Min BG. Classification of malignant and benign tumors using boundary characteristics in breast ultrasonograms. J Digit Imaging 2002; 15(Suppl 1):224–227.
22. Korkolopoulou P, Konstantinidou AE, Kavantzas N, Patsouris E, Pavlopoulos PM, Christodoulou P, Thomas-Tsagli E, Davaris P. Morphometric microvascular characteristics predict prognosis in superficial and invasive bladder cancer. Virchows Arch 2001; 438(6):603–611.
23. Lee DJ, Antani S, Long LR. Similarity measurement using polygon curve representation and Fourier descriptors for shape-based vertebral image retrieval. Proc SPIE 2003; 5032:1283–1291.
24. Loferer-Krossbacher M, Klima J, Psenner R. Determination of bacterial cell dry mass by transmission electron microscopy and densitometric image analysis. Appl Environ Microbiol 1998; 64(2):688–694.
25. Mertzios BG, Tsirikolias K. Statistical shape discrimination and clustering using an efficient set of moments. Pattern Recogn Lett 1993; 14(6):517–522.
26. Metzler V, Bienert H, Lehmann T, Mottaghy K, Spitzer K. A novel method for quantifying shape deformation applied to biocompatibility testing. ASAIO J 1999; 45(4):264–271.
27. Nie K, Chen JH, Yu HJ, Chu Y, Nalcioglu O, Su MY. Quantitative analysis of lesion morphology and texture features for diagnostic prediction in breast MRI. Acad Radiol 2008; 15(12):1513–1525.
28. Persoon E, Fu KS. Shape discrimination using Fourier descriptors. IEEE Trans Syst Man Cybern B 1977; 7(3):170–179.
29. Pohlmann S, Powell KA, Obuchowski NA, Chilcote WA, Grundfest-Broniatowski S. Quantitative classification of breast tumors in digitized mammograms. Med Phys 1996; 23(8):1337–1345.
30. Portero NR, Arlot ME, Roux JP, Duboeuf F, Chavassieux PM, Meunier PJ. Evaluation and development of automatic two-dimensional measurements of histomorphometric parameters reflecting trabecular bone connectivity: correlations with dual-energy x-ray absorptiometry and quantitative ultrasound in human calcaneum. Calcif Tissue Int 2005; 77(4):195–204.
31. Portero-Muzy NR, Chavassieux PM, Mitton D, Duboeuf F, Delmas PD, Meunier PJ. Euler(strut.cavity), a new histomorphometric parameter of connectivity reflects bone

strength and speed of sound in trabecular bone from human os calcis. Calcif Tissue Int 2007; 81(2):92–98.

32. Rangayyan RM, El-Faramawy NM, Desautels JEL, Alim OA. Measures of acutance and shape for classification of breast tumors. IEEE Trans Med Imaging 1997; 16(6):799–810.
33. Reeves AP, Prokop RJ, Andrews SE, Kuhl FP. Three-dimensional shape analysis using moments and Fourier descriptors. IEEE Trans Pattern Anal Mach Intell 1988; 10(6):937–943.
34. Russ JC. The Image Processing Handbook, 5th ed. Boca Raton, FL: Taylor & Francis, 2006.
35. Sánchez-Marín FJ. Automatic recognition of biological shapes with and without representations of shape. Artif Intell Med 2000; 18(2):173–186.
36. Shelly D, Goggin D, Janson SD. Shape features for recognition of pap smear cells. Proc SPIE 1996; 2823:224–235.
37. Shen L, Rangayyan RM, Desautels JL. Application of shape analysis to mammographic calcifications. IEEE Trans Med Imaging 1994; 13(2):263–274.
38. Sonka M, Hlavac V, Boyle R. Image Processing, Analysis, and Machine Vision, 2nd ed. Pacific Grove, CA: Brooks/Cole, 1998.
39. Wang PC, Lin KP, Lou SA, Lin HD, Chen TS. 3D MR image segmentation of prostate gland using two scan planes and Fourier descriptor technique. Proc SPIE 1999; 3661:1047.
40. Wehrli FW, Gomberg BR, Saha PK, Song HK, Hwang SN, Snyder PJ. Digital topological analysis of in vivo magnetic resonance microimages of trabecular bone reveals structural implications of osteoporosis. J Bone Miner Res 2001; 16(8):1520–1531.
41. Yeni YN, Zelman EA, Divine GW, Kim DG, Fyhrie DP. Trabecular shear stress amplification and variability in human vertebral cancellous bone: relationship with age, gender, spine level and trabecular architecture. Bone 2008; 42(3):591–596.
42. Zahn CT, Roskies RZ. Fourier descriptors for plane closed curves. IEEE Trans Comput 1972; 21(3):269–281.
43. Zhang T, Suen C. A fast parallel algorithm for thinning digital patterns. Commun ACM 1984; 27:236–239.
44. Zheng Y, Du J, Qi L, Yu D. Echocardiographic visual processing and quantitative analysis. In: Mu G, Jin G, Sincerbox GT, editors. International Conference on Holography and Optical Information Processing (ICHOIP '96), Nanjing, China, Dec. 31, 1996; 2866(1):46–49.

10

FRACTAL APPROACHES TO IMAGE ANALYSIS

Many natural objects have irregular, seemingly complex shapes. Yet often those complex shapes are assembled from simple structures. A good example is the fern leaf shown in Figure 10.1. This seemingly complex structure is made from small leaflets arranged left and right along a small stem. Many of these stems grow up and down from a larger, somewhat horizontally oriented stem. These horizontal stems, in turn, grow right and left from the main stem. At each level, the structure is similar to the whole: One of the branches growing sideways off the main stem, if turned by 90° and magnified, would look like the whole fern leaf, a phenomenon called *self-similarity*. In the human body, a good example are the lungs, where the descending trachea branches into the bronchial trees of the left and right lungs. Each bronchial tree consists of repeatedly (iteratively) smaller bronchial tubes branching off larger ones. The arterial and venous blood vessels, in the liver or kidney follow a similar rule: A major supply blood vessel splits into smaller vessels, which in turn split into even smaller vessels down to the level of the capillaries. The concept of self-similarity is widespread in living systems. In many cases, as in the example of the fern, a set of rules can be formulated that generates the complex object from its primitive structural elements. As explained later in this chapter, this set of rules is associated with a metric, the fractal dimension, which can be interpreted intuitively as a metric of the complexity of the shape. Shapes and textures occurring in medical images can be classified by using methods that estimate the fractal dimension, and since the early 1990s, the use of fractal approaches has become a standard method in

Advanced Biomedical Image Analysis, By Mark A. Haidekker

FIGURE 10.1 A seemingly complex structure such as a fern leaf is a complex arrangement of simple structures.

quantitative image analysis. Despite its popularity, the use of the fractal dimension to describe complexity in medical images depends on various factors. Depending on the image modality, image processing steps, and the method to estimate the fractal dimension, highly diverging results may be obtained. A thorough understanding of mathematical fractals, the property of self-similarity, and image operators to estimate fractal properties are needed.

10.1. SELF-SIMILARITY AND THE FRACTAL DIMENSION

A mathematical fractal is obtained by iterative application of the *Hutchinson operator*. A Hutchinson operator, w, takes an arbitrary object and arranges several transformed (usually reduced) copies of the object. In mathematical terms, application of the Hutchinson operator can be described by

$$w(S) = \bigcup_{i=1}^{N} w_i(S) \tag{10.1}$$

where S is a nonempty bounded set in the embedding Euclidean space R^n, w_i describes the process of the transformation (reduction and translation) of each copy, and $\bigcup$ denotes the union of the subsets created by each of the N transformations. Repeated iterative application of the Hutchinson operator, $S_{k+1} = w(S_k)$, will, after an infinite number of iterations, lead to an attractor A:

$$A = \lim_{k \to \infty} S_k \tag{10.2}$$

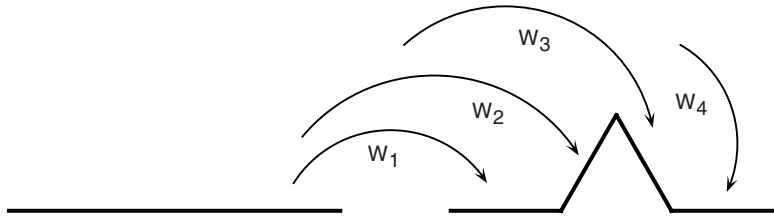


FIGURE 10.2 Example of a Hutchinson operator that creates four copies of the original object (a line of unit length), reduced in size to one-third and arranged in the manner depicted.

The attractor A is invariant under the specific Hutchinson operator used for its creation [i.e., $A = w(A)$], and it is independent of the original set S. Let us examine one specific example for a Hutchinson operator. This example operator takes an initial set, for example, a line of unit length, and creates four copies, w_1 through w_4, each of length $\frac{1}{3}$. w_1 shifts its copy to the left, w_4 shifts its copy to the right, and w_2 and w_3 shift and rotate the copy (Figure 10.2).

Repeated application of this specific Hutchinson operator creates a more and more jagged line (Figure 10.3). The attractor of this operation is called the *Koch curve* (named after Swedish mathematician Helge von Koch, who introduced this fractal object in 1904). It is interesting to observe that the Koch curve is infinitely long. The Hutchinson operator that generates the Koch curve arranges four copies of the original line (of unity length) that are reduced to one-third of its original size. The total length of the line at the first iteration, $k = 1$, is therefore $\frac{4}{3}$. At the kth iteration, the length is $(\frac{4}{3})^k$, which goes toward infinity for $k \to \infty$. Yet the curve is still bounded by the same length in both the horizontal and vertical directions. Furthermore, the Koch curve is self-similar; that is, any section of the curve, magnified, looks like the whole curve.

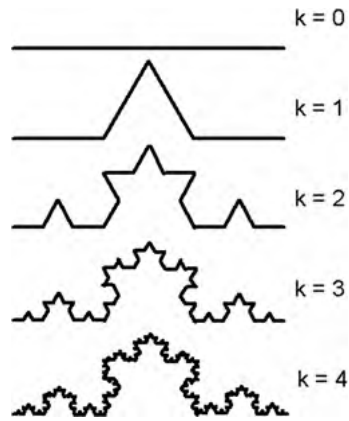


FIGURE 10.3 Iterative steps to generate a Koch curve.

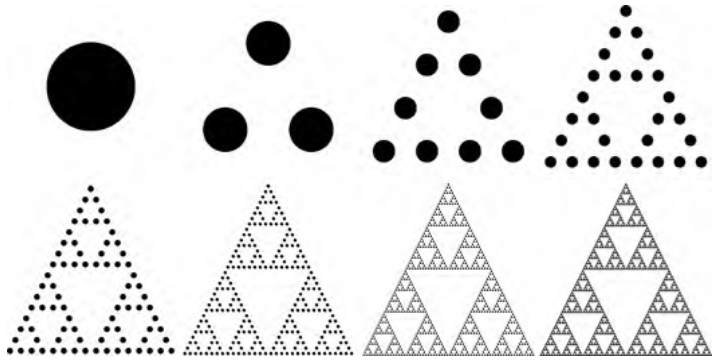


FIGURE 10.4 Iterative steps to construct the Sierpinsky gasket. The Hutchinson operator arranges three copies, which are reduced by a factor of 2, in a triangular pattern.

A different Hutchinson operator generates another well-known fractal, the *Sierpinsky gasket*. In this case, three copies, reduced by one-half, are arranged in a triangular pattern. Figure 10.4 shows eight iterations toward the Sierpinsky gasket. Notably, the attractor (the Sierpinsky gasket) is independent of the original set S, and in this example, the iteration starts with a circle as the initial bounded set S to demonstrate this property. If the original set has a total surface area of A, each of the reduced copies has a surface area of $A/4$ because of the factor-of-2 reduction of each side. Since three copies are arranged, the first iteration's surface area is $3A/4$. Analogous to the length of the Koch curve, the surface area changes by a factor of $(\frac{3}{4})^k$ for the kth iteration. Evidently, the area occupied by the Sierpinsky gasket vanishes for $k \to \infty$. Although this fractal is embedded in a plane (the Euclidean two-dimensional space), one would intuitively not consider it a two-dimensional object because of its vanishing surface area. Yet it is not one-dimensional. A more rigid mathematical approach uses the scaling laws defined by the Hutchinson operator. In one-dimensional space, a reduction of a line by a factor of s allows us to place s scaled lines side by side within the same space. Similarly, in two-dimensional space, the reduction of a square by a scale factor of s allows us to place s^2 of the reduced objects within the same area, and in three-dimensional space, s^3 cubes can be placed within the same volume. The construction rule of the Sierpinsky gasket allows us to place $a = 3$ objects, scaled by $s = \frac{1}{2}$, in the same space, and for the Koch curve, $a = 4$ objects, scaled by $s = \frac{1}{3}$, may be placed in the same space. This relationship can be generalized with a power-law rule:

$$a = \frac{1}{s^D} \tag{10.3}$$

Here D coincides with the Euclidean dimensions 1, 2, and 3 for lines, squares, and cubes. For fractal objects, Equation (10.3) can be solved for D, and we arrive at the

definition of the self-similarity dimension D:

$$D = \frac{\log a}{\log(1/s)} \tag{10.4}$$

where a is the number of copies arranged and s is the reduction factor. For the Koch curve ($a = 4$ and $s = \frac{1}{3}$), the self-similarity dimension is $D = \log 4/\log 3 \approx 1.26$, whereas the Sierpinsky gasket ($a = 3$, $s = \frac{1}{2}$) has a self-similarity dimension of $D = \log 3/\log 2 \approx 1.59$. The dimensions are noninteger, leading to the term *fractal*. The noninteger dimensions of the two example fractals are intuitively acceptable. The Sierpinsky gasket, as identified before, is not really two-dimensional with its vanishing area. It is clearly not a one-dimensional object, though. With a dimension of approximately 1.59, it lies between Euclidean one- and two-dimensional objects. The same is valid for the Koch curve, with an infinitely long line folded in two-dimensional space. However, the Koch curve is more closely related (intuitively) to a line; hence its dimension is closer to 1, whereas the dimension of the Sierpinsky gasket is closer to 2.

Many natural objects have self-similar properties. The silhouette of a mountain range or the shoreline of a lake, even erosion patterns in volcanic rock, appear self-similar; that is, smaller sections, when enlarged, look like the whole. In nature, however, there is an aspect of randomness involved. Therefore, natural self-similar objects do not follow the strict rules applied for the construction of mathematical fractals. The inclusion of a random element in the Hutchinson operator is known to create naturelike shapes. In fact, artistic renderings of synthetic landscapes of strange beauty make use of random-based fractal generators (see, e.g., the comprehensive book on procedural techniques by Ebert et al.,[19] which makes extensive use of fractal methods). The principle of the process is demonstrated in Figure 10.5. The starting shape is a hexagon. For the first iteration, each of the six sides is split in two and the midpoint of each side is displaced by a random distance. At each iteration, the number of sides doubles. After a sufficiently large number of iterations, this process generates a shape resembling the coastline of an island. The key to creating a self-similar, fractal-like shape is the scaling element. The random numbers that determine the displacement must be multiplied (i.e., reduced) by a factor of 0.5^{kH} at the kth

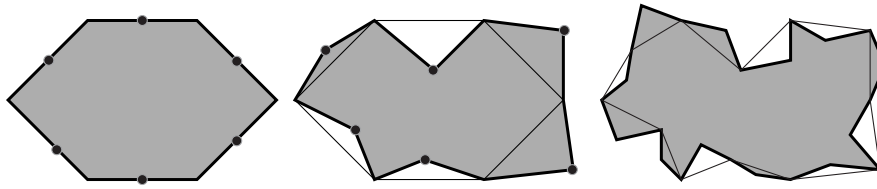


FIGURE 10.5 Iterative generation of the coastline of an island. The starting shape is a hexagon. From one iteration to the next, each side is split in two and the midpoint (the new vertex) is displaced a random distance. With each iteration, the coastline becomes more jagged and irregular.

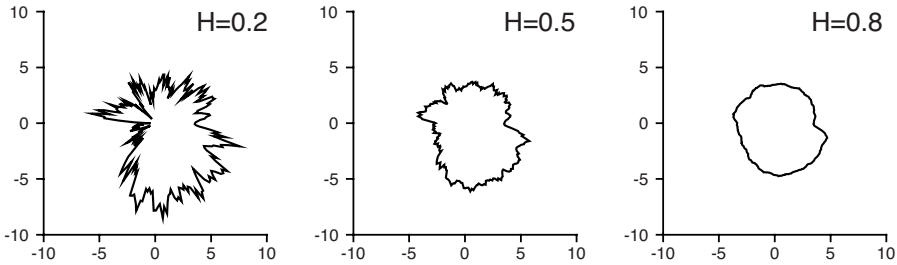


FIGURE 10.6 Islands with coastlines generated by random walk to demonstrate the influence of the Hurst exponent H. With a small Hurst exponent, the coastline appears more jagged. A large Hurst exponent (close to 1) creates a relatively smooth curve.

iteration. H, the *Hurst exponent*, is a factor that determines the overall roughness of the shape, with $0 < H < 1$. Hurst exponents close to 1 generate relatively smooth outlines, whereas Hurst exponents closer to 0 generate jagged and irregular outlines, as demonstrated in Figure 10.6.

The same principle can be applied to obtaining three-dimensional landscapes. The starting object is a flat shape such as a rectangle. The rectangle is then subdivided into four triangles and each of the vertices is displaced by a random height. Each of the triangles is again subdivided, and each new vertex is displaced in height. This process is repeated iteratively several hundred times. A representative result of this process is shown in Figures 10.7 and 10.8. The displacement map can be represented in gray scale, resulting in a gray-scale image such as that shown in Figure 10.7A. A suitable false-color mapping combined with contour lines creates the topographical map in Figure 10.7B. Figure 10.8 is a three-dimensional rendering of the landscape with several added features: The landscape is colored based on the elevation, a sea

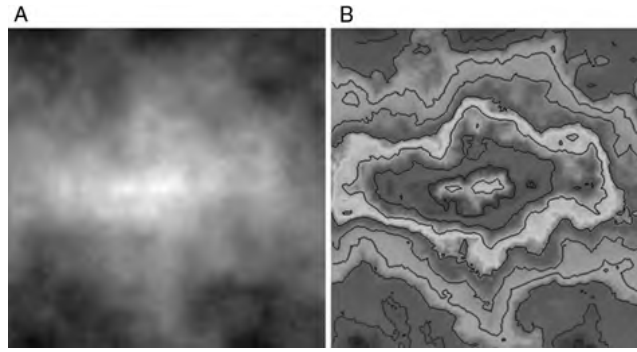


FIGURE 10.7 Landscape generation using the iterative midpoint displacement technique. A shows the random displacements as gray values (light gray areas have a higher elevation than dark gray areas). With suitable false coloring and added contour lines, a topographical map is created (B). (*See insert for color representation of the figure*.)

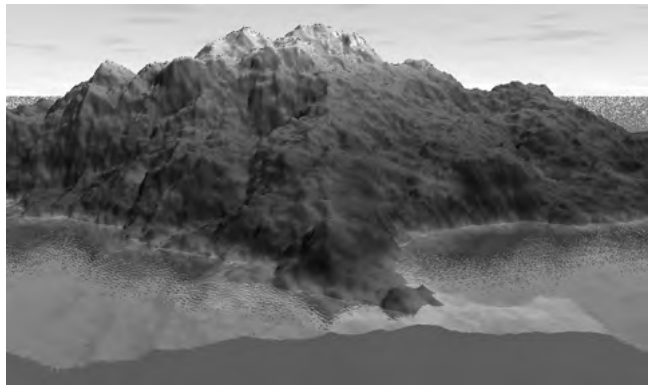


FIGURE 10.8 Three-dimensional rendering of the topographical map shown in Figure 10.7, with elevation-dependent colors, texture-mapped sea level, and texture-mapped sky added. (*See insert for color representation of the figure*.)

level is defined (i.e., a threshold), and any gray value below the sea-level threshold is rendered as submerged; the water surface has received a mapped texture, and the sky is rendered with a mapped cloud texture.

These considerations pave the way for the analysis of medical images using fractal analysis tools. In a prominent example, the shape of lesions found in x-ray mammography images has been linked to their degree of malignancy.[59,64] Irregular, particularly spicular, outlines indicate malignancy, whereas a more regular outline indicates a benign process (Figure 10.9). A strong relationship of the lesion outlines to the islands in Figure 10.6 is evident.

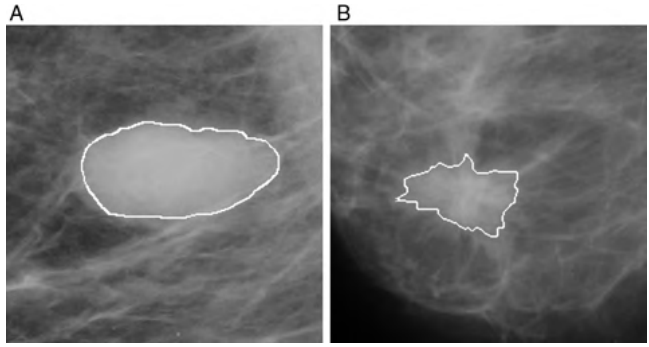


FIGURE 10.9 Abnormal masses in x-ray mammographies. (A) shows a circular, benign mass with regular outlines; (B) shows a spiculated mass with its irregular outlines. Irregular boundaries often indicate malignancy.

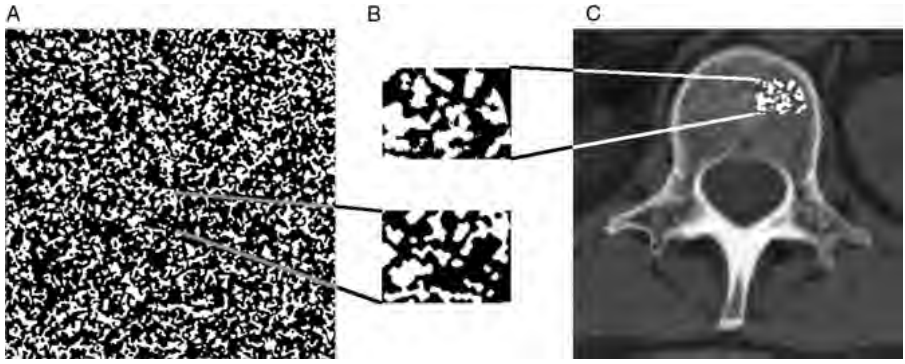


FIGURE 10.10 Similarity of the pattern of segmented trabecular bone (C) and a fractal generator that iteratively uses a neighborhood majority rule to set or reset a pixel (A). A small section of trabecular bone has been thresholded and magnified, and juxtaposed with an enlarged section of the fractal structure to demonstrate similarity (B).

A second image example for potential fractal properties are x-ray or CT images of trabecular bone. The fractal nature of trabecular bone has been subject to intense discussion (see e.g., references 12, 24, 42, and 48), but a number of studies that relate deterioration of trabecular bone to a change in fractal dimension are presented in Section 10.6. Let us consider an algorithmic process to create a fractal structure with a random element. On a square lattice filled with black pixels, white pixels are randomly seeded with a probability of about 50%. Next, we analyze each pixel and count the number of white pixels in a 3×3 neighborhood. If the majority of the pixels (i.e., five or more out of nine) are white, the central pixel will become white at the end of the analysis run. Otherwise (i.e., if the majority of the pixels is black), the central pixel will also be turned black. This run is now repeated iteratively until the pattern converges. A typical pattern that would result from this process is shown in Figure 10.10A. The pattern of segmented trabecular bone (Figure 10.10C) appears strikingly similar, as demonstrated in the magnified sections in Figure 10.10B.

Since natural objects with self-similar properties contain random elements, no construction rule exists to compute the fractal dimension analogous to the self-similarity dimension that we used on the Sierpinsky gasket and the Koch curve. It will therefore be necessary to investigate estimators of the fractal dimension, that is, numerical methods to assess self-similar properties in arbitrary images. The common principle of all estimators is to find the scaling rule (i.e., the power law) by taking iterative measurements at different scales. The principle can be demonstrated using the *compass dimension*. The length of a rugged outline is measured with different compass settings. As the compass settings become smaller, more detail is measured. A well-known example is the question of the length of the coastline of Great Britain.[51] A compass can be used to determine the approximate, coastline length. In the example shown in Figure 10.11, a compass setting of 120 km leads to a polygon with 24 sides,

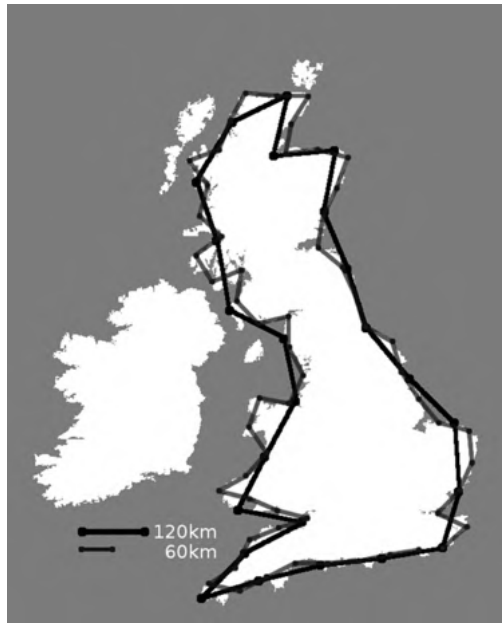


FIGURE 10.11 Determining the length of the coastline of England with different compass settings.

corresponding to 2880 km. With a 60-km compass setting, the compass can follow more details of the coast, and the new polygon has 56 sides and one incomplete side, corresponding to approximately 3390 km. In fact, with each smaller setting of the compass, more detail is captured, and the apparent length of the coastline increases. If we assume self-similar properties, we can expect the data pairs of compass setting s and measured length l to follow the power law described by

$$l = \frac{1}{s^D} \tag{10.5}$$

In Equation (10.5), D is the estimate of the fractal dimension. A similar power law was introduced in Equation (10.3). D can be determined by taking the logarithm of Equation (10.5) and thus obtaining the equation of a straight line that has the slope D and goes through the origin:

$$\log l_k = -D \log s_k \tag{10.6}$$

where l_k and s_k are the data pairs of compass setting s and the resulting length l for a number of different compass settings. D can now be determined by linear regression.

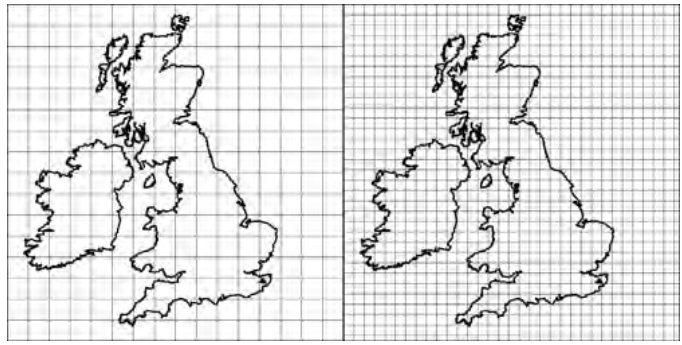


FIGURE 10.12 Determination of the box-counting dimension. The image is subdivided into squares of size s, and the number of squares that contain at least one pixel of the feature (in this case, the coast of England) is counted. Scaling properties are determined by repeating this process with different box sizes.

10.2. ESTIMATION TECHNIQUES FOR THE FRACTAL DIMENSION IN BINARY IMAGES

10.2.1. Box-Counting Dimension

The box-counting algorithm is arguably the most widely used method to estimate the fractal dimension. The reason for its popularity lies in its simple computer implementation and its wide applicability to features with or without self-similarity. Furthermore, the box-counting algorithm can be carried out in two or three dimensions. To compute the box-counting dimension, the image is subdivided into equal-sized squares of side length s. Under the assumption of a binary image [background of value zero, image object(s) of nonzero value], all boxes that contain at least one nonzero pixel are counted. Then the box size s is increased by a certain factor, usually doubled (Figure 10.12). The resulting data pairs of box size s_k and counted boxes n_k are then log-transformed, and the box-counting dimension D_B is determined by linear regression:

$$\log n_k = -D_B \log s_k \tag{10.7}$$

Pseudocode that performs a box-counting measurement as described above is provided in Algorithm 10.1. The algorithm expects a binary image and returns a table of box sizes $\log s_k$ and corresponding box counts $\log n_k$. Linear regression into these data pairs yields the box-counting dimension D_B . To provide a few examples, let us examine the Sierpinski gasket (Figure 10.4), with known exact self-similarity dimension, and the coastline (Figure 10.12), a random fractal with no known self-similarity dimension. Table 10.1 shows the box-counting results.

TABLE 10.1 Box-Counting Results for the Sierpinski Gasket and the Coastline

	Number of Boxes Counted			Log (Number of Boxes Counted)	
Box Size s	Gasket	Coastline	Log (1/s)	Gasket	Coastline
1	26,244	12,336	0	4.419	4.091
2	8,748	5,922	−0.301	3.942	3.772
4	2,916	2,595	−0.602	3.465	3.414
8	972	1,132	−0.903	2.988	3.054
16	324	470	−1.204	2.511	2.672
32	108	193	−1.505	2.033	2.286
64	36	80	−1.806	1.556	1.903
Regression slope (= box-counting dimension D_B)				1.585	1.220
				$r^2 = 1.000$	$r^2 = 0.9991$

```
set s=1;                                  // initial (smallest) box size
set iterations=5;                         // Choose scaling range
set itercnt=0; set k=0;
allocate lnb[5], ls[5];

while (itercnt < iterations) do   // main loop over the box sizes
   n=0;
   for (y=0 while y<ym increment y=y+s) do // double loop over the image
      for (x=0 while x<xm increment x=x+s) do

        // Double loop over each box to determine if a pixel is set

        set flag=0;
        for (y1=y while (y1<min(ym,y+s) and not flag) increment y=y+1)
           for (x1=x while (x1<min(xm,x+s) and not flag) increment x=x+1)
              if (IM(x1,y1)>0) then flag=1;
           endfor;
        endfor;
        if (flag>0) then n=n+1;   // at least one pixel set; count this box
      endfor;
   endfor;
   if (n=0) then                          // avoid log(0), rather omit the data point
      ls[k]=LOG(s);                       // Store size and box count data . . .
      lnb [k]=LOG(n);                     // . . . in double-log form for later regression
      k=k+1;
   endif;
   itercnt = itercnt+1;
   s=s*2;                                 // Double the scale, i.e., the box size
endwhile;
```

Algorithm 10.1 Box-counting dimension. The input image `IM(x,y)` is assumed to have the pixel dimensions `xm` and `ym`. A value of zero indicates a background pixel. The output are two corresponding tables of log box size `ls()` and log number of boxes `Lnb()`. The box-counting dimension D_B is computed by linear regression and is the slope of the regression line into `lnb` over `ls`.

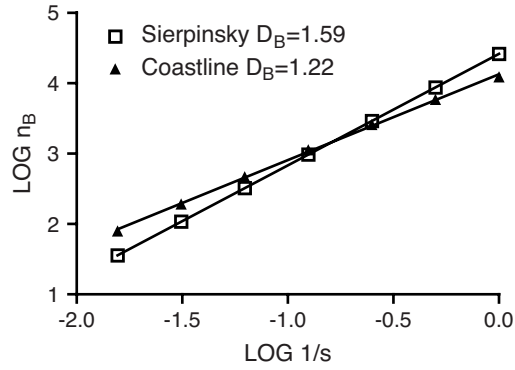


FIGURE 10.13 Graphical representation of the data in Table 10.1. The slope of the fitted data represents the box-counting dimension D_B. In this example, the quality of the fit is extremely high, with r^2 close to unity.

The graphical representation of the data in Table 10.1, shown in Figure 10.13, indicates a very good fit of a straight line in the log-log plot. It is crucial to verify the goodness of this fit, since several factors can affect the box-counting results. Most important, scaling limits exist: Box sizes smaller than the pixel size (theoretically possible through interpolation) will not yield reasonable results. At the other end of the scale, the data points will no longer fit the straight line either. If we superimpose four boxes over the Sierpinsky gasket (Figure 10.4), all four boxes will be counted. However, by merit of the construction rule, we know that the box count should be three. When the scale is reduced by a factor of 2 (a total of 16 boxes), 12 boxes would be counted where the construction rule allows for only 9. Only when the box size is much smaller than the object itself do the counts match the values predicted theoretically. The scaling rule fails in a manner similar to the coastline example. If we continue the box-counting process toward larger boxes, with the coastline as the object, a deviation can be observed as the data points follow a lower slope. A similar example is the fractal structure in Figure 10.10A. When the box size becomes larger than 8 (with a total image size of 512), all boxes are counted. For box sizes of 1, 2, 4, and 8, the resulting dimension is $D_B = 1.6$, whereas a box-counting measurement that starts with a box size larger than 8 yields $D_B = 2.00$, which is the Euclidean dimension of a plane. With unsuitably large box sizes, almost any value between 1.6 and 2 can be obtained.

In medical images, box counting at the pixel level will be confounded by the presence of noise. Gaussian noise contributes to the box-counting result on small scales. However, the r^2 value of the linear regression may still be close to 1.00 and does not necessarily give an indication of a result confounded by noise. It is therefore important to examine closely at what scales self-similar properties can be expected and to select the box size range accordingly. Different modalities (e.g., a CT image and an x-ray image) of the same object may produce different dimension results, often as a result of different resolution. Even different exposure rates (e.g., x-ray

images) may result in different values for D_B. For this reason, many studies compare only groups with image acquisition parameters that are as consistent as possible.

The positioning of the box origins relative to the image features has a strong influence on the results as well. The example of the Sierpinsky gasket in Table 10.1 shows a perfect match with the self-similarity dimension only if the fractal object is aligned perfectly with the boxes. Shifting box origins relative to the features may produce biased values for D_B that may be either too large or too small. The presence of multiple features (as in Figure 10.10A) may also lead to questionable results. Each feature may have its own self-similarity property, whereas the box-counting method yields only a single value. This value may not be characteristic for the structure analyzed. In this case it might be preferable to analyze each feature separately or to analyze the local scaling behavior and to examine the composition (histogram) of local dimensions of an image. This behavior is analyzed in more detail in Section 10.5.

10.2.2. Minkowsky Dimension

The *Minkowsky dimension algorithm*, also known as the capacitor dimension, uses morphological dilations to estimate scaling properties. The algorithm starts with the outline of the feature or image to be measured. Each pixel is then expanded into a filled circle of radius r, where r increases with each iteration. Some pixels of an expanded circle overlap with pixels belonging to neighboring circles. The total number of pixels gained (overlapping pixels are only counted once) is determined as a function of the dilation radius r, resulting in data pairs of n_p (the number of pixels) over r. The scaling law for the pixel gain is

$$n_p = r^H \tag{10.8}$$

where H is the Hurst exponent. In two-dimensional space, the fractal dimension D is related to the Hurst exponent H through

$$D = 2 - H \tag{10.9}$$

In the computer implementation, the initial feature outline is subjected to morphological dilation repeatedly and iteratively. For each dilation, the corresponding radius is identical to the iteration number (i.e., $r_k = k$), and the pixels gained for iteration k, that is, $n_{p,k}$, can be determined by counting the total pixels of the original image, subtracted from the dilated image. For the data pairs r_k and $n_{p,k}$ obtained from several dilations, the Hurst exponent H and the fractal dimension D can now be determined by linear regression of the log-transformed data:

$$\log n_{p,k} = H \log r_k = H \log k \tag{10.10}$$

Pseudocode for a sample implementation of the Minkowsky dimension is shown in Algorithm 10.2. A few examples illustrate the process further. Individual dots, subjected to iterative dilations, expand to squares of 9, 25, 49, ... total pixels. After one iteration ($k = 2$), the dilation operator has expanded the dots from 1 to 9 (by

$n_p = 8$) pixels. This is the first data pair, (2,8). Subsequent iterations yield data pairs of (3,17), (4,32), (5,49), and so on. A linear fit into the log-transformed data yields a slope of $H \approx 2$ and therefore a dimension of $D = 0$, which coincides with the Euclidean dimension of the dot: namely, zero. A straight line would dilate into a "sausage" of thickness $2r$. Its area would be approximately $2r\ l$, where l is the length of the line. The area scales with an exponent of $H = 1$, giving the correct Euclidean dimension of 1 for the line. An example of a jagged line is presented in Figure 10.14. As an additional example, we apply the Minkowsky algorithm to the coastline of England (Figure 10.11). Over a large number of dilations, $H = 0.74$ is obtained, resulting in a Minkowsky dimension $D_M = 1.26$. This value is in acceptable agreement with the dimension $D_B = 1.22$ obtained through box counting.

The Minkowsky dimension method is less sensitive than the box-counting method to the placement of features. However, the Minkowsky dimension algorithm fails for filled two-dimensional shapes. Since the dilation process can only grow outward from the shape, the pixel gain is similar to the gain obtained with a one-dimensional

```
set k=0; set intercnt=0;
set iterations=5;
allocate logrd[5], loga[5];
allocate IM2(xm,ym);                          // the dilated image

while (itercnt < iterations) do               // main loop over iterative dilations
   IM2 = IM;                                  // Copy IM into IM2
   dilate (IM2);                              // and dilate the copy (defined external to
                                              this function)

   set cnt=0;
   for (y=0 while y<ym increment y=y+1) do     // double loop over the image
      for (x=0 while x<xm increment x=x+1) do  // to count dilated pixels
         If (IM(x,y) != IM2(x,y) ) then cnt=cnt+1;
      endfor;
   endfor;

   if (cnt>0) then                            // avoid log(0), rather omit the data point
      logrd[k]=LOG(itercnt+1);                // Store size and pixel count data...
      loga [k]=LOG(cnt);                      // ... in double-log form for later regression
      k = k+1;
   endif;
   itercnt = itercnt+1;
   IM = IM2;                                  // Prepare dilated image for next iteration
endwhile;
```

Algorithm 10.2 Minkowsky dimension. Similar to the box-counting method, the input image `IM(x,y)` is assumed to have the pixel dimensions `xm` and `ym`. A value of zero indicates a background pixel. This function depends on morphological dilation, to be defined elsewhere. The output consists of two corresponding tables of log dilation radius `logrd()` and log dilated area `loga()`. The Minkowsky dimension D_M is computed by linear regression. H is the slope of the regression line into `loga` over `logrd`, and $D_M = 2 - H$.

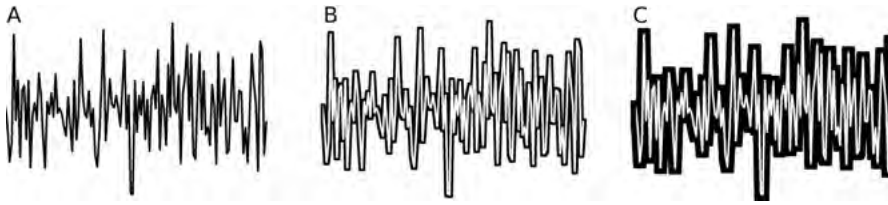


FIGURE 10.14 Iterative dilation of a jagged line to determine the Minkowsky dimension. Shown are the original line (A) and the difference image after 10 (B) and 24 (C) dilations. It can be seen that a larger dilation radius causes filling of the jagged parts, so that the sensitivity to detail is diminished after several dilations. This property defines the ability of the method to determine scaling behavior.

shape (outline). In the coastline example, a dimension $D_M = 1.26$ would be obtained irrespective of whether the shape of England is filled or the outline is used. Conversely, the box-counting method would yield a dimension of 1.83 for the filled shape as opposed to 1.22 for the outline.

10.2.3. Mass Dimension

The *mass dimension*, sometimes referred to as the *sandbox dimension*, is typically used on shapes with a clear center point, such as circular shapes or shapes with some level of rotational symmetry. Good examples are growth or aggregation processes that lead to dendritic structures growing out from an initial center point.[21] Examples include neuronal patterns[22] and structures found in melting carbohydrates.[60] These structures bear a strong similarity to structures generated with a computer simulation, introduced by Witten and Sander.[71] This type of simulated structure has been well explored, and it is known that they exhibit a fractal dimension of 1.7 over a wide scaling range with the mass dimension being the most suitable algorithm to use to estimate their fractal dimension.[33,50,54] In fact, both the box-counting method and the Minkowsky method underestimate the dimension and yield values for D closer to 1.5.

Starting with this central point, the number of pixels of the feature inside a circle with radius r is determined. Then the radius is increased and the number of enclosed pixels is determined again. The result is a series of data pairs of counted pixels, n_p, as a function of the radius r. If the object underneath the expanding circle has self-similar properties, the number of pixels obeys a power-law scaling function:

$$n_p = r^D \tag{10.11}$$

In Equation (10.11), D is the mass dimension and is determined in a fashion identical to the box-counting method by log-transforming Equation (10.11) and fitting a regression line into the data pairs of $\log n_p$ and $\log r$. The slope of this line is D.

Figure 10.15 shows the use of circles of increasing radius to count enclosed particles. The corresponding plot of the particle count as a function of the circle

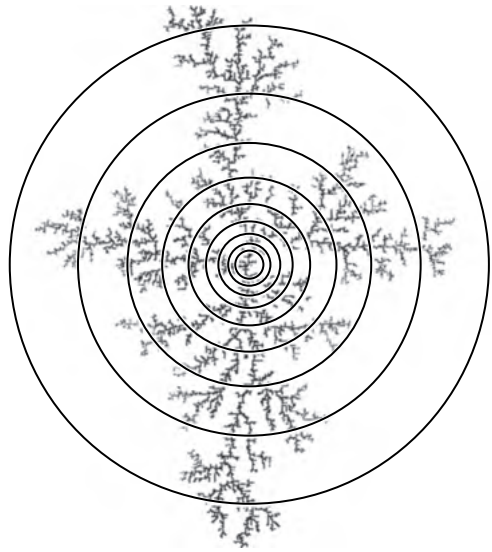


FIGURE 10.15 Determining the mass dimension of a simulated aggregation cluster. The mass (nonbackground pixels) of the aggregate inside the circles of increasing radius is counted, and the slope of the mass over the radius is determined on a double-logarithmic scale. (*See insert for color representation of the figure*.)

radius is shown in Figure 10.16. Similar to all other fractal estimation methods, the mass dimension is sensitive to the choice of scales. As can be seen in Figure 10.16, the power-law relationship between particle count and circle size fails when the circle size reaches the aggregate size.

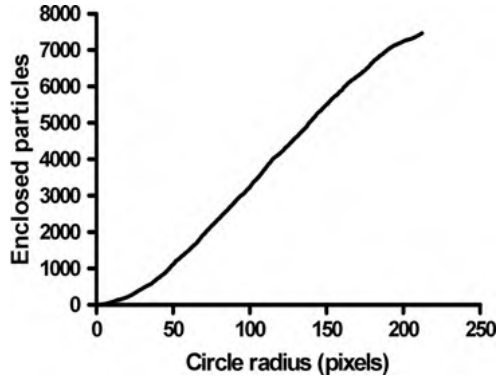


FIGURE 10.16 Number of particles enclosed in a circle plotted as a function of the circle diameter. This example shows the scaling limits particularly well. Whereas the particle count inside small circles (up to a radius of approximately 150 pixels) follows a power law with an exponent D of 1.74, the power law fails with large diameters once almost all of the aggregate is enclosed.

Unlike the Minkowsky method, the mass dimension is very sensitive against translations. A different choice of the circle's center point may lead to highly different dimension values. At the same time, this property can be construed as an advantage, because it allows us to analyze a structure's local properties. In fact, a mass dimension can be computed for each feature in a segmented image, or even for each pixel. In the second case, local fractal dimensions give rise to the concept of multifractals. These properties are examined more closely in Section 10.5.

Algorithm 10.3 explains how the mass dimension is computed for a single feature in the image. Variations exist where square regions are examined instead of circular regions, and the radius step may either be exponential as in Algorithm 10.3 or linear. These variations will not provide fundamentally different results.

```
set k=0;
set iterations=5;
set itercnt=0;
set radius=10;                              // initial radius
allocate logrd(5), logm(5);

set xc=xm/2; set yc=ym/2;                   // center of image

while (itercnt < iterations) do             // main loop over iterative dilations
   mass=0;
   for (y=yc-rad while y<=yc+rad increment y=y+1)
      for (x=xc-rad while x<=xc+rad increment x=x+1)
         if (SQR(x-xc)+SQR(y+yc) < SQR(rad)) then    // tests circle
            if (IM(x,y)>0) mass=mass+1;
         endif;
      endfor;
   endfor;

   if (mass>0) then          // precaution: avoid log(0); rather, omit data point
      logrd[k] = log(radius);
      logm[k] = log(mass);
      k=k+1;
   endif;

   itercnt=itercnt+1;
   radius = radius*1.4;     // exponential increase of radius

endwhile;
```

Algorithm 10.3 Mass dimension. Similar to the box-counting method, the input image `IM(x,y)` is assumed to have the pixel dimensions `xm` and `ym`. A value of zero indicates a background pixel. For the mass dimension, a suitable choice of the circle center `(xc,yc)` is crucial. In this example, the feature to be examined is assumed to be centered with respect to the whole image.

10.3. ESTIMATION TECHNIQUES FOR THE FRACTAL DIMENSION IN GRAY-SCALE IMAGES

In Section 10.2, estimation methods for the fractal dimension were introduced that work on sets. To obtain a set, the image must be segmented, that is, divided into the feature (set) and background. The notion of fractal dimension can be extended to gray-scale images. In this case, additional information from the image (the gray-value information) is retained. Whereas binary images allow quantitative analysis of the shape of a feature, the methods that operate on gray-scale images focus more on the texture. As demonstrated in Figure 10.8, the gray value (the image value) of a pixel can be interpreted as elevation. The image can be thought of as a mountainscape, with white pixels being the points with the highest elevation and dark regions being the valleys. In this interpretation, the image becomes a surface embedded in three-dimensional space, much as a jagged line (such as the Koch curve) is embedded in two-dimensional space. Correspondingly, the resulting value of the fractal dimension can be expected to lie between two and three.

10.3.1. Blanket Dimension

The most widely applied estimator of the fractal dimension in gray-scale images is a variation of the box-counting dimension often referred to as the *blanket dimension*. The surface of the landscape is tightly covered with a blanket, and the surface area is computed. Each pixel forms a triangle with two adjoining pixels. Since each pixel has an elevation, the area of each triangle is different. The sum of all triangular areas is the total surface area. In the next iterative step, four adjoining pixels are averaged to form a single pixel. The averaged image now has twice the base length of the triangles. Since the averaging process has a smoothing effect, the averaged surface is less jagged. Therefore, an irregular surface will have a smaller total area of the triangles than that of the original image, although the size of the individual triangles has increased. This process is illustrated in Figures 10.17 and 10.18. Figure 10.17 shows a cross-sectional CT image of segmented trabecular bone and the corresponding three-dimensional landscape representation. In Figure 10.18, neighboring pixels have been averaged to form larger blocks of size 1 (original resolution), 2, 4, and 8. The resulting surface becomes less jagged with a larger block size, and the surface area decreases correspondingly. Analogous to Equation (10.7), the scaling law of the binary box-counting dimension, the surface area A will relate to the box size s through a scaling law with noninteger exponent H:

$$\log A_k = -H \log s_k \tag{10.12}$$

The surface dimension D_s related to H through $D_s = 2 - H$ [Equation (10.9)].

The slope H is obtained through linear regression of the log-transformed data pairs, the surface area A_k, and the corresponding size of the averaging box s_k, for each averaging step k. The log-transformed data pairs corresponding to the surface plots in Figure 10.18 and the regression line are shown in Figure 10.19. It can be seen

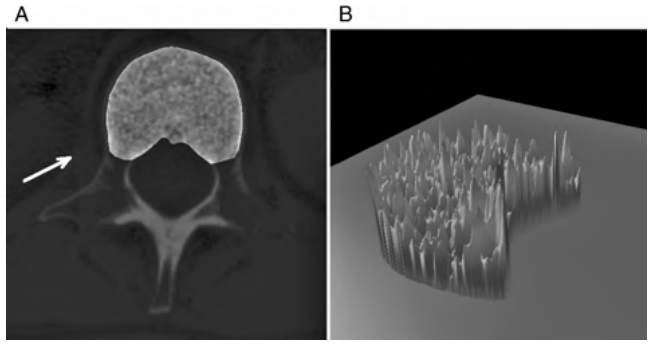


FIGURE 10.17 Elevation landscape representation of the segmented spongy area of a vertebral CT image. The left panel shows one cross-sectional CT slice through the vertebra with the spongy area highlighted. The right image is the corresponding elevation map. The white arrow indicates the view direction. (*See insert for color representation of the figure*.)

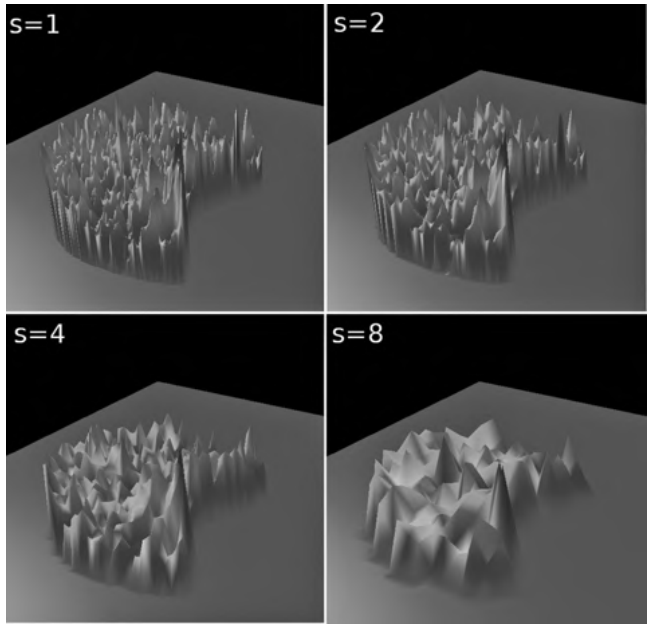


FIGURE 10.18 Four steps in the process of calculating the blanket dimension. Starting with the top left image, the landscape is drawn with one pixel per box ($s = 1$), whereas in subsequent iterations 4 pixels ($s = 2$), 16 pixels ($s = 4$), and 64 pixels ($s = 8$) are averaged. As a consequence, the elevation landscape becomes less jagged and the surface area decreases. (*See insert for color representation of the figure*.)

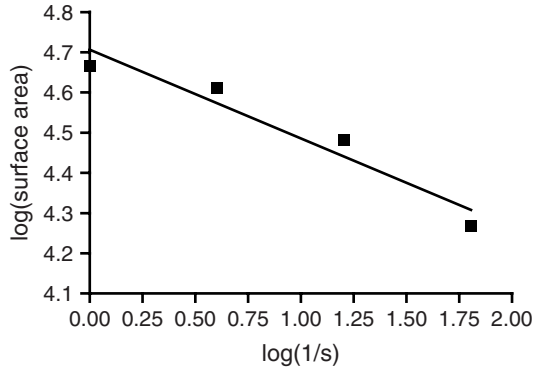


FIGURE 10.19 Double-logarithmic plot of the surface area as a function of the averaged box size, corresponding to $s = 1$ through $s = 4$ in Figure 10.18. The slope of the fitted line is -0.22, corresponding to a blanket dimension of 2.22. The line shows a poor fit, revealing that the surface area does not scale with the same factor at small box sizes.

that the scaling law does not hold over the box sizes analyzed. The scaling behavior needs to be examined properly. Several factors may account for a change of the slope over different scales. The presence of noise in medical images introduces its own contribution to the apparent dimension at small scales, because noise exists at the pixel level. Normally, noise would increase the apparent fractal dimension at very small scales. Conversely, if the resolution is higher than the image details, a reduction in the apparent fractal dimension at small scales is seen. Figure 10.19 is an example of this behavior. The loss of self-similarity at small scales could ideally be demonstrated using interpolated images. Because interpolation does not provide new information, the fractal dimension would be smaller at the interpolated scales. A careful choice of the scales at which the image is analyzed is essential for accurate estimation of the fractal dimension. Segmented objects with a jagged outline may contribute an additional element of self-similarity, because the outline itself influences the surface area calculation. An advanced algorithm used to compute the blanket dimension would therefore exclude any boxes that are not filled completely with pixels belonging to the feature.

Two principles are available for use in computing blanket dimensions. One principle uses the tessellation of the surface into triangles of increasing projected size s. This is the principle applied in Figure 10.18 and Algorithm 10.4. Similar results are obtained when the pixels are interpreted as elongated cuboids with a projected size in the xy-plane of $s \times s$ pixels and a height corresponding to the image value. The surface area of the sides is limited by the height of the neighboring pixels. Similar to Algorithm 10.4, the total surface area is computed and the scaling properties with increasing cuboid size are determined. This notion is often termed the *Manhattan dimension* because of the resemblance of the pixel cuboids to Manhattan skyscrapers. Note that Algorithm 10.4 requires the definition of a function to compute the area of a triangle given by its three vertices (`triangle_area`). If the vertex coordinates

```
set s=1;
set iterations=5;                          // Choose the scaling range
set itercnt=0;
allocate lnA[iterations], ls[iterations];
allocate IM2(xm,ym);                       // A scaled-down version of the image
set n=0;

while (itercnt < iterations) do      // main loop over the box sizes
   n=0;
   sarea=0;
   for (y=0 while y<ym-1 increment y=y+2) do    // double loop over the image
      for (x=0 while x<xm-1 increment x=x+2) do
        a1=triangle_area (0,0,IM(x,y), s,0,IM(x+1,y), 0,s,IM(x,y+1));
        a2=triangle_area (s,s,IM(x,y), s,0,IM(x+1,y), 0,s,IM(x,y+1));
        sarea = sarea+a1+a2;               // add to total surface area
        area=area+s;
        // Now create a scaled-down, averaged version of the image
        IM2(x/2,y/2)=0.25*(IM(x,y)+IM(x+1,y)+IM(x,y+1)+IM(x+1,y+1));
      endfor;
   endfor;
   IM=IM2;                         // Copy the scaled-down image to the main location
   if (sarea > 0) then
      ls[n]=LOG(area);             // Store log size data...
      lnA [n]=LOG(sarea);          // ... and log surface area for regression
      n=n+1;
   endif;
   itercnt = itercnt+1;
   s = s*2;                        // new scale factor
   xm=xm/2; ym=ym/2;               // new scaled-down image dimensions
endwhile;
```

Algorithm 10.4 Blanket dimension. In this basic version, the input image `IM(x,y)` with the pixel dimensions `xm` and `ym` is assumed to be completely filled by the feature, so that the feature boundary cannot introduce multifractal elements. The output consists of two corresponding tables of log box size `ls()` and log surface area `lnA()`. The blanket dimension D_s is computed by linear regression. With H being the slope of the regression line into `lnA` over `ls`, the dimension $D_s = 2-H$. This algorithm relies on a formula to compute the area of a triangle, given by its three vertices (`triangle_area`).

are designated x_i, y_i, and z_i with $1 \leq i \leq 3$ (and passed to the function in this order), the area A can be computed:

$$\begin{aligned} a &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \\ b &= \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2} \\ c &= \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2} \\ s &= 0.5(a + b + c) \\ A &= s(s - a)(s - b)(s - c) \end{aligned} \tag{10.13}$$

10.3.2. Gray-Scale Generalization of the Minkowsky and Mass Dimensions

The definition of the morphological dilation, which is the basis for the Minkowsky dimension, allows its application on gray-scale images. In gray-scale dilation, the center pixel is replaced by the maximum value of its 3×3 neighborhood. To determine the scaling behavior, the image mean value is calculated and recalculated after iterative dilations. The image mean value increases with each dilation, but the increase is less pronounced because of the loss of detail associated with each dilation. If a scaling behavior analogous to Equation (10.10) can be observed, with the exception that the number of pixels $n_{p,k}$ is now substituted by the image mean value, the measured exponent H can be used to compute the gray-scale Minkowsky dimension through $D_M = 2-H$. In an analogous definition of the mass enclosed by a circle of radius r, the number of pixels in Equation (10.11) needs to be substituted by the sum of the image intensity values. Algorithms 10.2 and 10.3 can be generalized to operate on gray-scale images with minimal changes. One advantage of generalized algorithms is elimination of the segmentation step. However, the gray-scale versions of algorithms report self-similar properties of the texture, whereas the binary versions report self-similar properties of the shape of the feature. Therefore, each of the two versions has specific applications.

10.4. FRACTAL DIMENSION IN THE FREQUENCY DOMAIN

The Fourier transform and its properties were discussed in Chapter 3. Of particular importance is the property of edges, boundaries, and irregularities in general to correspond to broadband frequency components. The underlying principle that allows a definition of a fractal dimension in the frequency domain is the question: How fast does the amplitude of the signal or image drop off toward higher frequencies? Intuitively, the frequency transform of an image with smooth features would show a steeper drop-off toward higher frequencies than would an image with many discontinuities and irregular features. Once again, the analysis of random processes lends itself to a rigid definition of the fractal dimension in the frequency domain. If the source of a random process produces values that are completely independent of each other, the random sequence is referred to as *white noise*. White noise has uniform local properties (i.e., its mean and standard deviation within a local window are widely independent of the window location). More important, white noise has a broadband frequency spectrum. In other words, each frequency has the same probability of occuring over the frequency range observed. In many random processes, however, consecutive samples are not completely independent. Rather, a trend can be observed that an increased value from one sample to the next is followed by a higher probability of additional value increases, an effect called *persistence*. The source is said to have a memory.

The origins of the analysis of the fractal dimension in the frequency domain lie in the analysis of random time series. H. E. Hurst observed water levels of the river Nile and found long-term dependencies (i.e., persistence). Moreover, Hurst found that the range of the water level scales with the duration of observation.[28] If the range R is defined as the difference between the highest level $L_{\max}(\tau)$ and the lowest level

$L_{\min}(\tau)$ in an observational period of length τ and σ is the standard deviation of the river levels during the period τ, then the range shows a power-law dependency on the length of the observational period:

$$\frac{R}{\sigma} \propto \tau^H \tag{10.14}$$

This scaling behavior of discrete time samples was found in many other areas, such as in financial data[52] and biology. One biomedical example is the potassium channel in pancreatic β-cells, where the opening and closing of the channels has been found to exhibit self-similar properties.[41] Moreover, the opening and closing of the channels have been found to exhibit memory effects, where an opening of the channel is followed by a higher probability of the channel opening again.[38] A scaling behavior as described in Equation (10.14) generally pertains to random events with a certain long-term memory. Such a series of events has been approximated with the model of fractional Brownian noise. For completely uncorrelated noise (white noise), $H = 0.5$, whereas persistence in the random time series yields $H > 0.5$ and antipersistence yields $H < 0.5$. A time series with persistence appears smoother. Equation (10.14) can be used to estimate the Hurst exponent from sampled data. Figure 10.20 demonstrates this process.

With a growing observational window, the window is more likely to contain extreme values, and the range R can be expected to grow. In the time domain, the scaling factor is estimated by rescaled range analysis. The time series of total length T is subdivided into $n = T/\tau$ observational windows of length τ. For each window, the rescaled range (minimum to maximum normalized by the standard deviation in that window) is determined and averaged. Iteratively, a new, longer τ is chosen and the average rescaled range is determined again. From a number of ranges R_i

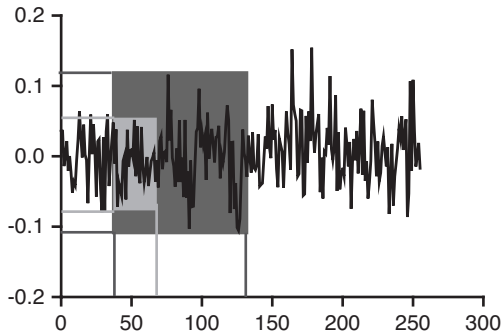


FIGURE 10.20 Random time series and two observational windows. A larger window (dark shade) has a higher probability of containing local extremes; therefore, the range (difference between highest and lowest value) is likely to be higher than in the shorter window (light shade).

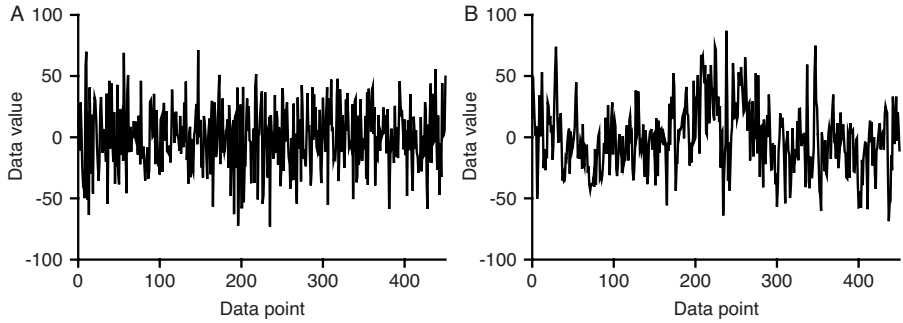


FIGURE 10.21 Uncorrelated Gaussian noise (A) and persistent noise of zero mean and matching standard deviation (B). Persistent noise exhibits large-scale, low-frequency trends and additional trends on smaller time scales and smaller amplitudes.

(τ_i) obtained at different window lengths τ_i, the scaling exponent H is obtained by nonlinear regression. The Hurst exponent is related to the fractal dimension by $D = 2-H$. This relationship is based on the scaling behavior of the amplitude of the Brownian time series and may not be valid for all time series. However, this relationship has been widely used and may be considered to be established.

Figure 10.21 shows two random time series of matching mean and standard deviation. Persistent noise (Figure 10.21B) exhibits trends on all time scales, and self-similar behavior can be expected. Conversely, uncorrelated Gaussian noise (Figure 10.21A) does not obey a scaling law. We have already quantified persistence by means of the Hurst exponent, which characterizes scaling behavior in the time domain. Low-frequency trends and scaling behavior can also be seen in the frequency domain: As opposed to broadband white noise, low frequencies dominate for persistent noise. More precisely, a time series that exhibits scaling behavior as described in Equation (10.14) also follows the frequency-domain scaling behavior described by,

$$A(\omega) \propto \omega^{-|\beta|} \tag{10.15}$$

where A is the amplitude of the frequency spectrum and ω the frequency. The negative sign in the exponent indicates an amplitude decay with higher frequencies. The scaling exponent, β, is related to the Hurst H exponent through

$$\beta = D_E - 2H \tag{10.16}$$

where D_E is the Euclidean dimension on which the signal is built (i.e., 1 for a time series and 2 for a two-dimensional image). The scaling exponent β in Equation (10.15) can be determined by fitting a regression line into the log-transformed data points of the spectral magnitude over the frequency as shown in Figure 10.22.

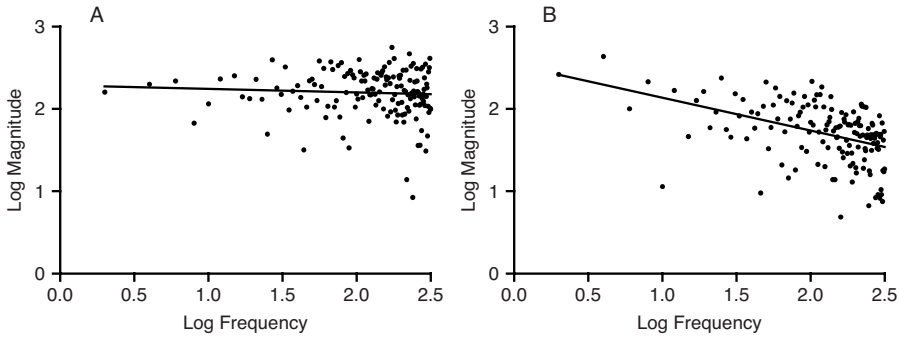


FIGURE 10.22 Magnitude of the frequency spectra of the signals in Figure 10.21. Uncorrelated Gaussian noise (A) shows no significant decay of the magnitude at higher frequencies (regression line: slope not significantly different from zero), whereas persistent noise (B) shows power-law scaling with decreasing amplitude at higher frequencies. The slope is $\beta = 0.52$ ($p < 0.0001$) which is in excellent agreement with the Hurst exponent of $H = 0.75$ determined for the corresponding series in Figure 10.21.

There are some special cases of β. Uncorrelated noise exhibits no frequency dependency of the spectral magnitude, and the slope β is not significantly different from zero. From Equation (10.16) follows $H = 0.5$, that is, the Hurst exponent associated with uncorrelated (white) noise. If the magnitude decays inversely proportional to the frequency ($\beta = -1$), the noise is referred to as $1/f$ *noise* or *pink noise*. An even steeper decay, $1/f^2$ ($\beta = -2$), is obtained from data series that originate from Brownian motion or random walks and is termed *brown* or *red noise*.

Similar to the estimators of the fractal dimension, the regression to determine β needs to be restricted to frequencies in which self-similar behavior can be expected. The presence of measurement noise can lead to apparent multifractal behavior where the magnitude at high frequencies decays with a different slope than the magnitude at low frequencies. A lowpass filter (such as Gaussian blurring) applied to white noise may lead to a spectrum that resembles pink noise, but it would not originate from a process with inherent scaling behavior.

The considerations described above can easily be extended to images. However, images may exhibit different spectral behavior, depending on the angle at which the spectrum is examined. Circles around the origin of a Fourier transform image connect locations of equal frequency, because the frequency ω increases with the Euclidean distance from the origin (i.e., $\omega = \sqrt{u^2 + v^2}$). As seen in Figure 10.23, spectra sampled along the u-axis and along a diagonal line starting at the origin look different and exhibit a different scaling exponent β. Multifractal behavior of Fourier transform images is therefore very common. In an extreme example, features with straight edges parallel to the y-axis and rugged edges along the x-axis would exhibit scaling behavior along the u-axis but not along the v-axis. Because of this complex behavior, the most common use of fractal analysis of Fourier transform images is to identify anisotropic fractal behavior. For this purpose, the scaling exponent β is

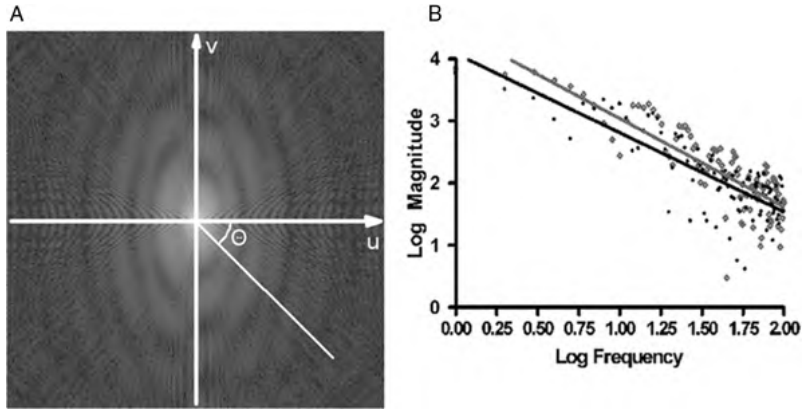


FIGURE 10.23 Frequency spectra observed at different angles in the Fourier transform of an image. Along the *u*-axis (gray data in the log-log plot), the frequency drops off faster than along a diagonal line of angle θ (black data in the log-log plot).

computed along one-dimensional spectra subtending various angles θ with the *u*-axis. The resulting function β(θ) can be displayed in the form of a rose plot. The information provided by a rose plot can be shown using two sample textures that were introduced in Chapter 8 (see the carpet and corduroy textures in Figures 8.10 and 10.28). Their fractal rose plots are shown in Figure 10.24. The average slope β is 1.29 for the carpet texture and 1.14 for the corduroy texture. However, the anisotropy, defined as the maximum value of β divided by the minimum value of β, is markedly

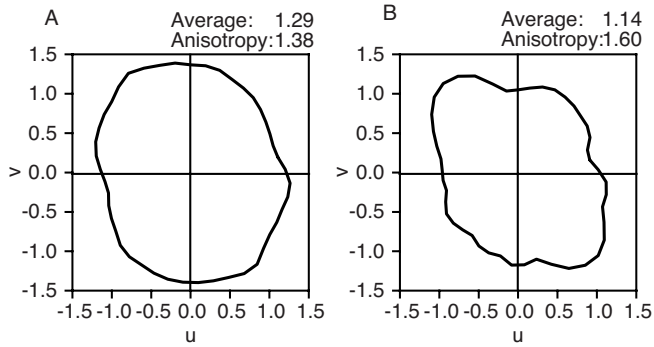


FIGURE 10.24 The slope of the double-logarithmic frequency decay of two textures from Chapter 8 (carpet, A, and corduroy, B, also shown in Figure 10.28), displayed as a function of the angle in form of a rose plot. The slopes (and therefore the Hurst exponents) are somewhat similar, but the corduroy texture (right) shows a higher anisotropy than the carpet texture, because the original texture has a stronger directional preference. The anisotropy angle of 120° also matches the directional preference of the texture in Figure 10.28.

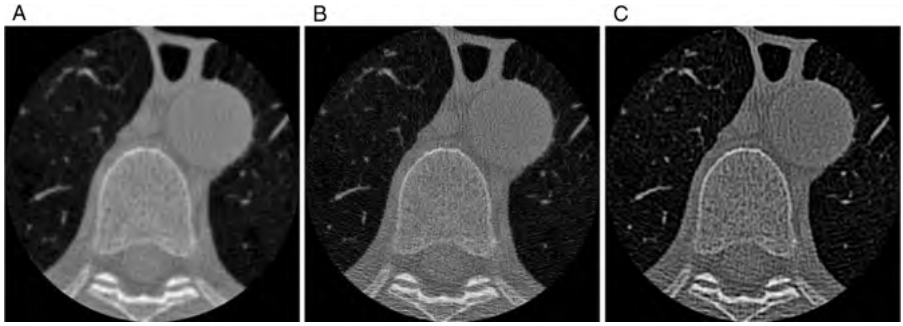


FIGURE 10.25 CT reconstructions of the thoracic region using different reconstruction kernels: (A) standard kernel, (B) bone kernel, (C) lung kernel. The reconstruction kernels emphasize different levels of detail, but they also create a pseudotexture with apparent fractal properties.

higher for the corduroy texture, indicating a higher directional preference. In addition, the angle of the anisotropy (the orientation of the idealized oval) of approximately 120° matches the orientation of the texture. The rose plot is therefore a convenient method to use for the quantitative determination of the anisotropic behavior of textures with self-similar properties. In addition to the average slope β, anisotropy magnitude, and angle, the intercept of the fit may be used to measure anisotropic behavior of the image intensity itself.

Figure 10.25 shows cross-sectional CT images of the thoracic region, that make possible easy identification of lungs, vertebra, and aorta. Corresponding rose plots are shown in Figure 10.26. The three panels show reconstructions performed with different reconstruction kernels: the standard kernel, the bone kernel, and the lung kernel. The purpose of these kernels (i.e., filters applied during reconstruction) is to emphasize specific details in the region of interest.

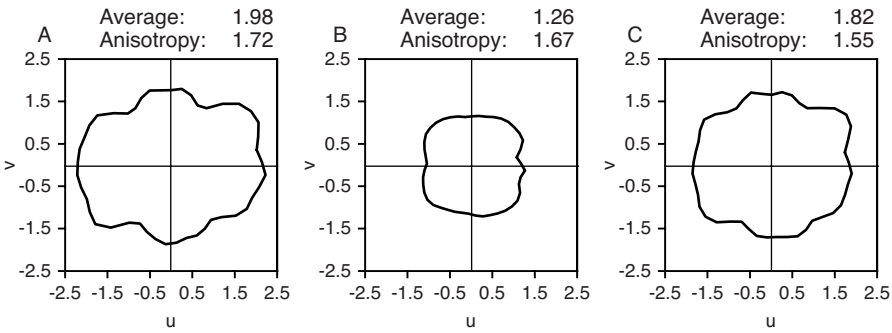


FIGURE 10.26 Rose plots of the frequency decay behavior of the CT reconstructions in Figure 10.25. The rose plots indicate strongly differing apparent self-similar behavior among the three CT images.

Although the reconstruction was performed using the same raw data, apparent fractal properties are markedly different. The texture that causes the apparent self-similar properties is a CT artifact. This can best be observed in the region of the aorta, where homogeneous blood flow should correspond to a homogeneous gray area in the reconstruction. In actual reconstructions, however, the aorta cross section is inhomogeneous, and the inhomogeneity depends on the reconstruction kernel. A linear regression into log-transformed data, in this case the logarithmic spectral amplitude over the logarithmic frequency, will often lead to a reasonable approximation of a straight line, but a thorough discussion of the origins of possible self-similar properties is needed in all cases.

10.5. LOCAL HÖLDER EXPONENT

The Hurst exponent H describing scaling behavior in random processes was introduced in Section 10.4. H can also be determined as a local property when a single window of variable length τ remains centered on the observational point in time t_0. In this case, H is referred to as the *local Hölder exponent* as opposed to the Hurst exponent, which applies to a global property. The rescaled range analysis yields one value of the Hölder exponent for each observational point in time and therefore provides a time series $H(t)$.

In images, time corresponds to the Euclidean distance between pixels. It is therefore possible to determine local scaling properties in binary and gray-scale images by analyzing the image values in expanding circles centered on the point for which the scaling property is to be determined. For this purpose, the gray-scale mass dimension is calculated for each pixel of the original image for a small region around the pixel. The scaling range (the range of radii for which the enclosed sum of image values is computed) varies with the total image size and the size of the features of interest, but typical radii range from 2 to 15 pixels. The algorithm to determine the local mass dimension is very similar to Algorithm 10.3, with the difference that the main loop of Algorithm 10.3 is embedded in another double loop over all of the image's pixels, and instead of using the central pixel (`xc`, `yc`), each individual pixel, addressed by the outer double loop, is used as the center for the mass dimension computation.

The effect of the local Hölder operator is demonstrated in Figure 10.27. The test image (Figure 10.27A) consists of four square patches of fractional Brownian noise (from left to right and top to bottom with $H = 0.2$, $H = 0.4$, $H = 0.6$, $H = 0.8$) over a background of white noise, with all areas having the same mean value and standard deviation. The local Hölder operator (Figure 10.27B) reveals the local scaling behavior. The areas with a high value of H can be identified clearly. An image (generally, any object) with regions that exhibit different scaling behavior is called *multifractal* and can be examined further by analyzing the histogram (the probability distribution) of the Hölder exponents.

Use of the Hölder operator with binary images provides information on the scaling behavior of the shape outlines. When applied to gray-scale images, the Hölder operator provides information on the scaling behavior of the texture. Whereas the

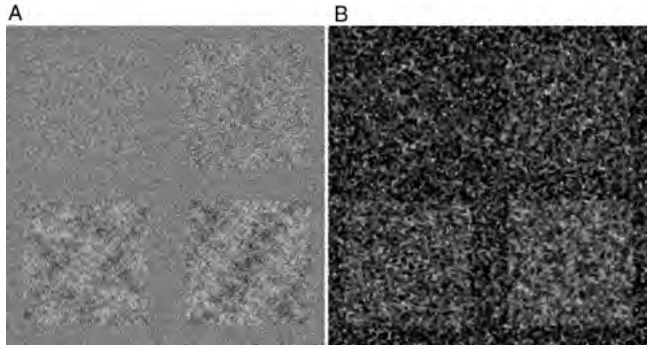


FIGURE 10.27 Demonstration of the effect of the local Hölder operator. Image A shows four patches of fractional Brownian noise over a background of Gaussian (white) noise. Image B shows the result after the application of the Hölder operator. Brighter areas indicate a higher exponent H.

analysis of time series is established, application of the Hölder operator requires special considerations that result from the nature of two-dimensional images. Strictly, the Hurst and Hölder exponents are defined only for random processes with long-term persistence. Therefore, the Hölder operator is best suited for analysis of the noise content of images. Furthermore, the numerical range of the exponents obtained from the Hölder operator can exceed the theoretical range of $0 < H < 1$. It is easy to construct cases where the central pixel lies in a dark image region and is surrounded by successively brighter areas as the Euclidean distance from the central point increases. In Figure 10.27B, values of up to 2.6 exist. This would translate into negative and thus invalid fractal dimensions. Finally, the two-dimensional nature of the expanding circles influences the measured mass as it averages more and more pixels with expanding radius. The larger circles contain enough pixels to eliminate many details by averaging. In the extreme case, the averaging behavior causes the scaling properties to degenerate to those of a flat surface. An unsuitable situation that fits this description can be recognized because the log-transformed data points of the mass $m(r)$ and its corresponding radius r no longer lie on a straight line. A linear fit would yield a poor correlation coefficient. It is therefore advisable to compute the correlation coefficient for each fit and reject any pixel with a poor correlation coefficient. Rejection criteria should be made strict, such as acceptance only of pixels with $r^2 > 0.8$. Furthermore, when analyzing the noise content of biomedical images, only few steps with small radii should be made to reduce the influence of large-area averaging.

To provide one example of texture analysis with the Hölder operator, Figure 10.28 shows two examples from the UIUC texture database[1] (see also Chapter 8) with their associated histograms of the Hölder exponent distribution. Preprocessing of the images is important. The influence of noise needs to be minimized by using noise reduction and filtering steps, and contrast should be maximized and standardized: for example, by local histogram equalization.

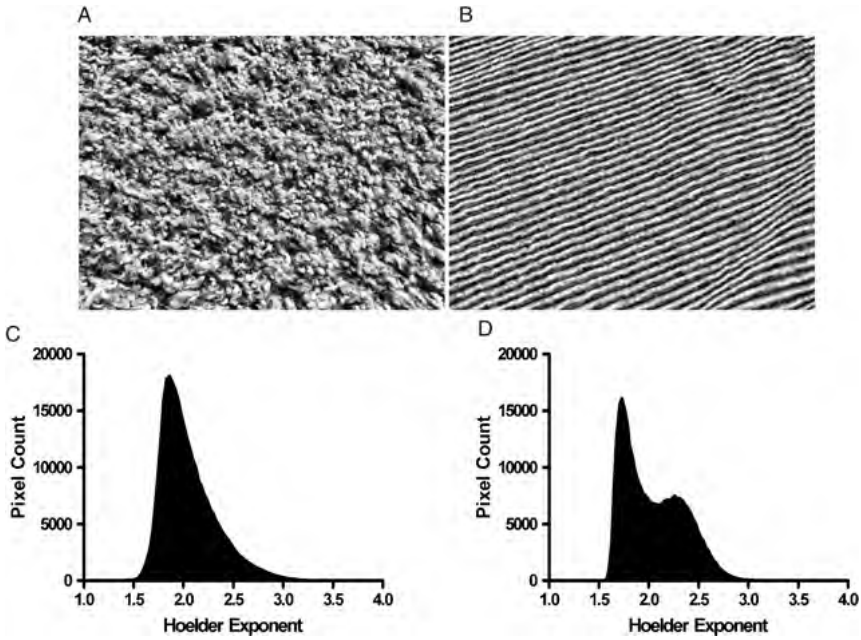


FIGURE 10.28 Comparison of the distribution of local Hölder exponents in images of two different textures: carpet (A) and corduroy (B) (see Chapter 8). The corresponding histograms of the local Hölder exponent are shown in parts C and D, respectively. The local Hölder operator may serve as an image-enhancement operator, and histogram analysis (described in Section 8.1) can be applied to obtain quantitative descriptors of the texture.

Different notions for the estimation of local Hölder exponents are possible. The rescaled range analysis can be applied directly to images. In this case, the minimum and maximum values (and therefore the range) are determined as a function of the circle radius. Equation (10.14) applies directly, whereby the radius r is used instead of the time window τ. A particularly efficient algorithm was proposed by Russ[61] in which an octagonal neighborhood with a total of 37 pixels is defined. For each of these pixels, the Euclidean distance to the central pixel is known (Figure 10.29). Based on this neighborhood definition, each pixel that surrounds the central pixel in a 7×7 box is labeled with a Euclidean distance to the central pixel, and a table is built that contains the distance and their associated ranges. The table contains seven rows ($1, \sqrt{2}, 2, \ldots, \sqrt{10}$) and two columns. The first column holds the distance. The second column is filled with the neighborhood range by considering larger and larger distances from row to row. Consequently, the first row contains the range of the four pixels labeled "1" in Figure 10.29. The second row contains the range of the previous four pixels and the four pixels labeled "$\sqrt{2}$," and so on. Once the table is filled, nonlinear regression yields the exponent H in Equation (10.14). The neighborhood range can be extended arbitrarily by extending the size of the octagon. However, local rescaled range analysis, similar to the local mass dimension, does not

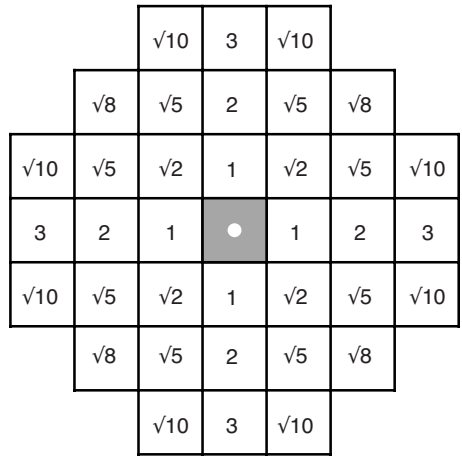


FIGURE 10.29 Octagonal neighborhood and the associated Euclidean distances to the central pixel (shaded).

necessarily yield valid values for the Hölder exponent. Consider the neighborhood given in Table 10.2 as an example. If the bold pixel is chosen as the central pixel, the range increases almost linearly with the Euclidean distance. However, if the central pixel to the right is chosen, the last three steps do not show an increase of the range. Although the regression would still output a value for H and a reasonable correlation coefficient, the scaling behavior has changed for the last three entries of the table (Figure 10.30), and it is debatable whether the resulting Hölder exponent should be accepted as valid.

10.6. BIOMEDICAL EXAMPLES

Power-law scaling behavior is widespread in nature on almost all scales.[6] It is therefore not surprising that power-law scaling behavior can be found in many biomedical

TABLE 10.2 Sample Octagonal Neighborhood with Discrete Image Intensity Values

		207	203	185	*222*		
	218	237	228	205	191	*227*	
200	225	235	223	205	201	205	*209*
188	198	207	**198**	191	201	227	*211*
190	134	190	178	176	190	222	*205*
	97	120	137	125	147	*163*	
		80	121	125	*171*		

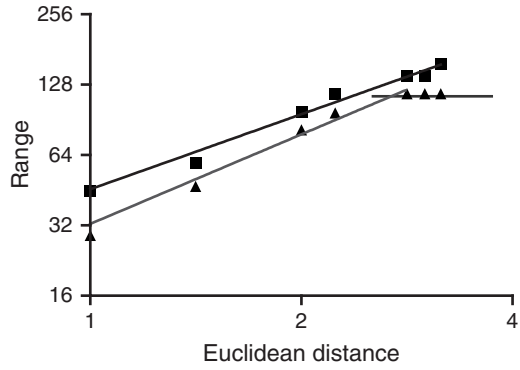


FIGURE 10.30 Nonlinear regression of the range data determined from Table 10.2. A good fit is found with the bold pixel (198) as the central pixel, whereas the neighborhood to the right (central pixel value 191) shows no scaling for the largest three distances (gray regression lines).

areas, including self-similarity properties of tumors, blood vessels, and neuronal trees.[16] The detection of scaling properties and self-similarity in biomedical images is considered mainstream in image analysis. In two areas, fractal methods for the description of structural properties are particularly popular and widespread: the analysis of trabecular bone in osteoporosis, and the analysis of lung structure in relation to pulmonary function.

Self-Similarity and Trabecular Bone The complex three-dimensional structure of trabecular bone attracted the interest of numerous groups of investigators (see, e.g., Link et al.[42] for a review) who examined changes of scaling properties associated with osteoporosis, a disease that is associated with loss of gross bone density but also with reduced trabecular connectivity. In microscopic images of biopsies, this process can be observed, but individual trabeculae show a smooth shape and therefore nonfractal behavior.[12] This observation is consistent with considerations that any image feature exhibits self-similar properties only within certain scales. At larger, nonmicroscopic, scales, however, self-similar properties were reported irrespective of the imaging modality, and the observation of a lower fractal dimension in bone affected by osteoporosis is highly consistent between research groups and studies. Lumbar vertebrae or the calcaneus were excised and radiographed.[7] It is possible to observe trabecular degeneration subjectively by visual examination of the radiographed texture.[3] Such an example is shown in Figure 10.31. The differences in trabecular architecture are immediately obvious under visual inspection, but unsupervised texture analysis is not straightforward. For example, the blanket dimension of the segmented trabecular region shows only minimal differences (1.96 *vs.* 1.92). However, the application of a weak highpass filter to remove inhomogeneous background and to emphasize the trabeculae, followed by a suitable threshold operation (such as Otsu's method), allows determination of the box-counting and Minkowsky

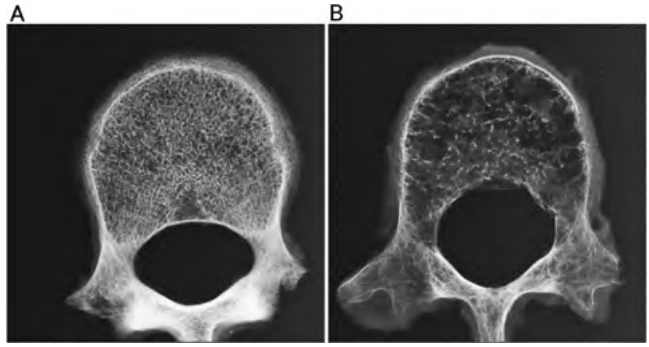


FIGURE 10.31 Radiographic images of slices of postmortem excised lumbar vertebrae from patients without (A) and with severe (B) osteoporosis. (Adapted from ref. 25.)

dimensions. Here the dimension values are markedly different. The box-counting method yields dimension values of 1.87 versus 1.52, and the Minkowsky dimension is 1.73 versus 1.26.

In vivo imaging methods such as projection x-ray imaging, CT, and MRI usually provide a much lower spatial resolution than that of direct x-ray imaging, as shown in Figure 10.31. Furthermore, the image formation process corresponds to a convolution with the point-spread function of the imaging system (usually, a function with lowpass characteristics), and the image contains added noise and possibly pseudotexture (see Figure 10.25). In the case of x-ray projection imaging, the imaged structure is a two-dimensional projection of a three-dimensional network. This limits the application of fractal methods to a comparison of the trabecular texture between patients who are imaged with exactly the same modality, and makes it difficult to compare fractal dimension values between different study groups. One difficulty arises from the question of a suitable binarization threshold.[26] Although it can be argued that image features above certain Hounsfield units in CT are likely to be related to bone, the imaging function and partial-volume effects will lead to voxels that are reconstructed from bone but have a lower Hounsfield number than is typically seen for bone. In projection x-ray imaging, the threshold depends strongly on the image exposure. Suitable fractal analysis of the texture would therefore have to include image values that typically are not associated with bone. The blanket dimension, for example, can be applied to images without the need for thresholding.

It is possible to perform a simple verification that the dimension values are not the result of differences in image intensity or exposure: If the image intensity values are subjected to linear pixel remapping with the function $I'(x, y) = a \cdot I(x, y) + b$, where a is a linear scaling parameter and b is an intensity offset, and the new image with intensities I' is subjected to the same fractal dimension analysis, the resulting dimension value should be independent of the choice of a and b as long as $a > 0$.[26] Tests such as this linear pixel remapping test are important in verifying that the slope

values really reflect self-similar behavior rather than scaling or intensity changes hidden in the image processing chain.

These cautionary remarks notwithstanding, the analysis of fractal properties in trabecular bone has been remarkably successful. Lynch et al. developed a dilation method related to the Minkowsky dimension to determine the fractal signature of trabeulcar bone in projection x-ray images of the knee[44,45] and related the dimension to osteoarthritis. The results of this study were later corroborated by Podsiadlo et al.[57] Weinstein and Majumdar used contour tracing to segment trabeculae in projection x-ray images of bone biopsies[69] and cross-sectional CT images[49] and consistently found a lower fractal dimension in cases affected by osteoporosis or with lower vertebral fracture load. Caldwell et al.[9] showed that the mass dimension of the trabecular structure in projection images was related to fracture load. X-ray imaging, performed routinely in dental examinations, may also help in detecting periodontitis as shown by Updike and Nowzari.[66] This study showed reduced complexity of the trabecular pattern, as evidenced by a lower box-counting dimension in patients with periodontitis. Despite the lower resolution and lowpass image transfer function of computed tomography, CT images can also be used to quantify trabecular structure.[15,17,18] Although magnetic resonance imaging is not typically used to image bone, MRI sequences can be designed to obtain high-resolution images of the trabecular structure,[10,32,67] for example, by making use of the low proton density in bone (low water content) or of susceptibility variations that alter T_2^*. Several studies indicate that an estimate of the fractal dimension is a meaningful parameter to estimate changes in bone microarchitecture.[4,27,42,47]

However, care must be taken in the design of fractal-based image analysis methods. Some examples include consideration of the limits of resolution,[12,56] the presence of image noise, variability of interactive components in the image analysis method,[68] or the presence of other tissue components.[11] With a carefully designed study, however, estimates of the fractal dimension are useful metrics in assessing the complexity of trabecular bone microarchitecture, as evidenced in a study by Berry et al.[4] In this study, trabecular bone was radiographed during exposure to nitric acid, a procedure designed to erode the trabeculae and therefore simulate the processes associated with osteoporosis. It was found that nitric acid exposure led to a significant reduction in the box-counting dimension.

Fractal Dimension in Pulmonary Structures Fractal properties of anatomical structures in the lung were discovered early[53] and continue to be the subject of research investigations. Fractal properties of lung tissues were found on different scales. Anderson et al.[2] found that the radial distribution of the cross-sectional area of bronchial capillaries was fractal. Kamiya and Takahashi[31] examined the fractal dimension of the three-dimensional vascular tree using microscopic images and used these fractal properties to quantify various hydrodynamic parameters, such as branch flow rate, mean flow velocity, wall shear rate and stress, internal pressure, and circumferential tension. In lung casts, a relationship between the degree of asthma and the fractal dimension was found.[5] Experienced radiologists can diagnose lung

disorders in projection x-ray images of the chest. Although automated image analysis of chest x-ray images is difficult, the fractal dimension was suggested as one metric to diagnose lung interstitial abnormalities.[36,37] Considering the three-dimensional character of lung structures, computed tomography and magnetic resonance imaging are preferable imaging modalities. Computed tomography provides three-dimensional images that allow the quantitative determination of lung degeneration.[46] Lung emphysema is a degenerative disease associated with biomechanical changes and degeneration of the lung tissues, including loss of elasticity of the lung tissue, degeneration of alveolar structures, and destruction of capillaries feeding the alveoli. Perfusion studies were performed using single-photon emission computed tomography (SPECT), and a dimension metric computed from the area inside gray-scale isocontour lines was found to be larger in patients with further developed emphysema.[55] However, these and similar findings were criticized by Chung and Huang[13] as being more representative of the total perfused area rather than changes in the scaling behavior of self-similar structures. Nonetheless, it is likely that CT images of the thoracic region, together with fractal analysis operators, can be used to determine the progress of emphysema. Figure 10.32 shows two examples of CT images of the lung with the vascular tree clearly visible. The healthy case shows a dense, branching vascular tree, whereas the emphysemic case exhibits a less complex vascular structure dominated by thinner and more linear blood vessels.

Although this type of structure can be well quantified with fractal methods, more studies would be needed to determine the optimum image processing path. As shown in Figure 10.32, the vascular tree can readily be isolated and its fractal dimension measured. In this special case, the box-counting dimension would yield a value of 1.59 for the healthy case and 1.56 for the diseased case. The Minkowsky dimension yields similar values of 1.60 and 1.56, respectively. The mass dimension shows larger differences in these two examples: 2.36 versus 1.88. Whether these values are representative of the state of the disease and significant in a statistical sense would have to be determined in a larger study. Furthermore, a potential fallacy arises from

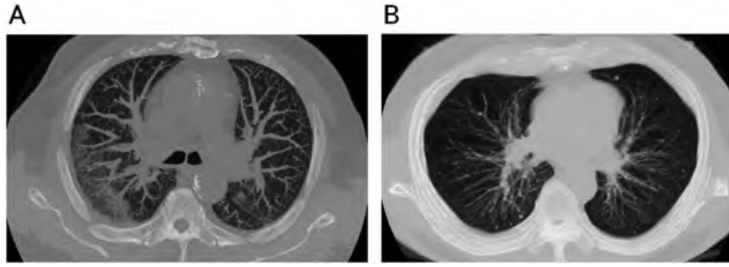


FIGURE 10.32 Computed tomography images of the thoracic region (maximum intensity projections) showing prominently the vascular structure inside the lungs. The blood vessels are fairly dense and intricately branched in the healthy case (A), whereas the emphysemic case (B) shows reduced complexity in the vascular structure.

the sensitivity of fractal methods toward the area fraction occupied by the feature. This potential pitfall can be avoided by using the structure's skeleton.[43]

Fractal Properties in the Brain Fractal properties were also found in structures inside the brain, from the neuronal scale[58,70] to the macroscopic scale.[8,14,23,65] Aging is associated with a loss of structural complexity, which has been quantified using the fractal dimension by Kedzia et al.[35] A similar reduction in the fractal dimension (loss of complexity) was associated with multiple sclerosis,[20] determined by measuring the fractal dimension of the segmented white matter. In healthy subjects, the brain hemispheres are asymmetric. This asymmetry was quantified by Lee et al.,[40] who segmented the brain surface in MR images and determined the box-counting dimension of the skeleton of the brain surface. The fractal dimension was significantly higher in the right hemisphere than in the left hemisphere, but no differences were found with age and gender. This study indicates that the fractal dimension may be used to quantify structural complexity in the brain surface. Im et al.[29] suggest the possibility that an elevated complexity, determined by measuring the fractal dimension of the brain surface, is correlated with education and the intelligence quotient. Schizophrenia is also hypothesized to be associated with structural changes in the brain . Sandu et al.[63] found that the boundary between white and gray matter shows a higher fractal dimension (as determined by the box-counting and Minkowsky methods) in patients with schizophrenia than in healthy controls. Furthermore, two example studies demonstrate the fractal nature of cerebral blood flow. In the fetal brain, the group of Rybaczuk and Kedzia introduced a fractal model for the fetal blood vessel architecture[62] and used this method to demonstrate a rapid increase of the dimension from 1.26 to over 1.5 during the second trimester,[34] thereby indicating that in this period a major functional development of the brain takes place. A study by Kalmanti and Maris[30] shows that the complexity increases further during adolescence, with predominant remodeling taking place in the left hemisphere, whereas remodeling in the right hemisphere takes place into adulthood. By using SPECT, Kuikka and Hartikainen[39] determined the heterogeneity of blood flow in the brain using an underlying fractal Brownian noise model. In this study, a fractal dimension of 1.16 was found in healthy subjects, compared to 1.04 in patients with frontal lobe dementia. In this model, a value of 1.0 would indicate completely homogeneous blood flow, and the study shows that dementia is associated with a loss of complexity.

REFERENCES

1. Anonymous. Texture database. http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/. Accessed Nov. 8, 2008.
2. Anderson JC, Babb AL, Hlastala MP. A fractal analysis of the radial distribution of bronchial capillaries around large airways. J Appl Physiol 2005; 98(3):850–855.
3. Andresen R, Radmer S, Banzer D. [Development of a CT data-based score for prediction of fracture risk in osteoporosis]. Aktuel Radiol 1997; 7(5):264–269.

4. Berry JL, Towers JD, Webber RL, Pope TL, Davidai G, Zimmerman M. Change in trabecular architecture as measured by fractal dimension. J Biomech 1996; 29(6):819–822.
5. Boser SR, Park H, Perry SF, Menache MG, Green FH. Fractal geometry of airway remodeling in human asthma. Am J Respir Crit Care Med 2005; 172(7):817–823.
6. Brown JH, Gupta VK, Li BL, Milne BT, Restrepo C, West GB. The fractal nature of nature: power laws, ecological complexity and biodiversity. Philos Trans R Soc Lond B 2002; 357(1421):619–626.
7. Buckland-Wright JC, Lynch JA, Rymer J, Fogelman I. Fractal signature analysis of macroradiographs measures trabecular organization in lumbar vertebrae of postmenopausal women. Calcif Tissue Int 1994; 54(2):106–112.
8. Bullmore E, Brammer M, Harvey I, Persaud R, Murray R, Ron M. Fractal analysis of the boundary between white matter and cerebral cortex in magnetic resonance images: a controlled study of schizophrenic and manic-depressive patients. Psychol Med 1994; 24(3):771–781.
9. Caldwell CB, Rosson J, Surowiak J, Hearn T. Use of the fractal dimension to characterize the structure of cancellous bone in radiographs of the proximal femur. In: Nonnenmacher TF, Loser GA, Weibel ER, editors. Fractals in Biology and Medicine. Basel, Switzerland: Birkhäuser, 1994; 300–306.
10. Chang G, Pakin SK, Schweitzer ME, Saha PK, Regatte RR. Adaptations in trabecular bone microarchitecture in Olympic athletes determined by 7T MRI. J Magn Reson Imaging 2008; 27(5):1089–1095.
11. Chappard D, Pascaretti-Grizon F, Gallois Y, Mercier P, Basle MF, Audran M. Medullar fat influences texture analysis of trabecular microarchitecture on x-ray radiographs. Eur J Radiol 2006; 58(3):404–410.
12. Chung HW, Chu CC, Underweiser M, Wehrli FW. On the fractal nature of trabecular structure. Med Phys 1994; 21(10):1535–1540.
13. Chung HW, Huang YH. Fractal analysis of nuclear medicine images for the diagnosis of pulmonary emphysema: interpretations, implications, and limitations. AJR Am J Roentgenol 2000; 174(4):1055–1059.
14. Cook MJ, Free SL, Manford MR, Fish DR, Shorvon SD, Stevens JM. Fractal description of cerebral cortical patterns in frontal lobe epilepsy. Eur Neurol 1995; 35(6):327–335.
15. Cortet B, Dubois P, Boutry N, Palos G, Cotten A, Marchandise X. Computed tomography image analysis of the calcaneus in male osteoporosis. Osteoporos Int 2002; 13(1):33–41.
16. Cross SS. Fractals in pathology. J Pathol 1997; 182(1):1–8.
17. Dougherty G. A comparison of the texture of computed tomography and projection radiography images of vertebral trabecular bone using fractal signature and lacunarity. Med Eng Phys 2001; 23(5):313–321.
18. Dougherty G, Henebry GM. Fractal signature and lacunarity in the measurement of the texture of trabecular bone in clinical CT images. Med Eng Phys 2001; 23(6):369–380.
19. Ebert DS, Musgrave FK, Peachey D, Perlin K, Worley S. Texturing and Modeling. San Francisco: Morgan Kaufmann, 2003.
20. Esteban FJ, Sepulcre J, de Mendizabal NV, Goni J, Navas J, de Miras JR, Bejarano B, Masdeu JC, Villoslada P. Fractal dimension and white matter changes in multiple sclerosis. Neuroimage 2007; 36(3):543–549.
21. Family F. Kinetics of Aggregation and Gelation. Amsterdam: North-Holland, 1985.

22. Fernandez E, Bolea JA, Ortega G, Louis E. Are neurons multifractals? J Neurosci Methods 1999; 89(2):151–157.
23. Free SL, Sisodiya SM, Cook MJ, Fish DR, Shorvon SD. Three-dimensional fractal analysis of the white matter surface from magnetic resonance images of the human brain. Cereb Cortex 1996; 6(6):830–836.
24. Geraets WG, van der Stelt PF. Fractal properties of bone. Dentomaxillofac Radiol 2000; 29(3):144–153.
25. Haidekker MA, Andresen R, Evertsz CJ, Banzer D, Peitgen HO. Assessing the degree of osteoporosis in the axial skeleton using the dependence of the fractal dimension on the grey level threshold. Br J Radiol 1997; 70(834):586–593.
26. Haidekker MA, Andresen R, Evertsz CJ, Banzer D, Peitgen HO. Issues of threshold selection when determining the fractal dimension in HRCT slices of lumbar vertebrae. Br J Radiol 2000; 73(865):69–72.
27. Hudelmaier M, Kollstedt A, Lochmuller EM, Kuhn V, Eckstein F, Link TM. Gender differences in trabecular bone architecture of the distal radius assessed with magnetic resonance imaging and implications for mechanical competence. Osteoporos Int 2005; 16(9):1124–1133.
28. Hurst HE. Long term storage capacity of reservoirs. Trans Am Soc Civ Eng 1951; 116:770–808.
29. Im K, Lee JM, Yoon U, Shin YW, Hong SB, Kim IY, Kwon JS, Kim SI. Fractal dimension in human cortical surface: multiple regression analysis with cortical thickness, sulcal depth, and folding area. Hum Brain Mapp 2006; 27(12):994–1003.
30. Kalmanti E, Maris TG. Fractal dimension as an index of brain cortical changes throughout life. In Vivo 2007; 21(4):641–646.
31. Kamiya A, Takahashi T. Quantitative assessments of morphological and functional properties of biological trees based on their fractal nature. J Appl Physiol 2007; 102(6):2315–2323.
32. Kang C, Paley M, Ordidge R, Speller R. R′2 measured in trabecular bone in vitro: relationship to trabecular separation. Magn Reson Imaging 1999; 17(7):989–995.
33. Kaye BH. Image analysis techniques for characterizing fractal structures. In: Avnir D, editor. The Fractal Approach to Heterogeneous Chemistry: Surfaces, Colloids, Polymers. New York: Wiley, 1990; 55–66.
34. Kedzia A, Rybaczuk M, Andrzejak R. Fractal dimensions of human brain cortex vessels during the fetal period. Med Sci Monit 2002; 8(3):MT46–MT51.
35. Kedzia A, Rybaczuk M, Dymecki J. Fractal estimation of the senile brain atrophy. Folia Neuropathol 1997; 35(4):237–240.
36. Kido S, Kuroda C, Tamura S. Quantification of interstitial lung abnormalities with chest radiography: comparison of radiographic index and fractal dimension. Acad Radiol 1998; 5(5):336–343.
37. Kido S, Tamura S. Computerized classification of interstitial lung abnormalities on chest radiographs with normalized radiographic index and normalized fractal dimension. Eur J Radiol 2001; 37(3):184–189.
38. Kochetkov KV, Kazachenko VN, Aslanidi OV, Chemeris NK, Gapeyev AB. Non-Markovian Gating of Ca^{2+}-activated K^+ channels in cultured kidney cells Vero: rescaled range analysis. J Biol Phys 1999; 25(2):211–222.

39. Kuikka JT, Hartikainen P. Heterogeneity of cerebral blood flow: a fractal approach. Nuklearmedizin 2000; 39(2):37–42.
40. Lee JM, Yoon U, Kim JJ, Kim IY, Lee DS, Kwon JS, Kim SI. Analysis of the hemispheric asymmetry using fractal dimension of a skeletonized cerebral surface. IEEE Trans Biomed Eng 2004; 51(8):1494–1498.
41. Liebovitch LS, Toth TI. Fractal activity in cell membrane ion channels. Ann N Y Acad Sci 1990; 591:375–391.
42. Link TM, Majumdar S, Grampp S, Guglielmi G, van Kuijk C, Imhof H, Glueer C, Adams JE. Imaging of trabecular bone structure in osteoporosis. Eur Radiol 1999; 9(9):1781–1788.
43. Liu JZ, Zhang LD, Yue GH. Fractal dimension in human cerebellum measured by magnetic resonance imaging. Biophys J 2003; 85(6):4041–4046.
44. Lynch JA, Hawkes DJ, Buckland-Wright JC. A robust and accurate method for calculating the fractal signature of texture in macroradiographs of osteoarthritic knees. Med Inf (Lond) 1991; 16(2):241–251.
45. Lynch JA, Hawkes DJ, Buckland-Wright JC. Analysis of texture in macroradiographs of osteoarthritic knees using the fractal signature. Phys Med Biol 1991; 36(6):709–722.
46. Madani A, Keyzer C, Gevenois PA. Quantitative computed tomography assessment of lung structure and function in pulmonary emphysema. Eur Respir J 2001; 18(4): 720–730.
47. Majumdar S, Genant HK, Grampp S, Newitt DC, Truong VH, Lin JC, Mathur A. Correlation of trabecular bone structure with age, bone mineral density, and osteoporotic status: in vivo studies in the distal radius using high resolution magnetic resonance imaging. J Bone Miner Res 1997; 12(1):111–118.
48. Majumdar S, Lin J, Link T, Millard J, Augat P, Ouyang X, Newitt D, Gould R, Kothari M, Genant H. Fractal analysis of radiographs: assessment of trabecular bone structure and prediction of elastic modulus and strength. Med Phys 1999; 26(7):1330–1340.
49. Majumdar S, Weinstein RS, Prasad RR. Application of fractal geometry techniques to the study of trabecular bone. Med Phys 1993; 20(6):1611–1619.
50. Malcai O, Lidar DA, Biham O. Scaling range and cutoffs in empirical fractals. Phys Rev E 1997; 56(3):2817–2828.
51. Mandelbrot B. How long is the coast of Britain? Statistical self-similarity and fractional dimension. Science 1967; 156(3775):636–638.
52. Mandelbrot B, Hudson RL. The (Mis)behaviour of Markets. A Fractal Risk of Risk, Ruin and Reward. New York: Basic Books, 2004.
53. McNamee JE. Fractal perspectives in pulmonary physiology. J Appl Physiol 1991; 71(1):1–8.
54. Meakin P. Simulation of aggregation processes. In: Avnir D, editor. The Fractal Approach to Heterogeneous Chemistry: Surfaces, Colloids, Polymers. New York: Wiley, 1990; 131–160.
55. Nagao M, Murase K, Yasuhara Y, Ikezoe J. Quantitative analysis of pulmonary emphysema: three-dimensional fractal analysis of single-photon emission computed tomography images obtained with a carbon particle radioaerosol. Am J Roentgenol 1998; 171(6): 1657–1663.
56. Penn AI, Loew MH. Estimating fractal dimension with fractal interpolation function models. IEEE Trans Med Imaging 1997; 16(6):930–937.

57. Podsiadlo P, Dahl L, Englund M, Lohmander LS, Stachowiak GW. Differences in trabecular bone texture between knees with and without radiographic osteoarthritis detected by fractal methods. Osteoarthritis Cartilage 2008; 16(3):323–329.
58. Porter R, Ghosh S, Lange GD, Smith TG Jr. A fractal analysis of pyramidal neurons in mammalian motor cortex. Neurosci Lett 1991; 130(1):112–116.
59. Rangayyan RM, Nguyen TM. Fractal analysis of contours of breast masses in mammograms. J Digit Imaging 2007; 20(3):223–237.
60. Riegler H, Köhler R. How pre-melting on surrounding interfaces broadens solid–liquid phase transitions. Nat Phys 2007; 3:890–894.
61. Russ JC. Image enhancement. In: The Image Processing Handbook. Boca Raton, FL: CRC Press, 1999; 227–304.
62. Rybaczuk M, Kedzia A, Paradowski L. Fractal characteristics of brain vessel microangioarchitecture during the fetal period. Med Sci Monit 2002; 8(8):MT145–MT152.
63. Sandu AL, Rasmussen IA Jr, Lundervold A, Kreuder F, Neckelmann G, Hugdahl K, Specht K. Fractal dimension analysis of MR images reveals grey matter structure irregularities in schizophrenia. Comput Med Imaging Graph 2008; 32(2):150–158.
64. Shen L, Rangayyan RM, Desautels JL. Application of shape analysis to mammographic calcifications. IEEE Trans Med Imaging 1994; 13(2):263–274.
65. Thompson PM, Schwartz C, Lin RT, Khan AA, Toga AW. Three-dimensional statistical analysis of sulcal variability in the human brain. J Neurosci 1996; 16(13):4261–4274.
66. Updike SX, Nowzari H. Fractal analysis of dental radiographs to detect periodontitis-induced trabecular changes. J Periodont Res 2008.
67. Vasilic B, Song HK, Wehrli FW. Coherence-induced artifacts in large-flip-angle steady-state spin-echo imaging. Magn Reson Med 2004; 52(2):346–353.
68. Vokes TJ, Pham A, Wilkie J, Kocherginsky M, Ma SL, Chinander M, Karrison T, Bris O, Giger ML. Reproducibility and sources of variability in radiographic texture analysis of densitometric calcaneal images. J Clin Densitom 2008; 11(2):211–220.
69. Weinstein RS, Majumdar S. Fractal geometry and vertebral compression fractures. J Bone Miner Res 1994; 9(11):1797–1802.
70. Wingate RJ, Fitzgibbon T, Thompson ID. Lucifer yellow, retrograde tracers, and fractal analysis characterise adult ferret retinal ganglion cells. J Comp Neurol 1992; 323(4):449–474.
71. Witten TA, Sander LM. Diffusion-limited aggregation, a kinetic critical phenomenon. Phys Rev Lett 1981; 47:1400–1403.

11

IMAGE REGISTRATION

When the same object is imaged with different modalities, or when the same object is rescanned, it may be desirable to match the exact spatial position of the features between the images. In some cases the resolution between modalities is not matched, or one image may be spatially distorted. The steps needed to spatially superimpose the images are called *image registration*. One example is the registration of magnetic resonance imaging (MRI) and nuclear imaging modalities, such as positron emission tomography (PET) or single-photon emission computed tomography (SPECT). Both PET and SPECT have a very low spatial resolution. MRI provides anatomical details by merit of its high spatial resolution and tissue contrast. Therefore, SPECT or PET images are often superimposed over an MRI scan to allow better visualization of the active sites. For an accurate match of the corresponding sites in both scans, the PET or SPECT image needs to be rescaled to match the resolution of the MR image, and translated and rotated to match the spatial locations between the images. It is possible to combine several modalities in one gantry to form hybrid imaging systems.[7,32] However, multimodality devices only provide registered images for a single exam. For repeated exams and intrasubject matching, software registration is still needed.

Registration may take place in two or three dimensions. Unlike in other chapters, where the two-dimensional case was presented because the three-dimensional case is a straightforward generalization, image registration will here be illuminated in three dimensions with a highlight on the two-dimensional special case whenever appropriate. Because of the high complexity of registration algorithms, this chapter

Advanced Biomedical Image Analysis, By Mark A. Haidekker

does not present algorithms in pseudocode. Rather, algorithms are explained in step-by-step fashion.

The goal of registration is to match homologous areas between two or more images. For this purpose, two different models can be used: rigid-body models and elastic models. To register two images under the rigid-body model, spatial transformations are applied to rotate, translate, and scale one image. The head is a good example of where the rigid-body model is applicable, because the head bones prevent any major deformation. Additional transformations that would fall into this category are the shear and perspective transforms. Rigid-body transforms are usually sufficient to correct mismatches caused by patient repositioning and by differences in resolution between modalities.

Elastic models and nonlinear transformations are required when the linear transformations above are not sufficient to correct positioning mismatches. Three typical examples of nonlinear effects that require nonlinear transformations are imperfections and artifacts in the imaging system, distortion of soft tissues, and intersubject variability. The latter usually produces the most complex requirements for a nonlinear transformation to match homologous points.

Imaging systems may cause nonlinear mapping distortions. For example, field inhomogeneities in a magnetic resonance scanner may cause nonlinear distortions in all three dimensions. A single inhomogeneous gradient coil will cause a nonlinear distortion in one spatial direction. MR scanners are usually carefully adjusted with shim magnets to compensate for any field inhomogeneities. If measurable spatial distortions remain, a mapping function needs to be determined experimentally and applied to each image scanned. PET is another modality that can exhibit spatial distortions because of detector alignment issues, differences in detector dead time, and differences in detector quantum efficiency.

Soft tissues are subject to deformation. Gravity plays a role when a patient is repositioned. Even more predominant are motion artifacts. Examples are breathing motion, heartbeat, and blood flow. Motion artifacts are either quasistatic or dynamic. A patient can hold his or her breath more or less deeply, causing different levels of deformation of chest and abdomen. Nonlinear transformations are capable of correcting this artifact. Heartbeat and blood flow are in many cases faster than the imaging process. This leads to specific distortions, such as motion blur in CT images and ghosting in MR images. This type of artifact can no longer be corrected by linear or nonlinear transformations.

Intersubject variability plays the most important role in the analysis of brain structure. Many research groups attempt to find a correlation between brain structure and function. Frequently, the task arises to register the brain images of different people or the image from a person to some representative shape, such as an atlas model. This type of registration is highly nonlinear with a high variability and presents one of the most challenging tasks in image registration.

To automate the registration process, it is necessary to define a quality function that provides a quantitative metric as to how well the two bodies match. In the simplest case, the surface contour or fiducial markers are brought to congruence. In more complex cases, shape- or intensity-based quality functions need to be defined.

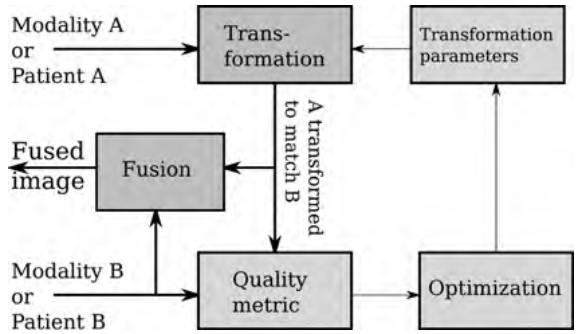


FIGURE 11.1 Schematic representation of the registration process. The image from modality or patient A undergoes a transformation to match the reference, that is, modality or patient B. After transformation, A and B are compared and a strategy to optimize the match between the images is applied. On the basis of this strategy, new transformation parameters are computed and applied. The process may be repeated iteratively. Once the registration process has converged, both images may be fused, that is, superimposed for visualization.

A general schematic of the registration process is shown in Figure 11.1. Usually, one image serves as the reference and the other image (often called the *floating image*) undergoes a transformation to match the reference image. The quality function is applied to determine how well the two images match, and a strategy to improve the match is developed. With the strategy, a new set of transformation parameters is generated, and the registration process is repeated iteratively.

11.1. LINEAR SPATIAL TRANSFORMATIONS

Let us consider a head scan performed with magnetic resonance as an example. If a patient is scanned repeatedly, the head may be positioned differently for each scan; that is, it may be rotated and translated. Any translation can be decomposed into orthogonal translations along the three principal axes, and any rotation can be similarly decomposed into rotations around the three principal axes. To describe translation and rotation, six parameters are therefore necessary, and the symbols Δx, Δy, and Δz will be used for translation, and ϕ_x, ϕ_y, and ϕ_z for the rotation around the x, y, and z axes, respectively. Translation and rotation are the fundamental operations in the rigid-body model. Linear spatial transforms can be represented elegantly by matrix operations.

The translation operation is a special case in the context of linear transformations in two respects. First, the translation is not a linear operation in the strict definition of a linear operator O applied to an operand p such that $q = O(p)$. If O is linear, the superposition rule and the scaling rule with a scalar a, respectively, must hold:

$$O(p + p') = O(p) + O(p') \tag{11.1}$$

$$O(ap) = aO(p) \tag{11.2}$$

Although the translation operation follows the superposition rule, it violates the scaling rule. Strictly, the translation operation is not a linear operation. However, most frequently the translation operation is listed among the linear operations in the context of registration, and in this chapter we also follow this convention. Second, the translation operation is additive, whereas matrix multiplications do not have a suitable additive element. The matrix representation of the translation operation therefore requires the introduction of a fourth dimension in which the addition can be modeled in multiplicative form. To allow chaining of operations, all transformation matrices, not only the matrix for translation, are 4×4 matrices for transformations in three-dimensional space. The translation of a point $P = [x \;\; y \;\; z]$ to a point $Q = [x' \;\; y' \;\; z']$ can be described by

$$x' = x + \Delta x; \qquad y' = y + \Delta y; \qquad z' = z + \Delta z \tag{11.3}$$

The same operation can be formulated in matrix form as $Q = P \cdot \mathbf{T}$ in

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix} \tag{11.4}$$

where $\mathbf{T}$ is the 4×4 translation matrix. It is necessary to add a fourth element to P and Q such that $P = [x \;\; y \;\; z \;\; 1]$ and $Q = [x' \;\; y' \;\; z' \;\; 1]$. The fourth element of P with value 1 allows the last row of $\mathbf{T}$ to become an additive component.

The rotation operation is most conveniently split into three separate operations: rotations around the x-axis, y-axis, and z-axis, respectively:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_x & \sin\phi_x & 0 & 0 \\ -\sin\phi_x & \cos\phi_x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.5}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_y & 0 & -\sin\phi_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi_y & 0 & \cos\phi_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.6}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi_z & \sin\phi_z & 0 \\ 0 & -\sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.7}$$

The three rotation matrices are designated $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$, respectively.

A scaling operation is also included in the rigid-body model. The scaling operation could, for example, account for different spatial resolution settings of the imaging

device. Nonistoropic scaling is fully described by three parameters: s_x, s_y, and s_z. The scaling operation is described by $Q = P \cdot \mathbf{S}$, with $\mathbf{S}$ being the scaling matrix according to:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.8}$$

In two dimensions, any point has three elements, $P = [x, y, 1]$, and the scaling, translation, and rotation matrices simplify to $\mathbf{S}$, $\mathbf{T}$, and $\mathbf{R}$ (note that only one rotation operation is possible, rotation around the axis normal to the image surface):

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}; \quad \mathbf{R} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11.9}$$

All matrix operations take place with respect to some arbitrarily defined origin. For convenience, either an image corner or the image center may be chosen. The order of the matrix operations is relevant, and the operations are generally not commutative. The operations $Q = P{\cdot}\mathbf{R}{\cdot}\mathbf{T}$ and $Q = P{\cdot}\mathbf{T}{\cdot}\mathbf{R}$ lead to different values of Q, as demonstrated by the two-dimensional example

$$\mathbf{R} \cdot \mathbf{T} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}; \qquad \mathbf{T} \cdot \mathbf{R} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ \Delta x' & \Delta y' & 1 \end{bmatrix} \tag{11.10}$$

where $\Delta x' = \Delta x \cos\phi - \Delta y \sin\phi$ and $\Delta y' = \Delta y \sin\phi + \Delta y \cos\phi$.

In three dimensions, the rigid-body model is fully described by nine parameters (three each for translation, rotation, and scaling). The rigid-body model maintains angles between arbitrary lines. If two lines in the image intersect at angle θ, they will still intersect at angle θ after the transformation. Linear distortions (shear and perspective transformations) can be introduced to form an extended rigid-body model. Linear distortions can be formulated as matrix operations by using off-diagonal elements. Linear distortions no longer maintain the angles. After the shear transformation, parallel lines will remain parallel. A set of transformations that includes rotation, translation, scaling, and shear is called *affine transformation*. In the most general case, that is, a set of linear transformations that includes all of the above transformations and the perspective transformation, even parallel lines will no longer be parallel after transformation; rather, parallel lines will converge toward a single point after a perspective transformation. The universal description of the shear operation in three

dimensions requires six additional parameters. The shear matrices **H** are defined in

$$\mathbf{H}_x = \begin{bmatrix} 1 & h_{yx} & h_{zx} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \qquad \mathbf{H}_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ h_{xy} & 1 & h_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$\mathbf{H}_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ h_{xz} & h_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.11}$$

In two dimensions, only two parameters h_x and h_y are needed, and the shear matrix **H** becomes

$$\mathbf{H} = \begin{bmatrix} 1 & h_y & 0 \\ h_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11.12}$$

The perspective transformation is controlled by three perspective parameters, p_x, p_y, and p_z. The perspective transformation cannot be implemented in a purely multiplicative form and requires the introduction of an auxiliary variable q. The perspective transformation with the matrix **P** is defined as

$$\begin{bmatrix} qx' & qy' & qz' & q \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.13}$$

and the resulting point Q is scaled by the variable q. Its elements therefore require division by q to obtain the final value of Q.

Multiple transformations can be combined into one matrix, as shown in Equation (11.10), and the general linear transformation model can be described as one single matrix **M**, where all elements are transformation parameters with the exception of the lower-right element, which is always unity. The general linear transformation requires 18 independent parameters in three dimensions and eight parameters in two dimensions. An optimum set of these parameters needs to be found either by manual interaction or by optimization strategies that make use of a registration quality metric (Section 11.3).

11.2. NONLINEAR TRANSFORMATIONS

When the constraint of linearity no longer applies, an infinite number of possible transformations are available. Nonlinear transformations become important when field distortions of an MR scanner need to be compensated or when body motion

(breathing, heartbeat, tissue deformation) needs to be compensated. Under nonlinear transformations, subregions of the image can be subjected to different distortions. There are two types of nonlinear transformations: First, there are transformations where parametric interpolation or approximation functions are applied to distort the image. Second, there are transformations that are obtained from physical models, such as elastic deformation, diffusion, or viscous flow. This set of transformations provides a vector field that describes the displacements down to the pixel level.

In the simplest case, a nonlinear transformation may be performed by a global higher-order polynomial, such as the parabolic model in

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} e_{00} & e_{01} & e_{02} & e_{03} & e_{04} & e_{05} & e_{06} & e_{07} & e_{08} & e_{09} \\ e_{10} & e_{11} & e_{12} & e_{13} & e_{14} & e_{15} & e_{16} & e_{17} & e_{18} & e_{19} \\ e_{20} & e_{21} & e_{22} & e_{23} & e_{24} & e_{25} & e_{26} & e_{27} & e_{28} & e_{29} \end{bmatrix} \cdot \begin{bmatrix} 1 & x & y & z & xy & xz & yz & x^2 & y^2 & z^2 \end{bmatrix}^{\mathrm{T}} \tag{11.14}$$

Polynomial distortions typically occur in inhomogeneous fields (MR) or in insufficiently corrected lens systems. Some are known as *pincushion distortion*, *barrel distortion*, or *lens* distortion. An example is the pincushion distortion shown in Figure 11.2. One of the main challenges of registration by nonlinear transformations can be seen in Equation (11.14). A very large number of parameters exist that need to be optimized. With such a large number of parameters, any search strategy employed to find a global optimum may find multiple local optima of similar quality. For this reason, high-dimensional optimum searches tend to be unstable and strongly dependent on the quality metric.

Instead of global nonlinear transformations, functions can be applied piecewise and locally. The simplest example is called *block matching*. For block-matching registration, the image is subdivided into rectangular blocks and each is subjected to a separate affine or rigid-body spatial transformation. This model is often combined

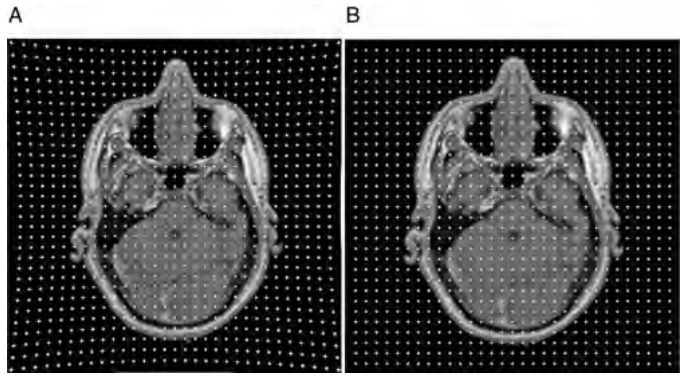


FIGURE 11.2 Pincushion distortion (A) is one example of a spatial distortion that can be corrected (B) with a polynomial transformation similar to Equation (11.14).

with intensity similarity metrics to obtain the registration quality information. Block matching has two disadvantages: First, the distortion between adjoining blocks is not necessarily continuous, and some regularization is needed to eliminate discontinuities (one method is lowpass filtering of the overall displacement field). Second, a lower limit of the block size is quickly reached when the number of voxels in each block becomes too low to provide sufficient information to drive the registration quality metric.

Two popular methods are the transformations using thin-plate splines[19] and B-splines. The displacement function to displace a point p to p' for thin-plate splines is

$$p' = \mathbf{A}p + \mathbf{B} + \mathbf{I}\sum_{i=1}^{N} F_i \left[(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 + d^2\right] \tag{11.15}$$

where $\mathbf{A}$ and $\mathbf{B}$ describe a rigid-body transformation, I is the identity matrix, and the F_i are coefficients determined by solving the equation system set up by all landmark points p_i. In contrast to thin-plate splines, B-splines have local support. This is a desirable property, because any B-spline affects the transformation only locally, and the influence of outliers is locally restricted. A B-spline-based transformation can be defined in analogy to Equation (11.15) as

$$p' = \mathbf{A}p + \mathbf{B} + \mathbf{T}_{\text{local}} \tag{11.16}$$

where $\mathbf{T}_{\text{local}}$ is the local B-spline-based transformation defined as follows. Over the object is superimposed a mesh with X_ϕ by Y_ϕ by Z_ϕ equidistant control points $\phi_{i,j,k}$. The indices i, j, and k and the arguments of the spline function u, v, and w are defined as

$$\begin{aligned} &i = \lfloor x/X_\phi \rfloor - 1; \qquad j = \lfloor y/Y_\phi \rfloor - 1; \qquad k = \lfloor z/Z_\phi \rfloor - 1 \\ &u = 1 + x/X_\phi - i; \qquad v = 1 + y/Y_\phi - j; \qquad w = 1 + z/Z_\phi - k \end{aligned} \tag{11.17}$$

That is, i, j, and k are the integer components of the coordinates of a point $P = (x, y, z)$, and u, v, and w are the corresponding decimal parts. With the basis functions of a cubic B-spline defined as

$$\begin{aligned} B_0(t) &= \frac{(1-t)^3}{6} \\ B_1(t) &= \frac{3t^3 - 6t^2 + 4}{6} \\ B_2(t) &= \frac{-3t^3 - 3t^2 + 3t + 1}{6} \\ B_3(t) &= \frac{t^3}{6} \end{aligned} \tag{11.18}$$

the local component $\mathbf{T}_{\text{local}}$ of the transformation can now be defined:

$$\mathbf{T}_{\text{local}}(x, y, z) = \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l(u)B_m(v)B_n(w)\phi_{i+l,j+m,k+n} \tag{11.19}$$

A complete registration algorithm requires the spatial transformation, a similarity metric, and a strategy to optimize the registration (see Figure 11.1). Rueckert et al.[36] propose an algorithmic approach that is based on the joint image entropy as a similarity metric:

$$C_{\text{sim}} = \frac{H(I_1) + H(I_2)}{H(I_2, I_2)} \tag{11.20}$$

where the entropies $H(I_1)$ and $H(I_2)$ are computed from the histograms of images I_1 and I_2, respectively, and the joint entropy is computed from the joint histogram (cf. co-occurrence matrix). The algorithm consists of the following steps:

Step 1. Set $\mathbf{T}_{\text{local}} = 0$. Find the optimum rigid-body transformation matrices A and B in Equation (11.16) by maximizing the joint entropy in Equation (11.20). This step can be done using either an exhaustive search or a multidimensional optimization strategy.

Step 2. Compute the initial mesh of equidistant control points $\phi_{i,j,k}$ at the coarsest resolution level.

Step 3. Repeat steps 4 to 8 while the control point mesh is larger than the lowest resolution:

Step 4. Repeat steps 5 to 7 for each control point.

Step 5. At the control point, determine the gradient of the similarity metric C_{sim} toward the nonrigid local transformation $\mathbf{T}_{\text{local}}$.

Step 6. Displace the control point by a small distance μ in the direction of the gradient.

Step 7. Recompute the gradient for the displaced control point and repeat steps 5 to 7 until the gradient falls below a preselected threshold. Steps 5 to 7 minimize the overall gradient for each iteration.

Step 8. Interpolate new control points between the existing control points, thus increasing the resolution of the control point mesh.

This example demonstrates two strategies frequently employed in nonrigid, nonlinear transformations. First, initial conditions are often found with the help of rigid-body transformations. A nonlinear, nonrigid transformation often starts with a rigid-body component to find an approximation of the final solution. Second, the nonrigid part proceeds from a coarse to a fine scale. Because of the high number of freedoms at the highest detail level, the optimization follows a multigrid approach, where an initial optimization of a similarity metric is performed on a coarse grid. This grid is then successively refined, followed by a new optimization process at the new resolution, until the desired finest resolution is reached.

At the extreme end of nonlinear registration models is the physical modeling of deformation processes, predominantly elastic deformation and viscous fluid flow. Elastic deformation modeling actually allows us to use parameters with physical relevance, such as the elasticity and deformability of soft tissue. One possible

approach to modeling elastic deformation is to compute the external force F acting on an infinitesimal particle of the object (in the discrete approximation, a pixel or a subvolume) through

$$F(\vec{r}, t) = m\frac{\partial^2 \vec{r}}{\partial t} + D\frac{\partial \vec{r}}{\partial t} + \kappa\frac{\partial \epsilon(\vec{r})}{\partial \vec{r}} \tag{11.21}$$

where m is the mass of the particle, D is the damping constant, $\epsilon(\vec{r})$ represents the elastic deformation energy, and κ is a proportionality constant for the elastic deformation component. In equilibrium, the force F is balanced by a force term that is obtained from a similarity metric and drives the registration. Alternatively, elasticity can be modeled using the Navier–Lamé partial differential equation for linearly elastic materials:

$$\mu\nabla^2 u(\vec{r}) + (\mu+\lambda)\nabla(\mathrm{div}u(\vec{r})) + f(\vec{r}) = 0 \tag{11.22}$$

where $u(\vec{r})$ is the displacement vector at location $\vec{r}$, μ and λ are the Lamé constants, and $f(\vec{r})$ is the external force that is imposed on the system by the similarity metric.[19] A large number of algorithmic approaches to solving elastic models exist. Key numerical methods are presented in detail by Modersitzki.[29] In addition, finite-difference[2] and finite-element[17] approaches have been employed.

Solving the Navier–Lamé equation assumes small displacements. A limitation of the elastic approach is therefore the restriction to images that need only small deformations to register. If large deformations are necessary, deformation can be achieved by modeling viscoelastic fluid flow. Viscous fluid flow can be described by the Navier–Stokes–Duhem equation:

$$\mu\nabla^2 v(\vec{r}) + (\mu+\lambda)\nabla(\mathrm{div}v(\vec{r})) + f(\vec{r}) = \rho\frac{\partial v(\vec{r})}{\partial t} + \nabla P \tag{11.23}$$

where ρ is the specific density, $v(\vec{r})$ the fluid velocity, and P the pressure. If flow is assumed to be sufficiently slow, the flow term on the right-hand side becomes very small and Equation (11.23) simplifies to

$$\mu\nabla^2 v(\vec{r}) + (\mu+\lambda)\nabla(\mathrm{div}v(\vec{r})) + f(\vec{r}) = \nabla p \tag{11.24}$$

Under the assumption of a homogeneous hydrostatic pressure, ∇P becomes very small, and the Navier–Stokes equation of a compressible fluid emerges:

$$\mu\nabla^2 v(\vec{r}) + (\mu+\lambda)\nabla(\mathrm{div}v(\vec{r})) + f(\vec{r}) = 0 \tag{11.25}$$

This equation has the same structure as Equation (11.22), and similar solution approaches can be taken.[29]

A complete algorithm that includes fluid flow deformation as one step for image registration with major displacements was introduced by Christensen et al.[11] The main

goal was to provide a stable method for intersubject registration of neuroanatomical structures. In the first step, manually aided and landmark-based rigid-body transformations are applied to produce an approximate global match. This step is followed by fluid-elastic deformation, that is, numerically solving the nonlinear partial differential equations describing the fluid deformation model. The cost function that drives registration is the squared image value mismatch between the floating image and the reference image. From the cost function, the driving force in Equation (11.25) can be computed as the gradient-emphasized absolute difference between the deformed floating image A and the reference image B,

$$f(\vec{r}) \propto -|A(\vec{r} - \Delta\vec{r}) - B(\vec{r})| \nabla A(\vec{r} - \Delta\vec{r}) \tag{11.26}$$

where the gradient ∇A is responsible for providing the direction of the deforming force.

Special consideration is given to numerical stability: By monitoring the determinant of the Jacobian of the transformation, deformation fields can be avoided where the displacement fields become singular. In such a case, the deformed image is regridded by interpolation and the process is restarted. The main disadvantage of this process is the computational complexity. To reduce this complexity, Bro-Nielsen and Gramkow[10] described an approach where the solution of a set of differential equations can be approximated by a convolution filter, which resulted in an order-of-magnitude speed improvement.

11.3. REGISTRATION QUALITY METRICS

In Section 11.2, the importance of the registration quality metric was highlighted as the force that drives the registration process. Registration quality can be determined globally (for rigid-body transformations and for an overall assessment) and locally (for nonlinear deformation models). A number of methods have been developed to determine if two images are registered, among them the use of fiducial markers and landmark points, registering of the segmented object surface, two-dimensional gray-scale histograms, and a chamfer matching technique. The purpose of registration quality metrics is to provide an optimizing algorithm with guidance as to where to find the optimum. Suitable registration quality metrics are therefore one- or multidimensional functions that reach a global extremum when the two images are optimally registered.

11.3.1. Fiducial Markers

Fiducial markers are small objects of high image contrast that can be attached to the skin, provided with a solid casing, a stereotactic frame, or in extreme cases even screwed into the bone. The advantage of fiducial markers is the possibility to determine their location automatedly and the relatively straightforward implementation of registration quality metrics. The disadvantage of fiducial markers attached to the skin

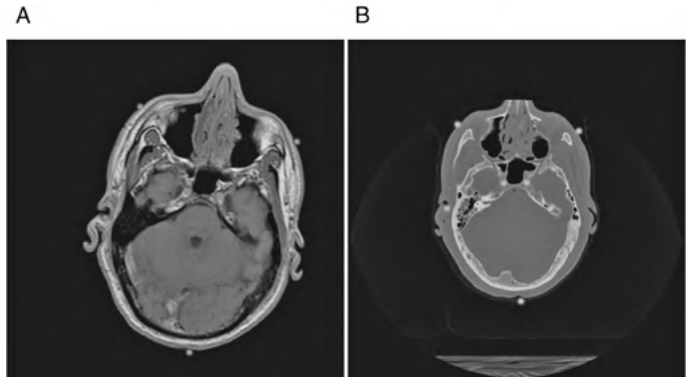


FIGURE 11.3 Matched, but unregistered slices from the Visible Human data set. (A) shows a T_1-weighted MR slice of the head, and (B) shows the corresponding CT slice. The images are mismatched in position, rotational orientation, and scale. Anatomy-based homologous fiducial markers (red in the MR image and green in the CT image) have been placed manually. (*See insert for color representation of the figure*.)

is the possibility that skin deformation displaces the markers between acquisitions. Usually, fiducial markers are used in conjunction with rigid-body transformations to register images. An example in two dimensions is presented in Figure 11.3, which shows matching MR (A) and CT (B) head slices from the Visible Human data set. Anatomy-based fiducial markers have been placed manually. The images show a registration mismatch in terms of rotation, position, and scale.

A simple procedure to register two images on the basis of fiducial markers can be devised that provides acceptable results in two dimensions. The procedure is described by the following three steps:

Step 1. Find the centroids of the fiducial markers P_c and Q_c by averaging the coordinates of the fiducial markers P_i and Q_i. Translate the floating image so that $Q_c = P_c$.

Step 2. Determine the angles of the lines connecting the fiducial markers with the centroid, φ_i for the reference image and θ_i for the floating image, and rotate the floating image by $\bar{\theta} - \bar{\varphi}$, where is $\bar{\theta}$ the average of the θ_i, and $\bar{\varphi}$ is the average of the φ_i.

Step 3. The average ratio of the lines that connect the fiducial markers to their centroids provides the isotropic scaling factor s.

The steps in the registration process are illustrated in Figure 11.4, where the MR image (Figure 11.4A) is the floating image and the CT image (Figure 11.4B) is the reference image. The centroid of the fiducial markers is determined and brought to match with a translation operation (Figure 11.4A and B). With a suitable scaling and rotation operation, the fiducial markers are brought in close proximity (Figure 11.4C). The fused and false-colored images (Figure 11.4D) show the images in registration.

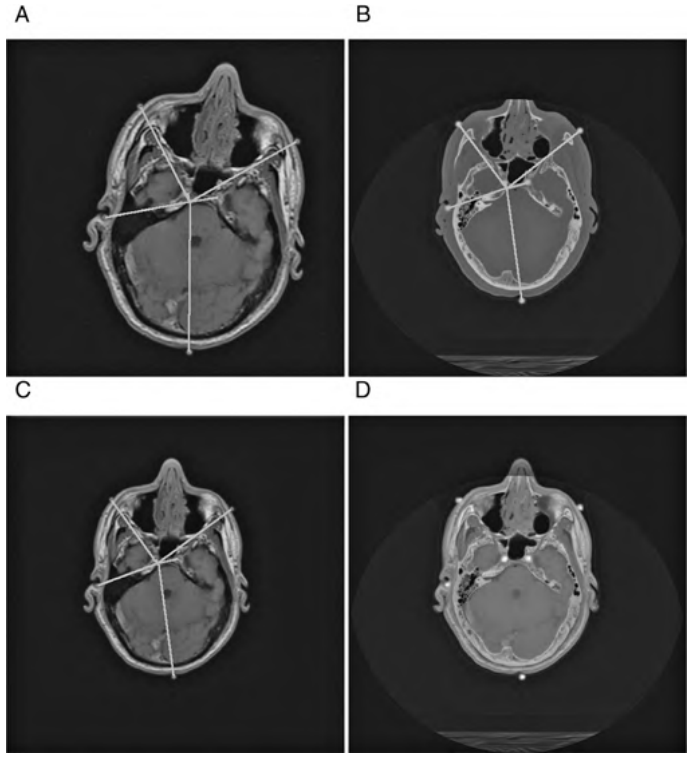


FIGURE 11.4 Registration process based on fiducial markers. First, the centroid of the markers is determined (green lines in A and B). Next, the centroids are brought in congruence with a translation operation. The angles of the green lines provide the necessary rotation angle, and the length ratio provides the scaling factor. After the transformation, the MR image (C) matches the CT image (B) in terms of position, rotation, and scale. It can be seen that the green lines, which connect the fiducial markers with the centroid, have the same length and orientation in (B) and (C). The images are then fused (D), with the MR image having a red tint and the CT image having a blue tint. A minor mismatch can still be seen; this mismatch cannot be corrected with rigid-body transformations. (*See insert for color representation of the figure*.)

More advanced solutions to the registration problem in three dimensions normally involve a least-squares approach, such as the solutions presented by Schönemann and Carroll[38] and Arun et al.[1] Seven parameters (three translations, three rotations, and one isotropic scaling operation) are possible and combined in the transformation matrix $\mathbf{M}$. A set of k homologous fiducial markers is available, defining the sets of points $\mathbf{P} = \{P_1, P_2, \ldots, P_k\}$ in the reference image and $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ in the floating image. The transformation with $\mathbf{M}$ gives rise to the set of equations

$$P_i = \mathbf{M}Q_i + \epsilon_i \tag{11.27}$$

where $1 \leq i \leq k$ and ϵ_i accounts for both the registration error and possible random influences, such as the shift of a fiducial marker between acquisitions. The registration strategy is to find a transformation matrix $\mathbf{M}$ such that the squared error E (i.e., the sum of ϵ_i^2) is minimized:

$$E = \sum_{i=1}^{k} (P_i - \mathbf{M}Q_i)^2 \rightarrow \min \tag{11.28}$$

In the definition of the registration error E [Equation (11.28)], a weight factor w_i can be introduced for each landmark point to account for a variable confidence in that specific landmark point. In this more general case, Equation (11.28) would be rewritten as

$$E = \sum_{i=1}^{k} [w_i(p_i - \mathbf{M}Q_i)]^2 \rightarrow \min \tag{11.29}$$

The transformation matrix $\mathbf{M}$ can be found by finding, in this order, the rotation matrix $\mathbf{R}$, the scaling matrix $\mathbf{S}$, and the translation matrix $\mathbf{T}$. The algorithm is based on the *Procrustes problem*,[18] which implies that a solution for Equation (11.29) exists if $\mathbf{M}$ is a matrix of rigid-body transformations. The following three steps describe the algorithm:

Step 1: Find the rotation matrix.

(1a) Define the centroids of the fiducial markers $P_c = (x_{pc}, y_{pc}, z_{pc})$ for the reference image and $Q_c = (x_{qc}, y_{qc}, z_{qc})$ for the floating image through

$$\begin{aligned} P_c &= \frac{\sum_{i=1}^{k} w_i P_i}{\sum_{k=1}^{N} w_i} \\ Q_c &= \frac{\sum_{i=1}^{k} w_i Q_i}{\sum_{k=1}^{N} w_i} \end{aligned} \tag{11.30}$$

and the central positions of the fiducial markers (i.e., the positions $\bar{P}_i$ and $\bar{Q}_i$ relative to the centroid) through

$$\bar{P}_i = P_i - P_c \qquad \bar{Q}_i = Q_i - Q_c \tag{11.31}$$

(1b) Compute the covariance matrix $\mathbf{C}$ through

$$\mathbf{C} = \sum_{i=1}^{k} w_i^2 \left(\bar{P}_i \bar{Q}_i^{\mathrm{T}}\right) \tag{11.32}$$

(1c) A singular value decomposition yields the unitary matrices $\mathbf{U}$ and $\mathbf{V}$ with the matrix of singular values Σ

$$\mathbf{C} = \mathbf{U}\,\Sigma\,\mathbf{V}^{\mathrm{T}} \tag{11.33}$$

A method for singular value decomposition can be found in *Numerical Recipes in C*.[35]

(1d) The rotation matrix $\mathbf{R}$ can now be computed:

$$\mathbf{R} = \mathbf{V}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{VU}) \end{bmatrix}\mathbf{U}^{\mathrm{T}} \tag{11.34}$$

Step 2. Determine the isotropic scaling matrix $\mathbf{S}$ with $s_x = s_y = s_z$:

$$\mathbf{S} = \frac{\sum_{i=1}^{k} w_i^2 \left(\mathbf{R}\bar{P}_i\right)\bar{Q}_i}{\sum_{i=1}^{k} w_i^2 \left(\mathbf{R}\bar{P}_i\right)\bar{P}_i} \tag{11.35}$$

Step 3. Determine the translation vector T from the centroids P_c and Q_c by using

$$T = Q_c - \mathbf{SR}P_c \tag{11.36}$$

The matrix operations for this algorithm differ slightly from the matrix operations laid out in Section 11.1 as the matrices $\mathbf{S}$ and $\mathbf{R}$ are 3×3 matrices for a three-dimensional operation, and the translation operation is performed in an additive manner with the vector T. The points P_i can now be registered with the points Q_i of the reference image through

$$P_i' = \mathbf{SR}P_i + T \tag{11.37}$$

At this point, it is efficient to extend the matrices $\mathbf{S}$ and $\mathbf{R}$ to 4×4 matrices and convert the vector T into a translation matrix $\mathbf{T}$ as in Equation (11.4), because now the final transformation matrix $\mathbf{M} = \mathbf{TSR}$ can be determined, and the transformation of each pixel can be performed with a single matrix operation.

The quality of the registration (measured by the squared error E) depends strongly on the number and placement of the fiducial markers. As with any statistical measurement, a larger number of samples (in this case, a larger number of markers) reduce the overall error caused by random displacement (placement inaccuracy and between-acquisition displacements). Some fine-tuning of the registration is possible by varying the weight factors w_i to reduce the influence of markers that show a stronger misregistration than other markers. Ambiguous results are possible if the markers are placed in a regular fashion: for example, if all markers are co-planar or if the markers are spaced regularly.

11.3.2. Anatomical Markers

The advantage of fiducial markers is the possibility of using automated segmentation to obtain the locations of the markers in an unsupervised fashion. However, fiducial markers must be placed before the study, and it is generally not possible to remove and reattach the markers between studies. Instead of external fiducial markers, anatomical landmarks can be used. Anatomical landmarks are unambiguously identifiable anatomical structures, such as bifurcations of blood vessels, the eyeballs, the cochlea, or points with high curvature on bones. The advantage of using anatomical landmarks is the ability to identify them poststudy, but generally the registration error is larger than with fiducial markers, and usually the anatomical landmark points are extracted manually. The strategy to minimize the registration error is the same, however, and the algorithm to determine the transformation matrix $\mathbf{M}$ is the same as for fiducial markers. Recently, strategies have been developed to extract landmark points in a more automated fashion. Two tangential circles, perpendicular to each other, can be inscribed to any convex point of a surface. Those points where the inscribed circles have minimal radius (corresponding to maximal curvature) are extremal points and can be used as landmark points.[30] Alternatively, extremal points on the surface can be found by varying the segmentation threshold.[41] If the feature-to-background transition is relatively smooth, the thresholded surface can be displaced outward by decreasing the segmentation threshold. Crest lines on the surface can be defined as lines with maximal curvature perpendicular to the line. As the threshold is decreased, the crest lines move outward, and multiple crest lines form another surface that is perpendicular to the object surface. A second crest surface, perpendicular to both the object surface and the first crest surface, can be obtained in the same manner. The intersection of the two crest surfaces with the object surface at one specific threshold defines an extremal point that can be used as a landmark point.[41]

11.3.3. Surface and Volume Matching

If a three-dimensional volume has a well-defined surface, registration can be performed with surface- and volume-matching techniques. For the registration of MR and PET images, Dhawan et al.[15] proposed a method to register the volume based on their principal axes. Principal axes (such as the major and minor axes of an ellipse) are perpendicular, and a procedure exists to find them (principal axis theorem). If two shapes can be approximated by an ellipsoid, a good degree of registration can be achieved if the respective principal axes are brought in congruence. If the centroid of the shape (i.e., a set of pixels belonging to a three-dimensional region $\Re$) is known as $P_c = (x_c, y_c, z_c)$, then the principal axes of the shape are the eigenvectors of the inertia matrix $\mathbf{I}$,

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{11.38}$$

with the components

$$\begin{aligned} I_{xx} &= \sum_{\Re} \left[(y-y_c)^2 + (z-z_c)^2\right] \\ I_{yy} &= \sum_{\Re} \left[(x-x_c)^2 + (z-z_c)^2\right] \\ I_{zz} &= \sum_{\Re} \left[(x-x_c)^2 + (y-y_c)^2\right] \\ I_{xy} &= I_{yx} = \sum_{\Re} \left[(x-x_c)(y-y_c)\right] \\ I_{xz} &= I_{zx} = \sum_{\Re} \left[(x-x_c)^2(z-z_c)\right] \\ I_{yz} &= I_{zy} = \sum_{\Re} \left[(y-y_c)(z-z_c)\right] \end{aligned} \tag{11.39}$$

Here the summation takes place over all pixels that belong to the region $\Re$. Let $\mathbf{E}$ be the normalized eigenvector matrix and $\mathbf{R}$ the rotation matrix with $\mathbf{R} = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z$; then the identity

$$\mathbf{R} = \mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \tag{11.40}$$

yields the rotation angles ϕ_x, ϕ_y, ϕ_z through[15]

$$\begin{aligned} \phi_y &= \sin^{-1} e_{31} \\ \phi_x &= \sin^{-1}\left(-\frac{e_{21}}{\cos\phi_y}\right) \\ \phi_z &= \sin^{-1}\left(-\frac{e_{32}}{\cos\phi_y}\right) \end{aligned} \tag{11.41}$$

By rotating the volume $\Re$ by ϕ_x, ϕ_y, and ϕ_z, the principal axes are now aligned to coincide with the x, y, and z axes. The same operation can be done with the second volume; and with a translation operation along the difference between the centroids, the centroids can be brought to congruence. Finally, an isotropic scaling matrix $\mathbf{S}$ with $s = s_x = s_y = s_z$ can be determined by computing the cube root of the ratio of volumes.

The method of principal axes registration was further refined by Dhawan et al.[15] into an iterative procedure that has the advantage of being able to register partial volumes. The IPAR (iterative principal axes registration) procedure is applicable whenever a field of view (e.g., from a PET scanner) covers the reference volume only partially. The idea behind the IPAR algorithm is to apply a field of view to the reference image. This field of view is iteratively refined until it matches the field of view of the PET image, registering both images in the process.

To match two surfaces, Pelizzari et al.[33] proposed a method described as fitting a hat to a head. Here, the head is used in a analogy to the object's surface in the

reference image, and the hat is an analogy of the object's surface in the floating image. The reference surface is created by outlining the object in each slice. Therefore, the head analogy is a stack of convex contours that approximate the closed three-dimensional surface of the reference object. Conversely, the hat analogy is created from the surface of the floating image as a set of independent points on the surface. Rather than representing the object's outline by a closed contour (as in the head), the floating object is represented by individual points along the contour and their centroid. For each of these points P_i, a line extends from the centroid through P_i, and the intersection of that point with the head (the reference surface), Q_i, is computed. For this purpose, the ray-tracing algorithm by Siddon[40] is proposed. The task now is to find a transformation $\mathbf{M}$ so that the registration error E, as defined in Equation (11.28), is minimized. Pelizzari et al.[33] do not explain their optimization approach, but rather hint at an exhaustive search of the parameter space, aided by some manual intervention during the parameter search. However, systematic optimization methods for multidimensional problems have become available, most notably Powell's multidimensional minimization method.[34] The Powell method performs an iterative series of one-dimensional optimizations, moving from the optimum of one parameter along a perpendicular (or conjugate) direction to find the optimum solution for the next parameter. This process is repeated for each parameter in turn before the algorithm returns to the first parameter. The algorithm converges when the difference of the error E between two iterations falls below a preset threshold. An excellent explanation with sample computer code for an implementation of Powell's method can be found in *Numerical Recipes in C*.[35] The combination of the head-and-hat method with the Powell optimization was proposed by Levin et al.[25]

Another popular algorithm for the optimization of a registration metric is the simplex algorithm. Like Powell's method, the simplex method is a multidimensional optimum finder, but the actual algorithm differs fundamentally from Powell's algorithm. Rather, the Simplex can be thought of as a creature that has $N + 1$ feet in a space of N independent parameters.[35] To initialize the algorithm, one of the Simplex's feet is placed at the location of an initial guess; the other feet may be placed a certain distance λ away from the initial guess. Thus, in two dimensions, the feet form a triangle, and in three dimensions, a tetrahedron, and λ is some length scale that also needs to be estimated initially. Let us assume that the Simplex wants to climb downhill in its N-dimensional space to find a minimum that is considered the optimum. The simplex algorithm lets the Simplex find its least optimally placed foot (i.e., the highest-placed foot) and move it to a better position, ideally by reflecting it at the N-dimensional hyperplane spanned by the better-placed feet. This reflection step is designed to maintain the volume of the Simplex and avoid degeneration (i.e., moving of all feet into the same location). Whenever possible, the Simplex is allowed to take larger steps, thus expanding its volume. However, once the Simplex reaches a valley floor, the algorithm lets the Simplex contract. The simplex algorithm is considered converged if the Simplex volume drops below a predefined threshold (i.e., its feet are within a certain range of each other). Like the Powell algorithm, the simplex algorithm may get caught in a local minimum. For this reason, the simplex

algorithm can be used at different resolutions to enhance its chances of finding the global optimum. Bernon et al.[5] compared Powell's method and the simplex method in conjunction with chamfer matching and found that the downhill simplex method provided better registration results.

11.3.4. Chamfer Matching

Chamfer matching is a surface-based registration method. Chamfer matching was introduced by Barrow et al.,[4] and its performance in multimodality registration was examined by Jiang et al.[23] Chamfer matching is based on a cost function that uses the Euclidean distance of each object voxel from the nearest surface voxel. The distance transform is applied only to the reference image, and instead of using the computationally expensive Euclidean distance map, the chamfer 3/4/5 transform[8] produces an adequate cost image. For the purpose of computing the cost image, the reference image A needs to be segmented, and all surface pixels are initially set to zero, whereas both the background pixels and interior pixels are initially set to unknown. Both background pixels and interior pixels are scanned along all three dimensions (x is fastest, z is slowest), thereby setting each unknown pixel to the minimum of its forward-diagonal elements according to the formula

$$I(x,y,z) = \min \begin{cases} I(x+1,y-1,z-1)+5 & I(x+1,y,z-1)+4 \\ I(x+1,y+1,z-1)+5 & I(x,y+1,z-1)+4 \\ I(x+1,y-1,z)+4 & I(x+1,y,z)+3 \\ I(x+1,y+1,z)+4 & I(x,y+1,z)+3 \\ I(x+1,y-1,z+1)+5 & I(x+1,y,z+1)+4 \\ I(x+1,y+1,z+1)+5 & I(x,y+1,z+1)+4 \\ I(x,y,z+1)+3 & I(x,y,z) \end{cases} \tag{11.42}$$

In a second pass, following the opposite direction (x is fastest, but from right to left, y from bottom to top, and z is slowest, from back to front), the pixels are set according to the modified formula

$$I(x,y,z) = \min \begin{cases} I(x-1,y+1,z-1)+5 & I(x-1,y,z-1)+4 \\ I(x-1,y-1,z-1)+5 & I(x,y-1,z-1)+4 \\ I(x-1,y+1,z)+4 & I(x-1,y,z)+3 \\ I(x-1,y-1,z)+4 & I(x,y-1,z)+3 \\ I(x-1,y+1,z+1)+5 & I(x-1,y,z+1)+4 \\ I(x-1,y-1,z+1)+5 & I(x,y-1,z+1)+4 \\ I(x,y,z-1)+3 & I(x,y,z) \end{cases} \tag{11.43}$$

The two passes are repeated until no unknown pixels are left. As a consequence, the image value of interior and background pixels increases with their distance from the surface. To compute the cost of a transform $\mathbf{M}$, the root mean square of all cost values of the distance-transformed image A is determined for all transformed pixels of the

surface of the reference image B. If the set of surface pixels of the floating image B is Ω and the corresponding set of pixels transformed into A-space is $\mathbf{M}\Omega$, the total cost C for transform $\mathbf{M}$ is defined by[23]

$$C(\mathbf{M}) = \left[\frac{1}{N} \sum_{p_i \epsilon \mathbf{M}\Omega} \min \left\{ t^2, I_A^2(\mathbf{M}p_i) \right\} \right]^{1/2} \tag{11.44}$$

where the p_i are the surface pixels from surface Ω, N the number of these surface pixels, and t a threshold value introduced to reduce the influence of outlier points in either image. The parameter space of $\mathbf{M}$ can be scanned exhaustively to find the global minimum of C. To improve computational efficiency, a multiresolution approach is recommended. Further refinement of the transformed position of the floating image is possible by gradually reducing the threshold t. Furthermore, interpolation of the cost function in the reference image allows for positioning with subpixel accuracy. One advantage of chamfer matching is its computational efficiency. The cost function can be computed beforehand and remains unchanged during the matching process. The computation of the cost function involves the summation over surface voxels only, which generally are substantially fewer than the voxels in a filled volume. A multigrid approach and the separation of parameter space further improves efficiency. A demonstration of the chamfer-matching process in given in Figure 11.5.

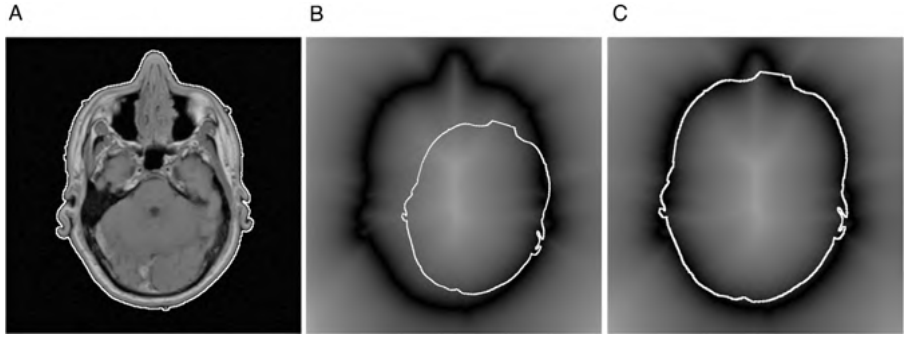


FIGURE 11.5 Example of a chamfer-matching process. The reference image (A) is the MR slice of the Visible Human head with its outline emphasized as a white line. The distance map is created from that outline, with intensity values increasing with their distance from the outline (B). Also seen in image B is the outline of the floating image, the corresponding CT slice. The CT slice has a mismatch of scale, translation, and rotation. Therefore, the average pixel value of distance map pixels underneath the white outline of the floating image is relatively high (B). In image C a search of the parameter space has provided improved scaling, rotation, and translation parameters. The average pixel value of distance map pixels underneath the white outline of the floating image is lowered as the images approach registration.

11.3.5. Intensity-Based Matching

Intensity-based methods build on the assumption that homologous regions of the images to be registered have similar intensities. This assumption gives rise to using the co-occurrence matrix as a registration quality metric. Unlike the co-occurrence matrix definition used in Section 8.2 that uses the same image with a displacement, the gray-level co-occurrence matrix used for image registration obtains the gray-value counts from two images. If both images have 256 gray levels, the co-occurrence matrix has a size of 256 $\times$ 256 pixels. Two identical images, perfectly registered, have nonzero values only along the diagonal of their co-occurrence matrix, whereas images that are not registered have a wider dispersion of nonzero values.

Intensity-based matching is well suited to register images of the same modality, because the physical basis of image intensity is the same. Registration of different modalities is not straightforward. A CT image will have high intensity values for bone, whereas bone has very low values in MRI. One possibility to adjust the image values in this special case is to subject the CT image to a triangular intensity lookup table. Such a lookup table ensures that soft tissue, originally midlevel gray in CT, is assigned the highest image values, and bone, originally the tissue with the highest image values in CT, matches the low image values of bone in MRI. An example is shown in Figure 11.6, where a CT image (B) was subjected to a suitable lookup table (C) to better match the corresponding MR image (A).

The gray-level co-occurrence matrix for the images in Figures 11.6A and C is shown in Figure 11.7. In the unregistered case, Figure 11.7A, a relatively wide dispersion of nonzero values can be seen. In particular, high values near the left or top borders of the co-occurrence matrix indicate a mismatch, where a large number of feature pixels in one image coincide with background pixels in the other image. When the images are registered, the co-occurrence matrix is less dispersed (Figure 11.7B).

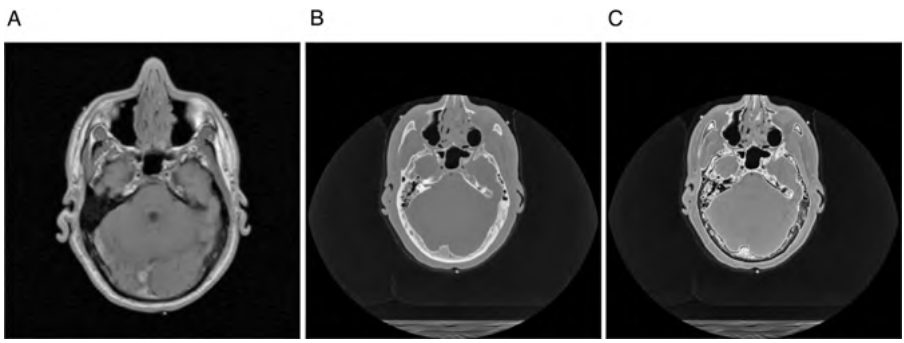


FIGURE 11.6 Intensity-based registration of MR images (A) with CT images (B). CT image values have a different physical meaning than MR image values. Bone, for example is bright in CT and dark in MR. A triangular lookup table converts midlevel gray values into high-level values, and high-level gray values (bone) into low values. This lookup table creates a remapped CT image (C) with an approximation of the MR intensity distribution.

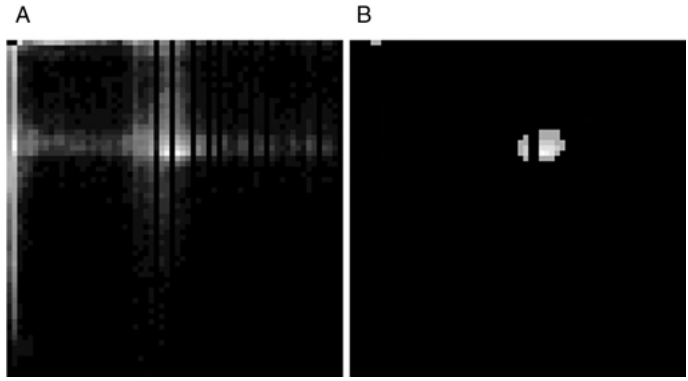


FIGURE 11.7 Gray-level co-occurrence matrix for the MR image in Figure 11.6A and the CT image in Figure 11.6C. Image A shows an unregistered case with a translation and scale mismatch, and image B shows the co-occurrence matrix of the registered images.

Some of Haralick's feature metrics (see Section 8.2) can be used to quantify the quality of the registration process. Particularly suitable are the texture features that weigh the matrix elements by their distance from the diagonal, such as contrast, inertia, and inverse difference. The inverse difference moment of the co-occurrence matrix in Figure 11.7A is 29, whereas the inverse difference moment of the matrix in Figure 11.7B is 1029. The normalized inertia of the matrix in Figure 11.7A is 159, whereas the same metric for the matrix in Figure 11.7B is 21. Another frequently employed similarity metric is the joint entropy that was introduced in Equation (11.20). Two representative metrics, the correlation and inertia, are presented in Figure 11.8 as a function of horizontal and vertical translation and of rotation. A clear minimum of the inertia, and a corresponding maximum of the correlation, can be seen when the images are in registration. By using a suitable similarity metric, a multidimensional minimum search can be performed analogous to the error minimum in the head-and-hat surface matching method.

11.4. INTERPOLATION METHODS FOR IMAGE REGISTRATION

The transformation matrix **M** used to register two images generally requires access to image elements at noninteger coordinates. To transform an image, it is convenient to invert the translation matrix:

$$P_i' = M P_i \leftrightharpoons P_i = M^{-1} P_i' \tag{11.45}$$

The reason to invert the matrix **M** is the more straightforward implementation of the transformation. In a triple loop over all integer coordinates x, y, and z of the transformed image (i.e., each point P_i'), the coordinate P_i in the nontransformed image is computed through Equation (11.45) with $\mathbf{M}.^{-1}$ In the general case, the

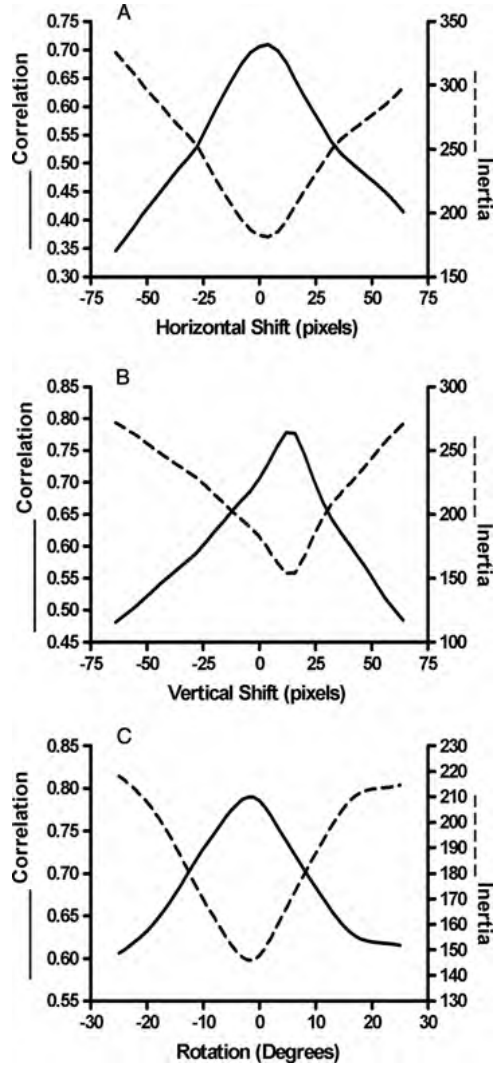


FIGURE 11.8 Determination of the quality of registration by gray-level correlation metrics. Shown are the correlation (solid line) and the inertia (dashed line) of the co-occurrence matrix of the MR and CT images in Figure 11.6. The CT image was scaled up by 33% beforehand to match the size of the MR image. All three rigid-body transformations [horizontal translation (A), vertical translation (B), and rotation (C)] show a clear optimum when the images are in registration. Correlation shows a global maximum, and inertia shows a corresponding global minimum. Multidimensional search strategies can be employed to find the optimum points automatically.

resulting point P_i contains noninteger elements. An image can be assumed to be composed of pixel-sized tiles or cubes of constant gray value. Under this assumption, the image value at any point in space is known: namely, the image value of its nearest integer neighbor, a scheme called *nearest-neighbor interpolation*. Although it is computationally efficient, nearest-neighbor interpolation results are prone to noise and blocking artifacts. Particularly with repeated interpolation, the nearest-neighbor principle shows poor results. For purposes of interpolation, the image values are therefore assumed to be known only on integer coordinates, and some interpolation scheme provides estimates of the image value on noninteger coordinates. Linear interpolation provides a smooth transition of the gray value from one vertex to the next. Linear interpolation is arguably the most frequently used interpolation scheme in pixel-based image editing programs and many image processing software packages. Linear interpolation assumes that the image value changes linearly from one pixel to the next. In one dimension, the interpolated function value $f_{\text{intp}}(t)$ at a noninteger position of t can be obtained by

$$f_{\text{intp}}(t) = af(\lfloor t+1 \rfloor) + (1-a)f(\lfloor t \rfloor) \tag{11.46}$$

where $\lfloor \cdot \rfloor$ indicates the integer operation, and $a = t - \lfloor t \rfloor$ is the decimal part of t. In two dimensions, four neighbors are needed, and with the definition of $a = x - \lfloor x \rfloor$ and $b = y - \lfloor y \rfloor$, the interpolation scheme to obtain the interpolated image value $I_{\text{intp}}(x,y)$ on noninteger coordinates x and y is a two-step process called *bilinear interpolation*:

$$\begin{aligned} I_{x1} &= aI(\lfloor x+1 \rfloor, \lfloor y \rfloor) + (1-a)I(\lfloor x \rfloor, \lfloor y \rfloor) \\ I_{x2} &= aI(\lfloor x+1 \rfloor, \lfloor y+1 \rfloor) + (1-a)I(\lfloor x \rfloor, \lfloor y+1 \rfloor) \\ I_{\text{imtp}}(x,y) &= bI_{x2} + (1-b)I_{x1} \end{aligned} \tag{11.47}$$

In three dimensions, the process is identical, except that eight neighboring image values on integer coordinates are used (vertices of a cube around the noninteger point), and the interpolation uses four interpolations in x, two interpolations in y, and one interpolation in z (trilinear interpolation).* Whereas linear interpolation is computationally very efficient, the image value planes that are spanned between adjoining integer pixels are not differentiable over adjoining squares. Furthermore, linear interpolation acts like a lowpass filter.[31] More sophisticated interpolation methods are recommended.

Cubic interpolation provides a differentiable hypersurface over adjoining squares or cubes. For cubic interpolation, a third-order polynomial is fitted into four consecutive data points in one dimension. In analogy to the weight factor a in Equation (11.46), four weight factors a_0, a_1, a_2, and a_3 are needed to obtain the interpolated value from the four closest neighbors. The weight factors in one dimension or in the

*It can be shown that the order of the interpolation steps does not matter. The sequence of interpolations suggested (first in the x-direction, followed by the y-direction, finally followed by the z-direction) is chosen arbitrarily and may be changed without affecting the result.

x-direction can be computed through

$$\begin{aligned} a_1(\zeta) &= \alpha(-\zeta^3 + 2\zeta^2 - \zeta) \\ a_2(\zeta) &= (2-\alpha)\zeta^3 - (3-\alpha)\zeta^2 + 1 \\ a_3(\zeta) &= -(2-\alpha)\zeta^3 + (3-2\alpha)\zeta^2 - \alpha\zeta \\ a_3(\zeta) &= \alpha(\zeta^3 - \zeta^2) \end{aligned} \tag{11.48}$$

where ζ is the fractional part of the noninteger coordinate x. A common choice of the parameter α is $\alpha = 0.5$. In one dimension (the x dimension), the interpolated value $I_{X,k}$ is

$$I_{X,k} = \sum_{i=1}^{4} a_i(\zeta) I(\lfloor x - 2 + i \rfloor, y_k) \tag{11.49}$$

In one dimension, k is irrelevant. In two dimensions, Equation (11.49) needs to be applied four times for $1 \leq k \leq 4$ and $y_1 = \lfloor y \rfloor - 1$ to $y_4 = \lfloor y \rfloor + 2$. Next, a second set of weight factors a_1 through a_4 is computed through Equation (11.48) with $\zeta = y - \lfloor y \rfloor$ and the final interpolated value is obtained with

$$I_{\text{intp}}(x,y) = \sum_{k=1}^{4} a_k(\zeta) I_{X,k} \tag{11.50}$$

In three dimensions, the interpolated point is in the center region of a cube with $4 \times 4 \times 4$ vertices. The first step involves the computation of 16 vertices interpolated in the x-direction, from which four vertices are interpolated in the y-direction, and the final point determined by interpolation in the z-direction. It can be seen that the computational effort for cubic interpolation is markedly higher than for linear interpolation.

Following Shannon's sampling theory, a signal $f(t)$ can be reconstructed from sampled values f_k with a sinc interpolation function:

$$f(t) = \sum_{k=-\infty}^{\infty} f_k \,\text{sinc}\,[\pi(\text{t} - \text{k})] \tag{11.51}$$

where $\text{sinc}(t) = (\sin t)/t$. The sinc function has several desirable properties. It is unity at the origin and vanishes for all integer arguments. Extending Equation (11.51) to three dimensions, we obtain

$$f(x,y,z) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} f_{i,j,k} \,\text{sinc}\,[\pi(x-i)]\,\text{sinc}\,[\pi(y-j)]\text{sinc}\,[\pi(z-k)] \tag{11.52}$$

where x, y, and z may be noninteger coordinates and i, j, and k are integer numbers that define the sampling coordinates. One difficulty associated with the sinc interpolation function is the infinite support. Theoretically, the summation takes place over an infinite volume, although the image occupies only a finite volume. In practice, a reasonable cutoff is chosen. However, this cutoff can still extend beyond the image boundaries, in which case suitable boundary conditions need to be applied, such as the mirror boundary condition, where, for example, $I(x,y,z) = I(-x,y,z)$ when $x < 0$. Although sinc-based interpolation methods have been proposed for signal and image interpolation, particularly in combination with the Fourier transform,[46] a sinc interpolator has a tendency to produce ringing, that is, high-frequency local oscillations.

It can be seen from Equation (11.51) that interpolation is actually a convolution operation. In fact, the interpolation of a discrete set f_k with an interpolation function $\varphi(t)$ can be expressed as a convolution operation defined by

$$f(t) = \sum_{k=-\infty}^{\infty} f_k \varphi(k - t) \tag{11.53}$$

If the interpolation function has finite support, the summation reduces to the range of the support of φ, thus reducing computational complexity. Notably, even the nearest-neighbor, linear, and cubic interpolation schemes can be expressed in terms of Equation (11.53) when the following interpolation functions are used:

$$\begin{aligned}
\varphi_{\mathrm{NN}}(t) &= \begin{cases} 1 & \text{for } |t| \le 0.5 \\ 0 & \text{otherwise} \end{cases} \\
\varphi_{\mathrm{lin}}(t) &= \begin{cases} 1 - |t| & \text{for } |t| \le 1 \\ 0 & \text{otherwise} \end{cases} \\
\varphi_{\mathrm{cub}}(t) &= \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & \text{for } |t| \le 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & \text{for } 1 < |t| \le 2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{11.54}$$

In the cubic interpolation function, the parameter $a < 0$ determines the amount of negative "overshoot" near ± 1 and can be used to approximate the sinc function. By inserting $\varphi_{\mathrm{cub}}(t)$ from Equation (11.54) into Equation (11.53), the cubic interpolation in Equation (11.48) emerges. Whereas $a = -0.5$ is most commonly used, a cubic interpolation function with $a = -0.75$ closely approximates a windowed sinc function with the Hamming window and $K = 5$:

$$\varphi(t) = \begin{cases} \operatorname{sinc} t \left[\alpha + (1 - \alpha) \cos \dfrac{2\pi t}{K - 1} \right] & \text{for } |t| < \dfrac{K - 1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{11.55}$$

where $\alpha = 0.54$ defines the Hamming window and K is the support. The windowed sinc function has been proposed as an alternative to Equation (11.51) to reduce ringing artifacts. The interpolation functions in Equations (11.54) and (11.55) are illustrated in

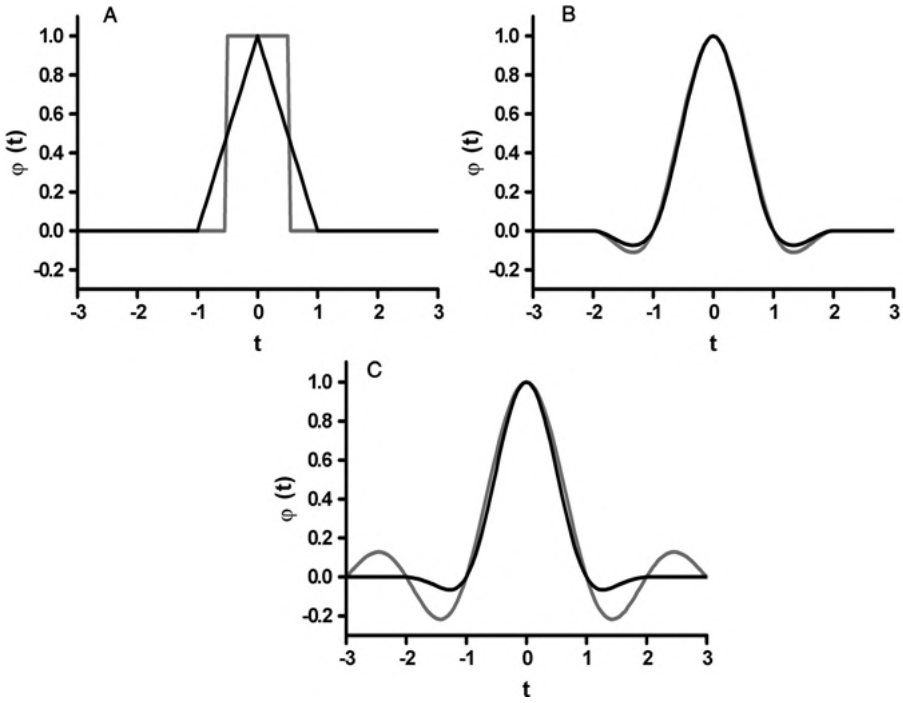


FIGURE 11.9 Some interpolation functions $\varphi(t)$ as presented in Equations (11.54) and (11.55): (A) the nearest-neighbor function (gray) and the linear interpolation function (black); (B) the cubic interpolation function for the commonly used cases $a = -0.5$ (black) and $a = -0.75$ (gray); and (C) the sinc function (gray) together with the windowed sinc function [Equation (11.55)] for $K = 5$ (black). The sinc function is the only function with infinite support.

Figure 11.9. From the figure it can be seen that the lower-order interpolation functions have undesirable frequency responses. The nearest-neighbor function has a sinc-type Fourier transform (conversely, the sinc interpolation function has an ideal boxlike frequency response). A very detailed analysis of a number of interpolation functions was performed by Thévenaz et al.[42] and the authors provided an efficient implementation in C for download on the Web at http://bigwww.epfl.ch/algorithms.html.

An alternative way to implement some transformations (i.e., the translation and rotation transformations)[16] is to use the properties of the Fourier transform (Chapter 3). The three-dimensional discrete Fourier transform $\mathcal{F}$ of an image I(x,y,z) and its inverse transform $\mathcal{F}^{-1}$ are defined by

$$\mathcal{F}\{I(x,y,z)\} = \frac{1}{XYZ}\sum_z\sum_y\sum_x I(x,y,z)\exp\left[-2\pi j\left(\frac{ux}{X}+\frac{vy}{Y}+\frac{wz}{Z}\right)\right]$$

$$\mathcal{F}^{-1}\{I(x,y,z)\} = \sum_z\sum_y\sum_x I(x,y,z)\exp\left[2\pi j\left(\frac{ux}{X}+\frac{vy}{Y}+\frac{wz}{Z}\right)\right] \qquad (11.56)$$

Although Equation (11.56) describes the discrete Fourier transform, all considerations in this section also apply to the fast Fourier transform. For this reason, implementations of rigid-body transforms in Fourier space are particularly efficient. A transformation of the type $P' = \mathbf{M}P$ applied to all XYZ pixels of an image (thus the transformation of the original image I into an image I') can be formulated as a filter F in the Fourier domain:

$$I'(x,y,z) = \mathscr{F}^{-1}\{F \cdot \mathscr{F}\{I(x,y,z)\}\} \tag{11.57}$$

The filter function F for a translation by Δx, Δy, and Δz can be determined from Equation (11.56) by substituting the translated coordinates. This is derived as follows (note that the normalization $1/XYZ$ was omitted to simplify the equation):

$$\begin{aligned}
&\mathscr{F}\{I(x', y', z')\} \\
&= \sum_z \sum_y \sum_x I(x,y,z) \exp\left[-2\pi j \left(\frac{u(x+\Delta x)}{X} + \frac{v(y+\Delta y)}{Y} + \frac{w(z+\Delta z)}{Z}\right)\right] \\
&= \sum_z \sum_y \sum_x I(x,y,z) \exp\left[-2\pi j \left(\frac{ux}{X} + \frac{vy}{Y} + \frac{wz}{Z}\right)\right] \exp\left[-2\pi j \left(\frac{u\Delta x}{X} + \frac{v\Delta y}{Y} + \frac{w\Delta z}{Z}\right)\right] \\
&= \mathscr{F}\{I(x,y,z)\} \exp\left[-2\pi j \left(\frac{u\Delta x}{X} + \frac{v\Delta y}{Y} + \frac{w\Delta z}{Z}\right)\right]
\end{aligned} \tag{11.58}$$

The resulting filter function is a frequency-dependent phase shift. A translation can therefore be implemented by a Fourier transform of the original image I, followed by a multiplication of each voxel at frequency (u,v,w) in Fourier space by the complex filter function

$$F(u, v, w) = \exp\left[-2\pi j \left(\frac{u\Delta x}{X} + \frac{v\Delta y}{Y} + \frac{w\Delta z}{Z}\right)\right] \tag{11.59}$$

followed by an inverse Fourier transform. Even with noninteger translation vectors, no interpolation is needed. The Fourier transform of an image rotated by a rotation matrix $\mathbf{R}$ is identical to the Fourier transform of the original image, rotated by $\mathbf{R}$ in the frequency domain. For this reason, implementation of a rotation filter in the Fourier domain is identical to the implementation in the spatial domain. However, the rotation operation requires interpolation in frequency space. To avoid this interpolation, the three-dimensional rotation matrix $\mathbf{R}$ can be represented by the product of four shear matrices[13] as introduced in

$$\mathbf{R} = \mathbf{H}_{y,1}\,\mathbf{H}_z\,\mathbf{H}_x\,\mathbf{H}_{y,2} \tag{11.60}$$

Since a shear operation adds a nonconstant term to the exponential expression in Equation (11.56), a straightforward filter function as in Equation (11.38) is not feasible. Instead, the shear operation corresponds to one-dimensional coordinate shifts. Each one-dimensional row can be processed independently from all others,

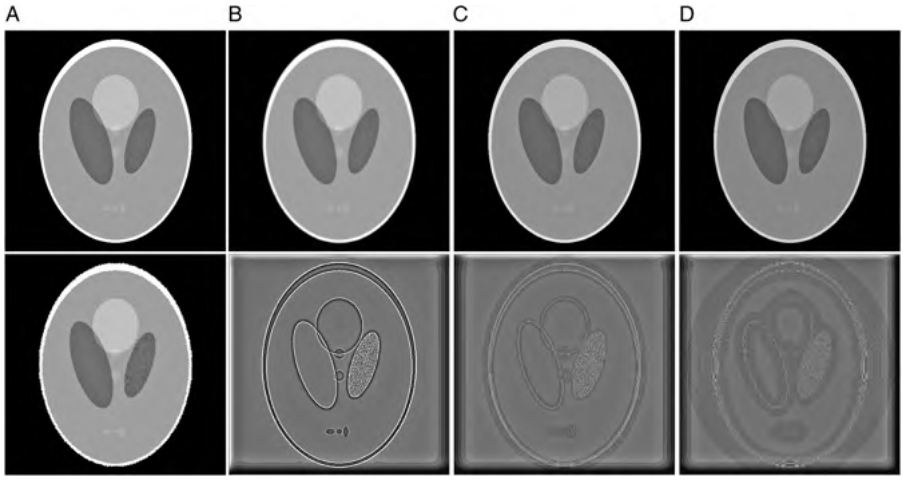


FIGURE 11.10 Comparison of the performance of various interpolation schemes. The original image (A) is a variation of the Shepp–Logan head phantom with enhanced contrast and some Gaussian noise added to a small region. Underneath is the result of repeated transformations with nearest-neighbor interpolation. Images B and C are the result of the same transformation sequence with linear and cubic interpolation, respectively. Underneath are the differences to the original image A. Image D is the result of the transformations with the interpolation proposed by Thévenaz et al.[42] and a fifth-order spline. The difference image illustrates that this interpolation scheme causes the smallest deviations from the original image.

and the shear operation can be implemented as a sequence of one-dimensional Fourier transforms in all three spatial directions, followed by application of the coordinate-dependent shift filter for each transformed line. A similarly elegant solution for the scaling operation does not exist. The scaling theorem stipulates that a dilation by a factor a corresponds to a contraction along the frequency axis by a factor of a. However, the scaling theorem is valid only for the continuous Fourier transform.

The importance of selecting a suitable interpolation method is demonstrated in Figure 11.10. In this example, a variation of the Shepp–Logan head phantom with enhanced contrast and added Gaussian noise in the right "ventricle" was subjected to multiple transformations: scaling by 0.83 followed by scaling by 1.7, followed by 15 rotations by 24°, followed by rescaling to its original size. Figure 11.10A shows the original phantom before interpolation, and the result of the transformations using nearest-neighbor interpolation underneath. Figure 11.10B and C show the result of the transformations with the linear and cubic interpolation, respectively, and the difference to the original image is shown underneath. Figure 11.10D was computed by using the interpolation scheme proposed by Thévenaz et al.[42] with a fifth-order spline. Nearest-neighbor interpolation leads to a jagged appearance, while both linear and cubic interpolation schemes cause a significant amount of blurring. The least loss of detail can be seen in the spline interpolation scheme in Figure 11.10D. However, some ringing and overshoot beyond the original image value range was

observed. Relative to the linear interpolation scheme, nearest-neighbor interpolation is 45% faster, cubic interpolation takes 44% more time, and spline interpolation takes 144% more time. The root-mean-squared coefficient of variation (root-mean-squared distance normalized by intensity average) between the transformed image and the original image is 15.7% for the nearest-neighbor interpolation, 10.2% for linear interpolation, 6.6% for cubic interpolation, and 5.1% for spline interpolation. Cubic interpolation is often considered an acceptable balance between computational effort and interpolation quality, but with more powerful computers, a shift toward more sophisticated interpolation schemes can be expected.

11.5. BIOMEDICAL EXAMPLES

Registration falls into one of these categories:

- The same subject is scanned repeatedly with the same modality (intrasubject, intramodality registration), for example, serial imaging or imaging at follow-up exams.
- The same subject is scanned with different modalities (intrasubject, intermodality registration).
- Different subjects are registered to each other or to a representative (normal or average) shape (intersubject registration).

The first type of registration can usually be covered with rigid-body transformations, particularly when the head was scanned. Fiducial markers may be helpful, but surface- or landmark-based matching is generally sufficient to achieve subpixel accuracy. The second type of registration poses its major challenge when there is a different physical basis for image intensity. Matched resolution can be achieved by scaling, and the optimum transformation is often determined with landmarks, fiducial markers, or intensity correlation. In the latter case, an intensity transform is sometimes necessary, for example, to adjust CT tissue intensity values to MR intensities. Depending on the device, the rigid-body model is often sufficient. The third type of registration is most commonly applied to head scans, and the deviation from the reference image is crucial. The key challenge in intersubject registration is to find a suitable nonlinear deformation model and to steer the registration process to locally optimal transformations. In this case, elastic deformation models or flow models are sometimes combined with local intensity correlation metrics. A multigrid-approach is often used to accelerate the computation and to achieve good registration results when large deformation is needed.

A large number of review articles aimed at the medical profession can be found. Hutton.[20] give a brief but comprehensive overview of the methods used for registration and specific applications, particularly in nuclear medicine and with a focus on the validation of the registration transformations. Little and Hawkes[26] review intrasubject registration for both the rigid-body model and nonrigid deformations with

examples of CT-MR registration. Holden[19] reviews state-of-the-art nonrigid transformation models, including elastic and fluid flow transformations, and piecewise transformations with basis functions (splines, functions with limited support, piecewise affine models, and models that are aided by the wavelet decomposition). Finally, an overview and comparative evaluation of some interpolation methods is provided by Parker et al.[31]

To study the motion of lumbar vertebrae under posture changes, McCane et al.[28] used two-dimensional rigid-body transforms to quantify vertebral motion in projection x-ray images. The vertebrae were outlined in a semimanual fashion with an operator marking landmark points along the vertebral outline, and the final shape obtained by spline interpolation through the landmark points. In this two-dimensional case, the transformation consists of a translation (to match the shape centroids) followed by a rotation. The optimum rotation angle was determined by four different methods: the iterative closest-point algorithm,[6] a method where 100 equally spaced points are extracted from the spline-interpolated contour,[14] and two related methods[9] where either ventral and dorsal corner points or ventral and dorsal midpoints of the shape were used for registration. The lowest registration error was found when the iterative closest-point algorithm was used, and registration had a mean angular error of 0.44°.

Another example for two-dimensional registration, described briefly by Jan et al.[22] registers autofluorescent and near-infrared images of the retina of the same subject. The method is landmark-based. The landmarks were bifurcations of the retinal blood vessels. To extract the landmarks, image preprocessing and segmentation were necessary with the following steps: image denoising using an anisotropic diffusion filter, correction of inhomogeneous illumination, histogram stretching for contrast enhancement, and local application of Otsu's thresholding method in a moving window to segment the blood vessels. A skeletonization process of the blood vessels revealed bifurcation points (nodes). These bifurcation points were used as landmark points, and a distance metric was used to determine corresponding nodes (as opposed to nodes that only showed up in one of the images). The iterative closest point method[6] was used to determine the rotation angle of the rigid-body transformation. This fully unsupervised method was developed to aid automated detection of several retinal diseases, primarily glaucoma and macular degeneration.

Rigid-body registration is a technique often considered sufficient for intrasubject and intermodality registration of brain images: for example, MR images and either PET or SPECT images. Viergever et al.[43] describe the two steps of spatial matching (the actual registration step) and the subsequent visualization of the registered images. Registration was based predominantly on the gray-scale co-occurrence matrix with a multiscale registration process. Examples are provided on how a SPECT image superimposed over the more detailed MR image can provide diagnostic information on patients with a tumor and with Tourette syndrome. Visualization of the registered images included both three-dimensional surface rendering and semiopaque volume rendering, the latter giving a better impression of the spatial arrangement.

Another example of three-dimensional rigid-body registration is provided by Livieratos et al.[27] to optimize PET imaging of the lung. Although respiratory gating is generally possible, the delay between matching time points of respiration is

associated with a loss of radioactive counts. To avoid the need for respiratory gating, Livieratos et al. suggested a method for motion correction. By first using gated imaging, a parametric representation of respiratory motion was obtained. The motion description was used as a basis for transformations to register the images. Validation of the method was based on tracer distribution in a phantom and with patient data.

An unsupervised algorithm for intrasubject rigid-body registration of fMRI images termed Autoalign was presented by Ciulla and Deek.[12] The authors postulated that three landmark points were sufficient to compute a rigid-body transformation. By using a feature extraction method based on the two-dimensional gradient operator, they extracted the nose and ears as the landmark points of choice. These three landmark points were used to define the head coordinate system, which was brought to congruence between images by means of three-dimensional translation and rotation. Images were rescaled by linear interpolation. Although registration accuracy was found to be comparable to other published algorithms, the strength of the Autoalign algorithm is its simplicity and straightforward implementation.

A large-scale study was performed by West et al.[45] to evaluate different methods of retrospective registration applied at different medical centers. For the comparison, the goal was intrasubject registration of CT, MR, and PET images of the head. To establish a gold standard, fiducial markers were attached to the patients, but the traces of the markers in the images were removed before the images were distributed to the participating centers. With the large number of participating centers, a large number of different algorithms were compared in this study. However, the study was restricted to rigid-body transforms. Example registration strategies included surface matching, intensity correlation, registration by means of virtual forces computed through a potential energy field, Pellizari's head-and-hat method,[33] and chamfer matching. The registration quality was evaluated using a metric of total registration error (TRE) applied to anatomically relevant regions. Whereas there were no major differences in the median TRE, the maximum TRE did vary substantially between different centers. This study demonstrated that automated registration is a feasible technique, but manual supervision is advisable to guard against failures of the registration algorithm.

An elastic deformation-based method to register abdominal and thoracic CT images with PET images was presented by Shehkar et al.[39] One interesting aspect of this study is the formulation of the transformations using quaternions.[44] Quaternions are an extension of complex numbers with one scalar (real-valued) and three orthogonal imaginary components. Quaternions allow a particularly elegant formulation of rotation operations. The registration algorithm was based on a coarse registration through rigid-body transformations, followed by a hierarchical octree-based volume subdivision. Subvolumes on all levels were also registered using the rigid-body model with the transformation limited on each subvolume. The registration strategy used in this study was the maximization of mutual information (i.e., entropy), a technique based on voxel intensities. Finally, a global nonrigid mapping scheme was computed from the voxel-level registration transformations by means of tricubic interpolation of the voxel centers. The registration scheme was evaluated by expert radiologists. Registration error was found to be comparable to the interexpert registration error.

A particularly challenging registration task is the intrasubject registration of x-ray mammograms. Although this registration task is a two-dimensional problem, the inhomogeneous tissue composition allows for highly nonlinear and inhomogeneous deformation. Since radiologists would use texture similarities to compare mammograms visually, Sallam and Bowyer[37] propose registration based on outline extraction, removal of the pectoral muscle from the image, and a curve-matching algorithm based on the assumption of local linear stretching. After a global linear matching step, homologous points on the surface were determined by their curvature, and stretching was calculated from the displacement. Surface elasticity constraints could now be imposed on inner homologous feature points that were matched to determine a final warping field. Registration accuracy was determined by calculating the co-occurrence matrix.

An important, although less obvious area where image registration plays a key role is image-guided radiotherapy, in which the goal is to align the irradiation beams with the target area to optimize irradiation of the treatment area (tumor) and minimize irradiation of the surrounding tissues. Yue et al.[47] presented a method to apply rigid-body transformation techniques to move the therapy device in a manner to compensate for variations in the position of the target. A comprehensive overview of registration and data fusion techniques for radiation therapy, including diagnostic imaging, treatment planning, and treatment delivery, is provided by Kessler[24] and by Balter and Kessler.[3]

Software for image registration, ready for immediate use, exists. A number of software options are presented and discussed by Hutton and Braun.[21] Of particular interest is the free and open-source *Insight Toolkit*, (ITK), available at http://www.itk.org. This project is sponsored primarily by the National Institutes of Health and the National Science Foundation. ITK contains a large number of algorithms for image registration, bias-field correction, filtering, and several algorithms for segmentation. ITK is an excellent toolkit for anyone who needs to get image registration tasks done without having the resources for major software development.

REFERENCES

1. Arun KS, Huang TS, Blostein SD. Least-squares fitting of two 3-D point sets. IEEE Trans Pattern Anal Mach Intell 1987; 9(5):698–700.
2. Bajcsy R, Lieberson R, Reivich M. A computerized system for the elastic matching of deformed radiographic images to idealized atlas images. J Comput Assist Tomogr 1983; 7(4):618–625.
3. Balter JM, Kessler ML. Imaging and alignment for image-guided radiation therapy. J Clin Oncol 2007; 25(8):931–937.
4. Barrow HG, Tenenbaum JM, Bolles RC, Wolf HC. Parametric correspondence and chamfer matching: two new techniques for image matching. Proc 5th Int Conf Artif Intell 1977; 659–663.
5. Bernon JL, Boudousq V, Rohmer JF, Fourcade M, Zanca M, Rossi M, Mariano-Goulart D. A comparative study of Powell's and downhill simplex algorithms for a fast multimodal surface matching in brain imaging. Comput Med Imaging Graph 2001; 25(4):287–297.

6. Besl P, McKay N. A method for registration of 2d shapes. IEEE Trans Pattern Anal Mach Intell 1992; 14:239–256.
7. Bocher M, Balan A, Krausz Y, Shrem Y, Lonn A, Wilk M, Chisin R. Gamma camera-mounted anatomical x-ray tomography: technology, system characteristics and first images. Eur J Nucl Med 2000; 27(6):619–627.
8. Borgefors G. Distance transformations in arbitrary dimensions. Comput Vis, Graph, Image Process 1984; 27:321–345.
9. Brinckmann P, Frobin W, Biggemann M, Hilweg D, Seidel S, Burton K, Tillotson M, Sandover J, Atha J, Quinell R. Quantification of overload injuries of the thoracolumbar spine in persons exposed to heavy physical exertions or vibration at the workplace: I. The shape of vertebrae and intervertebral discs—study of a young, healthy population and a middle-aged control group. Clin Biomech 1994; Suppl 1:S5–S83.
10. Bro-Nielsen M, Gramkow C. Fast fluid registration of medical images. In: Höhne KH, Kikinis R, editors. Visualization in Biomedical Computing. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 1996:267–276.
11. Christensen GE, Rabbitt RD, Miller MI. Deformable templates using large deformation kinematics. IEEE Trans Image Process 1996; 5(10):1435–1447.
12. Ciulla C, Deek FP. Performance assessment of an algorithm for the alignment of fMRI time series. Brain Topogr 2002; 14(4):313–332.
13. Cox RW, Jesmanowicz A. Real-time 3D image registration for functional MRI. Magn Reson Med 1999; 42(6):1014–1018.
14. Crisco JJ, Hentel K, Wolfe S, Duncan J. Two-dimensional rigid-body kinematics using image contour registration. J Biomech 1995; 28(1):119–124.
15. Dhawan AP, Arata LK, Levy AV, Mantil J. Iterative principal axes registration method for analysis of MR-PET brain images. IEEE Trans Biomed Eng 1995; 42(11):1079–1087.
16. Eddy WF, Fitzgerald M, Noll DC. Improved image registration by using Fourier interpolation. Magn Reson Med 1996; 36(6):923–931.
17. Gee JC, Haynor DH, Reivich M, Bajcsy R. Finite element approach to warping of brain images. Proc SPIE 1994; 2167(327):327–337.
18. Golub GH, van Loan CF. Matrix Computations, 3rd ed. Baltimore: Johns Hopkins University Press, 1996.
19. Holden M. A review of geometric transformations for nonrigid body registration. IEEE Trans Med Imaging 2008; 27(1):111–128.
20. Hutton BF. Image registration: an essential tool for nuclear medicine. Eur J Nucl Med Mol Imaging 2002; 29(4):559–577.
21. Hutton BF, Braun M. Software for image registration: algorithms, accuracy, efficacy. Semin Nucl Med 2003; 33(3):180–192.
22. Jan J, Kubecka L, Kolar R, Chrastek R. Analysis of fused ophthalmologic image data. Systems, Signals and Image Processing, 2007, and 6th EURASIP Conference Focused on Speech and Image Processing, Multimedia Communications and Services 14th International Workshop, 2007; 33–36.
23. Jiang H, Robb RA, Tainter KSH. New approach to 3-D registration of multimodality medical images by surface matching. Robb RA, editor. Visualization in Biomedical Computing '92, Chapel Hill, NC, Sept. 22, 1992; 1808(1):196–213.
24. Kessler ML. Image registration and data fusion in radiation therapy. Br J Radiol 2006; 79(Spec Issue 1):S99–S108.

25. Levin DN, Pelizzari CA, Chen GT, Chen CT, Cooper MD. Retrospective geometric correlation of MR, CT, and PET images. Radiology 1988; 169(3):817–823.
26. Little JA, Hawkes DJ. The registration of multiple medical images acquired from a single subject: why, how, what next? Stat Methods Med Res 1997; 6(3):239–265.
27. Livieratos L, Stegger L, Bloomfield PM, Schafers K, Bailey DL, Camici PG. Rigid-body transformation of list-mode projection data for respiratory motion correction in cardiac PET. Phys Med Biol 2005; 50(14):3313–3322.
28. McCane B, King TI, Abbott JH. Calculating the 2D motion of lumbar vertebrae using splines. J Biomech 2006; 39(14):2703–2708.
29. Modersitzki J. Numerical Methods for Image Registration. Oxford, UK: Oxford University Press, 2004.
30. Monga O, Benayoun S. Using partial derivatives of 3D images to extract typical surface features. Comput Vis Image Underst 1995; 61(2):171–189.
31. Parker JA, Kenyon RV, Troxel DE. Comparison of interpolating methods for image resampling. IEEE Trans Med Imaging 1983; 2(1):31–39.
32. Patton JA, Delbeke D, Sandler MP. Image fusion using an integrated, dual-head coincidence camera with x-ray tube-based attenuation maps. J Nucl Med 2000; 41(8):1364–1368.
33. Pelizzari CA, Chen GTY, Spelbring DR, Weichselbaum RR, Chen CT. Accurate three-dimensional registration of CT, PET, and/or MR images of the brain. J Comput Assist Tomogr 1989; 13(1):20–26.
34. Powell MJD. An efficient method for finding the minimum of a function of several variables without calculating derivatives. Comput J 1964; 7(2):155–162.
35. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C, 2nd ed. Cambridge, UK: Cambridge University Press, 2002.
36. Rueckert D, Sonoda LI, Hayes C, Hill DL, Leach MO, Hawkes DJ. Nonrigid registration using free-form deformations: application to breast MR images. IEEE Trans Med Imaging 1999; 18(8):712–721.
37. Sallam MY, Bowyer KW. Registration and difference analysis of corresponding mammogram images. Med Image Anal 1999; 3(2):103–118.
38. Schönemann PH, Carroll RM. Fitting one matrix to another under choice of a central dilation and a rigid motion. Psychometrika 1970; 35(2):245–255.
39. Shekhar R, Walimbe V, Raja S, Zagrodsky V, Kanvinde M, Wu G, Bybel B. Automated 3-dimensional elastic registration of whole-body PET and CT from separate or combined scanners. J Nucl Med 2005; 46(9):1488–1496.
40. Siddon RL. Prism representation: a 3D ray-tracing algorithm for radiotherapy applications. Phys Med Biol 1985; 30(8):817–824.
41. Therion J-P. New feature points based on geometric invariants for 3D image registration. Int J Comput Vis 1996; 18(2):121–137.
42. Thévenaz P, Blu T, Unser M. Interpolation revisited. IEEE Trans Med Imaging 2000; 19(7):739–758.
43. Viergever MA, Maintz JBA, Niessen WJ, Noordmans HJ, Pluim JPW, Stokking R, Vincken KL. Registration, segmentation, and visualization of multimodal brain images. Comput Med Imaging Graph 2001; 25(2):147–151.
44. Walimbe V, Zagrodsky V, Raja S, Bybel B, Kanvinde M, Shekhar R. Elastic registration of three-dimensional whole-body CT and PET images by quaternion-based interpolation of multiple piecewise linear rigid-body registrations. Proc SPIE 2004; 5370:1191–1228.

45. West J, Fitzpatrick JM, Wang MY, Dawant BM, Maurer CR Jr, Kessler RM, Maciunas RJ, Barillot C, Lemoine D, Collignon A, Maes F, Suetens P, Vandermeulen D, van den Elsen PA, Napel S, Sumanaweera TS, Harkness B, Hemler PF, Hill DLG, Hawkes DJ, Studholme C, Maintz JBA, Viergever MA, Malandain G, Pennec X, Noz ME, Maguire GQ Jr, Pollack M, Pelizzari CA, Robb RA, Hanson D, Woods RP. Comparison and evaluation of retrospective intermodality brain image registration techniques. J Comput Assist Tomogr 1997; 21(4):554–568.
46. Yaroslavsky LP. Signal sinc-interpolation: a fast computer algorithm. Bioimaging 1996; 4:225–231.
47. Yue NJ, Knisely JP, Song H, Nath R. A method to implement full six-degree target shift corrections for rigid body in image-guided radiotherapy. Med Phys 2006; 33(1):21–31.

12

IMAGE STORAGE, TRANSPORT, AND COMPRESSION

Images require a sizable amount of memory. A 12-bit computed tomography slice of 512 × 512 pixels requires 384 kilobytes (kB) of uncompressed storage. A three-dimensional CT volume of 30 such slices requires about 10 megabytes (MB) of memory. By today's standards, this does not appear large. However, the Health Insurance Portability and Accountability Act of 1996 (HIPAA) stipulates retention of medical images for 10 years with the exception of mammograms (five years) and pediatric records (21 years). Estimates are wide-ranging, but a typical clinical center accumulates about 20 gigabytes (GB) per year for each hospital bed. With easier access to medical imaging devices, higher resolution of medical images, new modalities, and an increasing number of imaging procedures that are performed, no steady state can be expected. Rather, the volume of medical images that need storage grows by estimated 10 to 20% per year. A 10% annual increase in medical image volume corresponds to doubling the volume every seven years. One large health plan with about 377,000 patients was examined,[25] and in a 10-year period, those patients underwent almost 5 million diagnostic tests. In the period from 1997 to 2006, imaging with computed tomography (CT) doubled, from 81 examinations to 181 examinations, and imaging with magnetic resonance imaging (MRI) tripled, from 22 examinations to 72 examinations per 1000 patients. Digital imaging modalities, such as CT and MRI, are not the only sources of digital images. Frequently, film archives are digitized for computer archiving and computer image analysis.

Clearly, image compression (the reduction of the image storage size) would reduce the burden on storage systems and on network bandwidth. There are two fundamental

Advanced Biomedical Image Analysis, By Mark A. Haidekker

types of image compression: lossless compression and lossy compression. *Lossless compression* is a compression scheme that allows the exact reconstruction of the original data from compressed data. Typically, a lossless compression scheme can achieve up to 50% reduction of the storage size, but the compression rates are much lower in images with a large noise component. Conversely, *lossy compression* schemes do not allow exact reconstruction of the original data. However, lossy compression rates by far exceed those of lossless compression. When choosing a lossy compression scheme, it is therefore crucial to find an acceptable balance between the compression rate and the restored image quality. Similarly, film digitization requires finding a balance between digitized resolution (both spatial resolution and the number of gray levels) and image size.

The human eye is fairly insensitive to gray levels.[28] About 20 different levels of intensity can be distinguished in a small image region. Because the eye is able to adapt to different levels of brightness, about 100 levels of intensity are reasonable in a digital image. While 100 levels of intensity can be represented in 7 bits, the most common choice is a resolution of 8 bits/pixel or 256 possible levels of intensity. The number of discrete intensity levels is often called the *depth* of the image. Figure 12.1 demonstrates the effect of low image depths.

Although a depth of 8 bits is typical for MRI, ultrasound, and many instances of digital photography, CT and digital x-ray images are usually stored with a higher depth. Typically, a CT image is reconstructed at 12 bits/pixel, allowing us to distinguish 4096 different CT values. Most CT scanners provide image values calibrated by x-ray attenuation of the tissue, given in Hounsfield units (HU). Hounsfield units are x-ray density values normalized by the density of water:

$$I_{\mathrm{CT}}(x,y) = \frac{\mu(x,y) - \mu_{\mathrm{water}}}{\mu_{\mathrm{water}}} \cdot 1000\ \mathrm{HU} \tag{12.1}$$

where $\mu(x,y)$ is the tissue absorption coefficient at a spatial location x,y and $I_{\mathrm{CT}}(x,y)$ is the CT image value in Hounsfield units. Under this definition, water has a CT value of 0 HU and air a value of −1000 HU. With 12 bits, it is possible to cover the range from −1000 HU (air) to approximately 3000 HU (compact bone) in discrete steps of 1 HU. Although the intensity differences of different types of soft tissue are relatively small, this image depth allows reliable differentiation between tissues. Many digital x-ray scanners also provide 12-bit depth. When x-ray films are digitized, depths from 8 to 14 bits are typically used. X-ray mammography is one of the most critical applications, and depths from 12 to 14 bits are common.[10] Technically, even higher bit depths are possible but would require acquisition elements with extremely low noise (almost 100 dB for 16 bits). A higher depth allows the digital image to contain more detail. Although the eye cannot distinguish subtle intensity differences, it is possible to reveal them by applying contrast/brightness windows or lookup tables in a digital imaging workstation.[13]

To display a CT image on a typical workstation, however, reduction of the image depth to 8 bits/pixel (value range from 0 to 255) is necessary, and this reduction is associated with a loss of detail. If the image value range is linearly compressed, gray

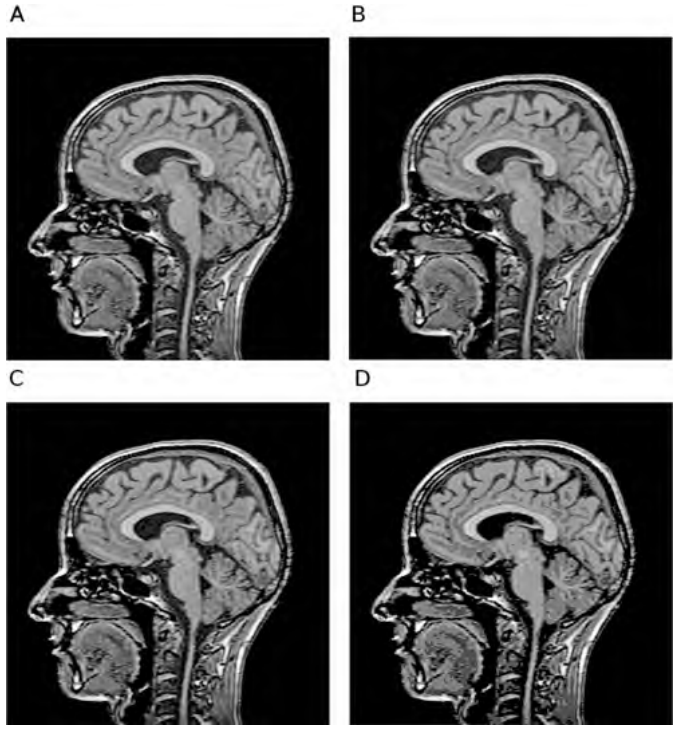


FIGURE 12.1 Relevance of the image depth (number of bits per pixel). The original T_1-weighted MR image of a head is represented with 8 bits (A), allowing a total of 256 shades of gray. Image B is represented by 6 bits/pixel (64 shades), and almost no difference to A can be seen. Image C is reduced to 4 bits/pixel and image D to 3 bits/pixel (16 and 8 shades of gray, respectively). In both images, false contouring and flat areas without contrast become obvious.

shades of less than 14 HU difference cannot be distinguished. It is possible, however, to map a limited CT value range onto the 8-bit display range, and an example is given in Figure 13.1. By using suitable mapping functions (referred to as *window/center functions*), a contrast window can be devised that maps the interior of the lung (-900 to -200 HU) to the value range from 0 to 255, resulting in higher contrast in the lung but saturation of the soft tissue and bone regions. In the same example, another contrast window is applied that maps soft tissue values (-200 to 200 HU) to the value range from 0 to 255. This contrast allows good differentiation between different types of soft tissue (e.g., the darker adipose layer and the lighter muscle tissue), but no contrast is seen inside the lung and in the bone areas. Provided that the CT image is stored at its full depth, the window/center mapping functions can be applied when the image is viewed.

The spatial resolution of medical images varies greatly, particularly between modalities. In some modalities (i.e., CT, PET, and SPECT), the in-plane resolution is limited by the source-detector system. Typical CT pixel resolutions are between 0.1

and 2 mm/pixel, and PET and SPECT devices have pixel sizes of about one order of magnitude larger. Conversely, MRI and ultrasound imaging features variable pixel sizes. The limiting factor in MR scanners is the ability of the signal acquisition system to distinguish between different frequency components in a narrow frequency band. Since it is relatively easy to manipulate the magnetic field gradients, the resulting field of view is highly variable, and with a fixed reconstruction matrix, so is the final pixel size. Ultrasound resolution is limited by ultrasound burst frequency and bandwidth in the axial direction and by the probe focus in the lateral direction. Ultrasound scanners allow easy manipulation of the scan angle and the scan depth, and the final pixel size is highly variable and likely to be anisotropic. The pixel sizes may even differ within the same image. Furthermore, different ultrasound transducers feature different resolutions.

In the three-dimensional imaging modalities (primarily CT, MRI, SPECT, and PET), the slice thickness varies greatly because it can easily be manipulated. Generally, the slice thickness is larger than the in-plane resolution, and three-dimensional image voxels are anisotropic in the axial direction. Slice thickness varies from 1–2 mm (CT) to 5 mm (MRT) to several centimeters (PET and SPECT), to provide typical examples.

In most cases, high-resolution instruments exist that acquire images at a much higher spatial resolution than that of standard clinical scanners. Micro-CT devices with a 5- to 50-μm in-plane pixel size exist, and micro-MRI devices can provide in-plane pixel sizes of 100 μm or better. However, high resolution often comes at the cost of a reduced field of view or a small device bore, and these specialized instruments are restricted to ex vivo samples or small animals.

12.1. IMAGE ARCHIVING, DICOM, AND PACS

The digital imaging and communications in medicine (DICOM) standard describes the medical image format, which allows distributing and viewing any type of medical image, irrespective of its origin, modality, or analysis software. PACS (picture archiving and communication system) refers to any computer system or network that allows storage, archiving, retrieval, and transfer of medical images. PACS normally (but not necessarily) uses the DICOM format to store images, to ensure compatibility with other PACS systems and with possible PACS upgrades and replacements.

The main purpose of PACS is to integrate the various components that contribute or analyze medical images, to integrate ancillary components (such as hospital information systems), and to allow the transfer of images within and between medical centers. At the center of PACS is the storage and archive server, which not only archives the images, but also runs a database for image retrieval. Connected image modalities such as CT scanners or digital x-ray stations store images directly on the archive server. Radiology workstations allow image retrieval, processing, and visualization. A typical PACS configuration is shown in Figure 12.2.

The installation of PACS in a medical center is a major investment. Several centers reported their experience with the planning and execution stage (see, e.g., references 4,

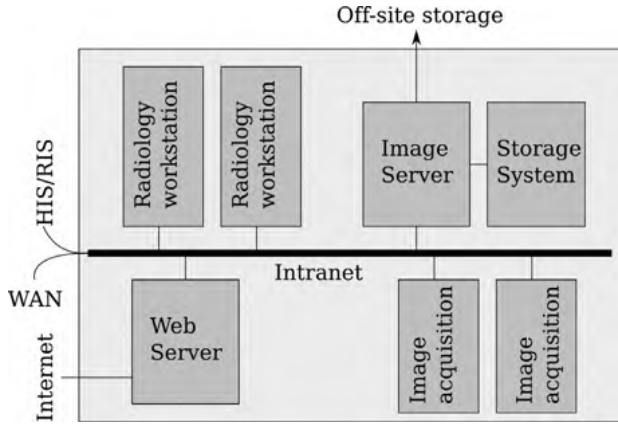


FIGURE 12.2 Schematic of a typical PACS configuration. The central component is the image server, which allows image storage and retrieval on a fast local disk system (short-term storage), a larger, but slower long-term storage unit (e.g., tape storage), and generally also off-site storage for backup purposes. Image acquisition stations (CT, MRI, digital radiography) can store images directly on the server through the intranet. Radiology workstations are available to retrieve, process, and view images. Patient information can be obtained through the hospital or radiology information system (HIS/RIS), and communication with other centers is possible through a wide-area network (WAN). More and more often, intercenter communication takes place over the Internet, which requires a Web server.

5, 8, and 19). Smaller centers have reported the implementation of small-scale low-cost PACS replacement systems.[11,15] In fact, setting up a fault-tolerant server system based on personal computer hardware is an attractive option for small centers and study groups that do not have the revenue to justify a full-scale PACS. Hard-drive capacity is rising faster than image volume, and fault-tolerant systems known as *redundant arrays of inexpensive disks* (RAIDs) with terabyte capacity are available at surprisingly low cost. Since the storage server is the most critical element, a double level of redundancy (i.e., redundant servers with redundant disks) is an attractive proposition to increase fault tolerance and decrease downtime due to component failure.[15] However, any custom solution would have to include precautions not only against data loss but also against unauthorized access.

A prerequisite for the successful deployment of PACS is the availability of an image format that can be read universally. The DICOM image format is such a format. Contrary to most other image formats, DICOM allows us to store extensive metadata together with the image data. In some respects, the principle behind the DICOM format is similar to the tagged image file format (TIFF). The TIFF format specifies a header that contains image metadata, such as image size, data type, representation, and color model. The definition of TIFF tags is extensive, which is the main reason why TIFF is one of the most flexible image formats available. DICOM follows a similar principle, but the DICOM metadata block contains information not only about the image itself but also about the patient, exam, modality, acquisition parameters,

physician, and health center. The DICOM metadata is stored in the DICOM header, a block of binary data that precedes the actual image data. The metadata are organized as a sequence of information objects (tags), each of which is preceded by a unique identifier (UID). The unique identifier consists of two 16-bit words that specify the group and the element of the metadata. Following the unique identifier is a 2-byte value representation (VR) that defines the data type of the metadata field, followed by 2 bytes that define the length of the metadata field. While the complete list of object definitions is extremely comprehensive (the 2008 DICOM object definition list is almost 1100 pages long), only a few objects need to be retrieved to extract the image data. A single-file DICOM image begins with 128 empty bytes (usually zero-value bytes), followed by four ASCII characters: D, I, C, M. Any software that is capable of reading DICOM images would therefore read the first 132 bytes of the file and verify the presence of the string "DICM" in the 129th through 132rd byte. Starting with the 133rd byte, metadata information is stored.

The first tags in the DICOM header normally pertain to the image itself. Such data are contained in Group 0002. For example, UID 0002:0000 (Group 0002, Element 0000) is a field called MetaElementGroupLength. This field has the value representation UL (unsigned long, that is, an integer value of 32 bits depth that cannot become negative), and the subsequent bytes are a binary number to represent the data (i.e., the value of MetaElementGroupLength). Important groups are 0008 (information about the modality), 0010 (information about the patient), 0018 (information about the imaging instrument settings), and 0028 (information about the image format). An example for a DICOM header is given in Table 12.1.

The transfer syntax UID contains coded information about the metadata format and the image format. The string in Table 12.1, "1.2.840.10008.1.2," indicates uncompressed data with high byte–first order (Intel order or "little-endian") and implicit value representation. The byte order is microprocessor-specific. Microprocessors of the Intel series store integer values that need more than one byte such that the most significant byte is stored first and the least significant byte last. Motorola-series microprocessors store multibyte values in reverse order ("big-endian"). A special tag, Implicit VR; tells the DICOM reader software that the value representation (e.g., UL, CS, or US, as seen in Table 12.1) is not part of the metadata, and the DICOM reader must use an internal VR table. A transfer string UID of "1.2.840.10008.1.2.*x*" indicates explicit VR; that is, the VR is provided in the 2 bytes after the unique identifier. In this case, *x* may either be 1 (little-endian) or 2 (big-endian). A transfer string UID of "1.2.840.10008.1.2.4.*xx*" indicates compressed data, namely JPEG compression (lossy or lossless, depending on the value of *xx*), and a transfer string UID of "1.2.840.10008.1.2.5" indicates compression with the lossless run-length encoding compression scheme.

A DICOM reader needs to be capable of reading all relevant tags and interpreting the raw image data accordingly. DICOM allows the storage of color data in a variety of different color models, including RGB (red–green–blue), HSV (hue–saturation–value), CMYK (cyan–magenta–yellow–black), or even palette colors, where each image value is an entry into a color table. Color displays use the RGB format, and it is necessary to identify the DICOM color model and convert it into

TABLE 12.1 Sample DICOM Metadata Header

Unique Identifier Group:Element	Value Representation	Tag Data	Content	Comment
0002:0000	UL (unsigned long)	MetaElement GroupLength	0x00000084 (132)	Length of the DICOM preamble
0002:0001	OB (other byte)	FileMeta InfoVersion		
⋮				
0002:0010	UI (unsigned integer)	TransferSyntax UID	1.2.840.10008.1.2	Description of data format and compression
0010:0010	PN (person name)	Patient Name	"Anonymous"	ASCII string of the name
⋮				
0028:0002	US (unsigned short)	Samples per pixel	1	Usually 1 for monochrome or 3 for color
0028:0004	CS (code string)	Photometric interpretation	"MONO-CHROME1"	Interpretation of pixel value in terms of brightness
0028:0008	IS (integer string)	Number of frames	16	Tells the number of different frames (images) in file
0028:0010	US	Rows	256	Image height in pixels
0028:0011	US	Columns	256	Image width in pixels

RGB. Furthermore, a DICOM image can also store the brightness and contrast settings (window width and center with the unique identifiers 0028:1050 and 0028:1051, respectively), in which case a gray-value translation similar to Figure 13.1 needs to be applied.

Despite the complexity of the DICOM format, a number of free DICOM-capable image viewers, image converters, and DICOM software libraries are available. A selection of these are listed in Table 12.2.

12.2. LOSSLESS IMAGE COMPRESSION

A lossless data compression algorithm converts the original data into a transformed data stream, which generally (but not necessarily) occupies less memory. An inverse transform (decompression) is available that exactly restores the original data. Therefore, the compression—decompression process is called *lossless*. A fundamental concept of the compression transform is to create a code with a variable word length. The data values with the highest probability are assigned to the shortest codes. As a

TABLE 12.2 Selection of DICOM-Capable Software

Software Package	URL	Comment[a]
dicom2	http://www.barre.nom.fr/medical/dicom2/	Command-line program to convert DICOM to various other formats (L, W).
ezDICOM	http://www.sph.sc.edu/comd/rorden/ezdicom.html	Offers conversion of several manufacturer-proprietary image formats (L, W, S).
MRIcron	http://www.sph.sc.edu/comd/rorden/mricron/index.html	Image analysis software with a focus on fMRI (L, W, M, S).
ImageMagick	http://www.imagemagick.org/	Comprehensive suite of image manipulation and processing programs (L, W, M, S).
AMIDE	http://amide.sourceforge.net/	Powerful tool for viewing, analyzing, and registering three-dimensional image data (L, S).
Medcon/ XMedcon	http://xmedcon.sourceforge.net/	Image conversion tool; Medcon is a command-line tool and XMedcon has a graphical interface (L, W, S).
ImageJ	http://rsb.info.nih.gov/ij/	Popular comprehensive image analysis program (see Chapter 14) (L, W, M, S).
dicom2pgm	http://sourceforge.net/projects/cdmedicpacsweb/files/	Source code for a simple and straightforward converter from DICOM to PGM; a good starting point for custom software (S).
Imebra C++ DICOM library	http://puntoexe.com/content/view/11/2/	DICOM3-compliant library for custom software projects (L, W, M, S).
Offis dcmtk	http://dicom.offis.de/dcmtk.php.en	DCMTK is a collection of C/C++ libraries and applications implementing large parts of the DICOM standard (L, W, M, S).

[a]The letter symbols indicate the availability of Linux binaries (L), Windows binaries (W), Mac OS/X binaries (M), and availability of the source code (S).

consequence, the entropy of the compressed data stream is increased. One of the first widely adopted compression schemes was the Huffman coding scheme.

Consider Figure 12.1D, an image with eight discrete shades of gray. These shades of gray or possible image values are called *symbols* in the terminology of signal processing. A histogram reveals the probabilities of each gray level (Table 12.3). Normally, 3 bits are required to represent the eight different image values. To generate a Huffman code with variable-length symbol representation, the symbols need to be arranged in a binary tree where the nodes at each level have approximately balanced probabilities. Such a tree is displayed in Figure 12.3. The codes are generated by

TABLE 12.3 Eight-Level Huffman Coding of the Image in Figure 12.1D

Gray Value	Probability (%)	Binary Code	Huffman Code
0	59.1	000	0
1	11.7	001	100
2	13.7	010	101
3	11.9	011	110
4	2.3	100	11100
5	0.7	101	11101
6	0.3	110	11110
7	0.3	111	11111

traversing the tree from the root node to each symbol leaf, adding one bit at each link. The resulting code is listed in the last column of Table 12.3.

The entropy E of the image in Figure 12.1D can be computed as

$$E = -\sum_{s=0}^{N-1} P(s)\log_2[P(s)] \tag{12.2}$$

and was determined to be $E = 1.79$. The entropy provides the theoretical minimum length of any code. In this example, the average symbol length cannot be lower than 1.79 bits per symbol. The code in Table 12.3 has an average symbol length of 1.89 bits. Compared with the uncompressed 3-bit representation, a compression ratio of 37% was achieved. It should be noted that the tree example in Figure 12.3 is not optimal, and Huffman's algorithm would create a more unbalanced tree with marginally (0.05%) lower average symbol length. Based on the entropy of the data,

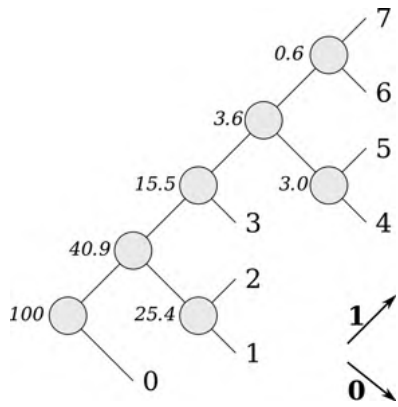


FIGURE 12.3 Binary tree to generate a Huffman code. The symbols form the leaves of the tree, and each node level is created to have approximately the same probability at each level (the probabilities of each node are shown in italic). The code is generated by walking along the tree from the leftmost node to the leaf, adding a bit value **0** or **1** to the code, depending on the direction (arrows). The binary code for the symbol "3" is therefore 1–1–0.

a more efficient code is theoretically possible, but not with this compression scheme. A Huffman tree can be generated with the following steps:

Step 1. Create one node (leaf) for each symbol and assign the symbol's probability to the node. Put all nodes into a pool (queue).

Step 2. While there is more than one node in the queue, perform steps 3 and 4:

Step 3. Remove from the queue the two nodes with the lowest probabilities.

Step 4. Link the two nodes (child nodes) to a new node and assign the sum probability of the two child nodes to the new node. Place the new node into the queue.

Step 5. The remaining single node is the root node.

Table 12.3 now contains a code with variable bit length, and the image can be stored as a sequence of the bit codes in the last column of Table 12.3. Instead of 512 kB, the image would now require 322 kB of storage space, plus storage of the code table. Since is it not easy to work with compressed data, the image needs to be decoded (i.e., decompressed) before use. For this purpose, an algorithm would analyze the bit sequence of the coded image and replace each coded string of bits with the binary number (third column of Table 12.3). After decompression, the image is restored exactly.

When a few symbols are very dominant and occur with a high probability, the Huffman code has a good compression ratio. Conversely, when the image values are distributed almost equally (high entropy), the compression ratio is low. This is a general rule that applies to all lossless compression schemes. In the worst case, it is even possible that a coded image with a low compression ratio takes up more space than the original image, because the decoding instructions (i.e., the code table) have to be stored with the coded image. In the context of this chapter, compression rates C are given as percentages according to

$$C = \frac{S_U - S_C}{S_U} \cdot 100\% \tag{12.3}$$

where S_C is the size of the compressed image and S_U is the size of the uncompressed image. Compression ratios in the form $1 : n$ relate to C through $(1 - 1/n) \cdot 100\%$. The Huffman coding example above corresponds to 37% compression. When an image is compressed to 10% of its original size, often referred to as a $1 : 10$ compression ratio, the compression according to Equation (12.3) is 90%. A $1 : 50$ compression ratio corresponds to 98% compression.

A different compression scheme is run-length encoding. The run-length method was introduced as a method to characterize texture in Section 8.4. In the example of an image with 8-bit depth, a run can be defined by two bytes: symbol and run length. The process of generating a run-length code is demonstrated in Figure 12.4, which shows one scan line (one horizontal pixel line) from an image with four gray levels. By assigning the symbols A (white) through D (black) to the gray levels, the run-length code is D1 A5 B2 C6 A5 D3 C1 B5, indicating one black pixel followed by

FIGURE 12.4 Run-length encoding. This figure shows one scan line from an image with four gray levels: A, B, C, and D. The run-length encoded image is D1 A5 B2 C6 A5 D3 C1 B5.

five white pixels, followed by two light gray pixels, and so on. The 28 original pixels in the example are encoded in 16 bytes of run-length code, which corresponds to a compression ratio of 43%. In this example, the length of a run is limited to 256 pixels, because the length code is only one byte. If a run exceeds 256 bytes a new run is started. For example, 384 consecutive white pixels can be stored as A256 A128.

Run-length encoding is a very popular method. The compression scheme of telefax machines is based on run-length encoding with an additional Huffman coding stage. A popular variation of run-length encoding is the Packbits scheme, found in TIFF images. Packbits uses a control byte to represent runs up to 128 pixels in length. Therefore, one bit (the sign bit) allows us to toggle between two modes, *uncompressed bytes* and *run*. Assume that the control byte represents a signed value n. If $n \geq 0$, the next $n + 1$ bytes are uncompressed data. If $n < 0$, the next byte is the value of a run and needs to be repeated $|n| + 1$ times. The Packbits encoding scheme has an advantage over pure run-length encoding when many runs of a single byte exist. If neighboring pixels differ, pure run-length encoding requires 2 bytes for each pixel. Packbits needs only one control byte for a sequence of differing pixels, and the overhead is therefore generally lower than with pure run-length encoding.

In 1977, Ziv and Lempel[34] introduced a different compression algorithm, often referred to as *LZ compression* or *sliding-window compression*. LZ compression makes use of the occurrence of repeated patterns. Such repeated patterns can typically be found in plain text, where the patterns correspond to words. Instead of storing the word "pattern" here (**x**), a data pair offset : length can be stored. At the position of the (**x**) in the preceding sentence, this would be 15 : 7. This combination means the following: From the current position (the **x** mark), go back 15 characters, including spaces, and copy 7 bytes of data (namely the word "pattern" without the quotes) to replace the (**x**). The repeat occurrence of the word is now encoded in 2 bytes. The LZ family of compression algorithms is also called *dictionary-based compression*, because the algorithm builds a dictionary of patterns that already occurred in the data stream. The longer the algorithm can look into the past (i.e., the longer the sliding window), the more efficient the algorithm becomes. In 1978, Ziv and Lempel introduced an improved algorithm where the sliding window was also capable of considering future (forward) data.[35]

The combination of the Lempel–Ziv dictionary-based algorithm and Huffman encoding, known as DEFLATE, found widespread use in commonly available data compression utilities such as pkzip, gzip, and in the portable network graphics (PNG) image format. T. A. Welch improved the LZ compression algorithm further[33] and introduced the compression method known as LZW. The LZW encoding algorithm creates a grow-as-you-go dictionary, which is stored as a separate table in the encoding

stage. Initially, this dictionary is populated with binary codes for each possible symbol. In the example of Figure 12.1D, eight symbols exist in the image. One additional symbol is needed to represent the end of the data stream. Thus, nine initial symbols require 4 bits per symbol, and seven more symbols can be encoded with the same number of bits. As the encoding algorithm analyzes the data stream, longer and longer symbols are entered into the dictionary: If the last $n - 1$ symbols (excluding the current symbol) exist in the dictionary, a new word of n symbols (including the current symbol) is entered into the dictionary and assigned a new bit code. If the same n symbols occur again, they are referenced by the bit code. Returning to the example in Figure 12.4, the dictionary initially consists of the symbols A, B, C, D, and a stop symbol. When the second character is examined, the symbol D already exists, and a new symbol E = DA is entered. At the third character, since both DA and A exist, two new symbols, F = DAA and G = AA, are defined and added to the dictionary. Thus, at the fifteenth character, the sequence AAAAA is a repeat of the sequence starting at the second character and referenced with the appropriate single code from the dictionary. The number of symbols will quickly exceed the number of combinations allowed in the original bit representation of the initial dictionary. In this example, five original symbols require 3 bits and allow for three additional symbols. Once this number is exceeded, the LZW coder adds an additional bit. Therefore, the LZW-coded data stream has a variable symbol length. Since the dictionary is not transmitted with the data stream, the decoder needs to keep track of the dictionary. It builds the dictionary from the encoded stream and preferentially tests the longer bit code in ambiguous cases. In this way the decoder also keeps track of the increasing symbol length in the coded stream. One major advantage of the LZW coding scheme is the implicit nature of the dictionary. Whereas Huffman and LZ coding schemes need to transmit the symbol tables, LZW does not have this type of overhead.

The image in Figure 12.1A, which contains a moderate amount of noise, can be compressed by 44% with LZW and 45% with Packbits. For the image shown in Figure 12.1C, where much of the image noise has disappeared because of the low number of gray levels, LZW allows a compression rate of 86% while Packbits compresses by 67%. It can be seen that the LZW algorithm is particularly effective in images that have wide, homogeneous areas. This was the reason why the LZW compression was chosen by CompuServe for the graphics interchange format (GIF), one of the first widely used color graphics formats in Web pages.

The Joint Photographic Experts Group (JPEG) standard defines two lossless compression standards that are less widely known than the lossy JPEG compression standards and the coding schemes described above. The first encoding scheme, named *JPEG-LS*, is based on a compression scheme owned by Hewlett-Packard. The scheme is called *low-complexity, context-based image compression*,[31,32] or *LOCO-I*. The general data flow in the compression algorithm is shown in Figure 12.5. It can be seen that the JPEG-LS algorithm has considerable complexity. Two main compression modes are available and activated, depending on image data: (1) run-length encoding, used in flat image regions, and (2) entropy encoding, used in image regions with a nonzero gradient. A flat-region discriminator (gradient image threshold) activates one of the two encoding paths.

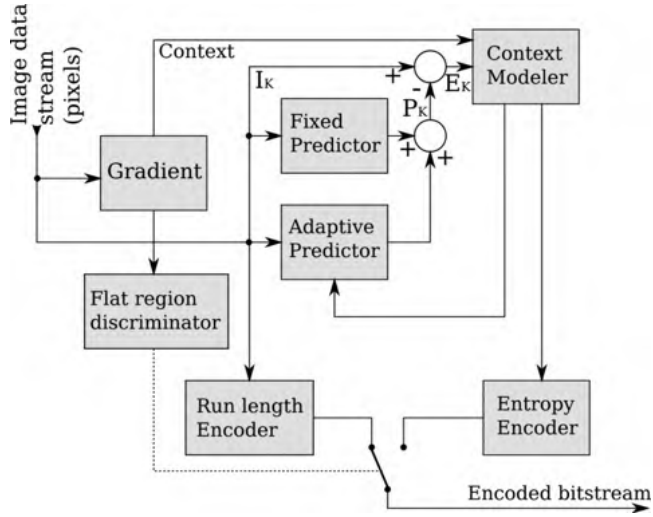


FIGURE 12.5 Schematic of the JPEG-LS compression algorithm. Two compression methods are available: (1) run-length compression for flat regions, and (2) entropy-minimizing compression in nonflat regions. A flat-region discriminator, driven by the image gradient, selects which encoding scheme is active. To improve the entropy compression rate, the image data are first subjected to a context-based model before compressed with an entropy-minimizing scheme (Golomb coding). The idea behind context-based modeling is to use a deviation from a predicted value. A predictor stage guesses an image value P_K from the neighborhood of image value I_K. The difference $E_K = I_K - P_K$ is usually a small value, and the overall entropy of the difference data stream E_K is lower than that of I_K. (From ref. 31.)

The context modeling stage deserves some explanation. Since the image is compressed by scan lines, the 8-connected neighbors to the top and to the right of a pixel are already known. The pixel configuration in the neighborhood of pixel x (corresponding to the pixel intensity value I_K in Figure 12.5) is shown in Figure 12.6. In the first modeling step, a guess $\hat{x}$ is made from the neighbors a, b, and c according to

$$\hat{x} = \begin{cases} \min(a,b) & \text{if } c \geq \max(a,b) \\ \max(a,b) & \text{if } c \leq \min(a,b) \\ a + b - c & \text{otherwise} \end{cases} \tag{12.4}$$

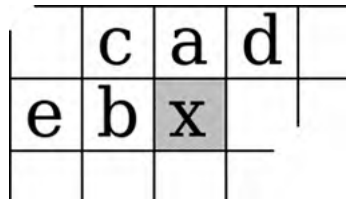


FIGURE 12.6 Pixel neighborhood configuration in the JPEG-LS algorithm.

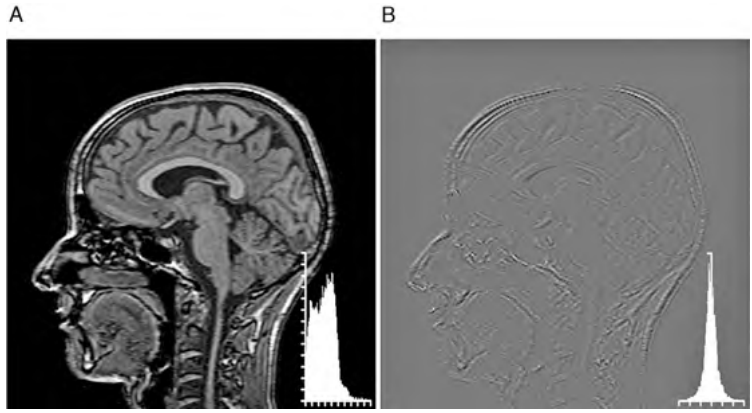


FIGURE 12.7 Prediction encoding for JPEG-LS compression. Image A shows the original MR slice, and image B contains the prediction error E_K, that is, the deviation of the predicted value from the image value I_K computed through Equation (12.4). The insets show the histogram of the respective image. The histogram of the prediction error E_K (i.e., the residuals) is zero-symmetric with an exponential decay, which allows for excellent entropy encoding.

If $\hat{x}_k$ is the predicted value, $E_k = I_k - \hat{x}_K$ is the prediction error (the residual). The transformation from I_K to E_K creates a set of pixels with an advantageous histogram, as demonstrated in Figure 12.7. The histogram of the residuals is relatively narrow, symmetric around zero, and the histogram values are decaying exponentially with the intensity value (known as *Laplacian distribution*). Such a histogram is advantageous because it allows for better entropy compression.

Because of outliers, the histogram of the residuals (see Figure 12.7B) tends to deviate from the ideal Laplacian distribution. An adaptive predictor remaps those outliers and optimizes the predicted values P_K so that the residuals E_K are optimal with respect to one of several predefined Huffman codes. Therefore, a single parameter defines the Huffman table to be used for decompression. The adaptive prediction (and therefore the modeling of the E_K to fit the Laplacian probability density function) is helped further by context modeling. The context is defined by quantized gradients $g_1 = d - b$, $g_2 = b - c$, $g_3 = c - a$ with the neighborhood notation in Figure 12.6. The actual entropy encoding makes use of the Golomb–Rice compression[21] rather than Huffman compression, because Golomb–Rice compression is optimal for exponentially decaying probability density functions.

The JPEG-2000 standard defines one additional lossless compression method. This method is based on wavelets. However, the lossless wavelet-based JPEG-2000 compression algorithm requires more computational effort than JPEG-LS and does not necessarily provide better compression rates.[23]

Several of the lossless compression algorithms are or were subject to patent restrictions. The patent for the LZW algorithm was held by the Unisys corporation, and at some point in the past, Web authors who used GIF images in Web pages were at risk of lawsuits forcing them to pay license fees. Amid considerable

controversy, Unisys decided to allow noncommercial use without a license fee, and later allowed royalty-free use of the LZW decoder. Nonetheless, these patent restrictions sparked development of the PNG format. Nonrestricted compression schemes (such as DEFLATE or Packbits) enjoyed more widespread use than restricted schemes. Although the Unisys patents expired in 2003 and 2004, thus making GIF decoders and encoders free of patent restrictions, the alternative PNG format still enjoys great popularity. The patent rights for the LOCO-I algorithm used in JPEG-LS are held by Hewlett-Packard, but licenses can be obtained free from the Hewlett-Packard Web site (http://www.hpl.hp.com/loco/).

12.3. LOSSY IMAGE COMPRESSION

Whereas lossless compression can achieve typical compression rates of 50% with many files, lossless compression rates of medical images are generally even lower because of the noise component. Lossy compression techniques lead to some image degradation. High-frequency components are dropped, and compression rates increase steeply with the amount of detail removed from the image. The idea of lossy compression is based on the observation that the human eye does not necessarily need the full spatial frequency range to discern features. As opposed to, for example, computer program code or text files, where exact restoration of the datastream is imperative, some loss of information in images is considered acceptable. Lossy compression schemes allow us to choose a balance between compression rate and decompressed image quality.

Lossy compression becomes particularly effective when image information components can be selected for which the human eye is less sensitive. In color images, the luminance (or intensity) information is perceived at much higher spatial frequencies than the chrominance (color) information. This is demonstrated in Figure 12.8, an example of the synthetic fractal landscape (Chapter 10). An adaptive anisotropic diffusion filter causes blurring in most regions except near edges. The blurring effect in Figure 12.8B is obvious. When the color information is blurred but the intensity information is not (Figure 12.8C), the information loss is much less obvious, although the difference image (Figure 12.8D) reveals that information loss takes place. Since a blurred (lowpass-filtered) image contains less noise and more homogeneous regions, entropy-maximizing compression is more effective. Instead of filtering all color components to achieve a higher compression rate, it is possible to convert the image space to (in this example) the hue-saturation-value model. In this model, the intensity information is represented by the value channel. The hue and saturation channels can be filtered readily, whereas the value channel remains unfiltered. The reconstructed image shows only small differences to the original image.

There are several basic methods of lossy data compression. All of them have in common that the image data are transformed in a manner that individual coefficients of the transformed data may be dropped with a minimum effect on the image after the inverse transform. Most commonly used is a compression standard defined by the Joint Photographic Experts Group (JPEG), which is based on the discrete cosine transform. The lossy JPEG encoding chain is sketched in Figure 12.9.

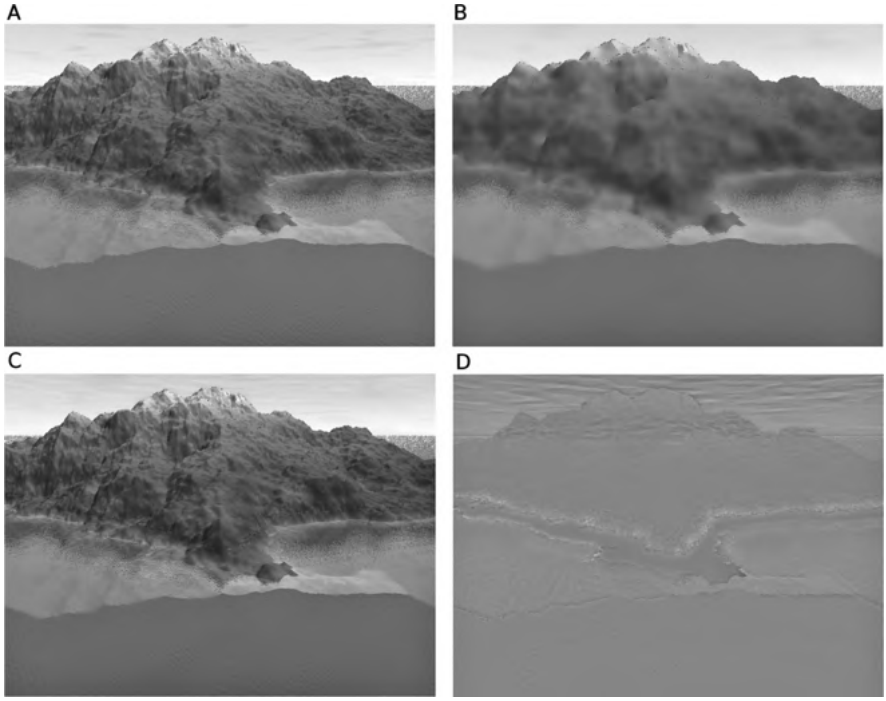


FIGURE 12.8 Demonstration of the effect of selective image information loss. Image A, subjected to an adaptive anisotropic diffusion lowpass filter, is strongly blurred (B). The loss of detail at high spatial frequencies is obvious. Conversely, when only the color (hue and saturation) information is blurred and the intensity information is left unaltered (C), almost no differences to the original image are detectable. The difference between images A and C is shown in image D. (*See insert for color representation of the figure*.)

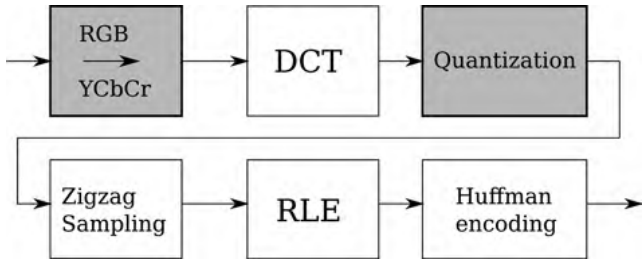


FIGURE 12.9 Schematic representation of the encoding steps for JPEG lossy compression. The shaded boxes represent the lossy part, and the white boxes represent the lossless part of the compression scheme. The uncompressed image is first converted from its original color model (such as RGB) to a luminance–chrominance model (YCbCr). Each channel is then subjected to the discrete cosine transform (DCT). In the quantization step, a specified number of DCT coefficients are set to zero. A special sampling scheme, zigzag sampling, creates long runs of zeros which readily compress in a run-length encoding (RLE) stage. Finally, Huffman encoding optimizes entropy and creates a small compressed image.

JPEG compression acts on nonoverlapping blocks of 8 × 8 pixels. The first step of JPEG compression is a conversion of the original color model to the YCbCr model followed by subsampling of the chrominance components. The luminance component (Y) remains at full resolution. The conversion of RGB to YCbCr is performed using the matrix equation

$$\begin{bmatrix} \mathrm{Y} \\ \mathrm{Cb} \\ \mathrm{Cr} \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.055 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{bmatrix} \tag{12.5}$$

Since this step is omitted in monochrome images, the initial block size is 16 × 16 pixels for compatibility. The luminance component is rounded to 8 bits and subdivided into four blocks of 8 × 8 bytes. The chrominance components Cb and Cr are first subsampled by averaging blocks of 2 × 2 neighboring pixels, then rounded to 8-bit integers. Consequently, six blocks of 8 × 8 bytes have been created from the original 16 × 16 block of three bytes. This first step corresponds to a compression of 50%.

The next step of the compression is based on the discrete cosine transform,

$$D(u,v) = \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} [I(x,y) - 128] \cos\left[\frac{\pi u}{N}\left(x + \frac{1}{2}\right)\right] \cos\left[\frac{\pi v}{N}\left(y + \frac{1}{2}\right)\right] \tag{12.6}$$

where $I(x,y)$ are the image values from one of the 8 × 8 blocks (therefore, $N = 8$), and $D(u,v)$ are the 8 × 8 cosine transform coefficients. The subtraction of the constant offset 128 is performed to reduce the image average [$D(0,0)$]. The coefficients are rounded to signed integers and may temporarily exceed the 8-bit value range. Since the cosine arguments are constant, they may be precomputed for a major speed gain, and recursive schemes [fast cosine transform (FCT) in analogy to the FFT] are used. The $D(u,v)$ represent the frequency components of each 8 × 8 block. The actual compression takes place by dividing each $D(u,v)$ by a quantization coefficient $Q(u,v)$ from an 8 × 8 quantization table:

$$Q(u,v) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{12.7}$$

Since the division $D(u,v)/Q(u,v)$ is an integer operation, many of the resulting coefficients will be zero, while the remaining coefficients are small integer numbers. This distribution of coefficients represents an ideal situation for run length and entropy encoding.

Commonly used values for the quantization table are given in Equation (12.7).[29] Since the DCT coefficients are divided by the entries of the quantization and the values increase toward higher-frequency components, the quantization process has the function of a lowpass filter. Moreover, the DCT coefficients drop quickly toward zero with higher frequencies. Therefore, the resulting quantized DCT coefficients are zero predominantly in the lower right region below a diagonal. The quantization table also determines the compression quality. For a lower quality (and therefore a higher compression rate), larger values in the quantization table are chosen. Consequently, more quantized DCT coefficients will be zero. More specifically, the quantization table in Equation (12.7) defines the quality level $q = 50$. For higher qualities, each component of the matrix can be multiplied by $(100 - q)/50$, and for lower qualities the multiplication factor is $50/q$. Furthermore, different quantization tables are used for the luminance and chrominance components, allowing a higher compression rate for the chrominance (color) information.

The final compression step is a step of run-length encoding followed by entropy encoding of the quantized DCT coefficients. To generate long runs, the quantized DCT coefficients are sampled along a zigzag line from the lowest to the highest frequencies, as shown in Figure 12.10. The resulting data stream of 64 bytes length has a high probability of a run of zeros towards the end. Run-length encoding of such a stream is highly efficient. Finally, the run-length encoded data are Huffman-encoded. The final JPEG-compressed image contains not only the compressed image data but also the Huffman table and the quantization tables, because the decoder cannot generate these automatically.

The decompression steps reverse the compression steps. First, the table of quantized coefficients is restored by decoding the Huffman-coded data stream and expanding the run-length encoded data into an 8×8 matrix, resulting in the table of quantized coefficients. These are multiplied with the values from the quantization

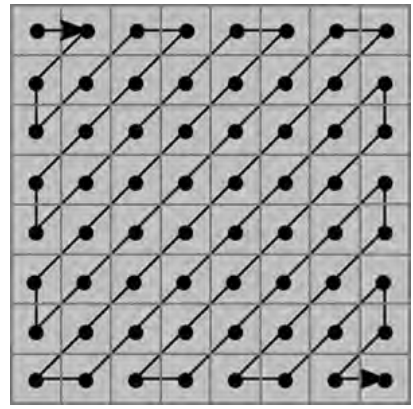


FIGURE 12.10 Sampling direction for the discretized coefficients $D(u,v)/Q(u,v)$. The pattern in which the two-dimensional subimage data are converted to a one-dimensional vector makes use of the high likelihood of having zero coefficients below one of the diagonals, thus optimizing the run-length compressed stream.

table and subjected to an inverse DCT. At this point, the reason for the data loss becomes obvious, because those coefficients of the DCT that were set to zero due to rounding remain zero during decompression. Generally, these are high-frequency components, and the 8×8 block has lost detail information. If the compressed image is a color image, the final step is the expansion of the chrominance components and conversion to the RGB color model.

JPEG encoding achieves good compression rates as long as a relatively high quality is chosen. Compression rates below about 90% show hardly any degradation. Minor image degradation can be seen at compression rates from 90 to 95%. At even higher compression rates, blocking artifacts become visible. These are a peculiarity of JPEG compression and are a consequence of the subdivision of an image into 8×8 or 16×16 squares which are compressed independently.

Typical JPEG artifacts are shown in Figure 12.11. The original image allows lossless TIFF compression with Packbits of 92%. With lossy JPEG compression, the

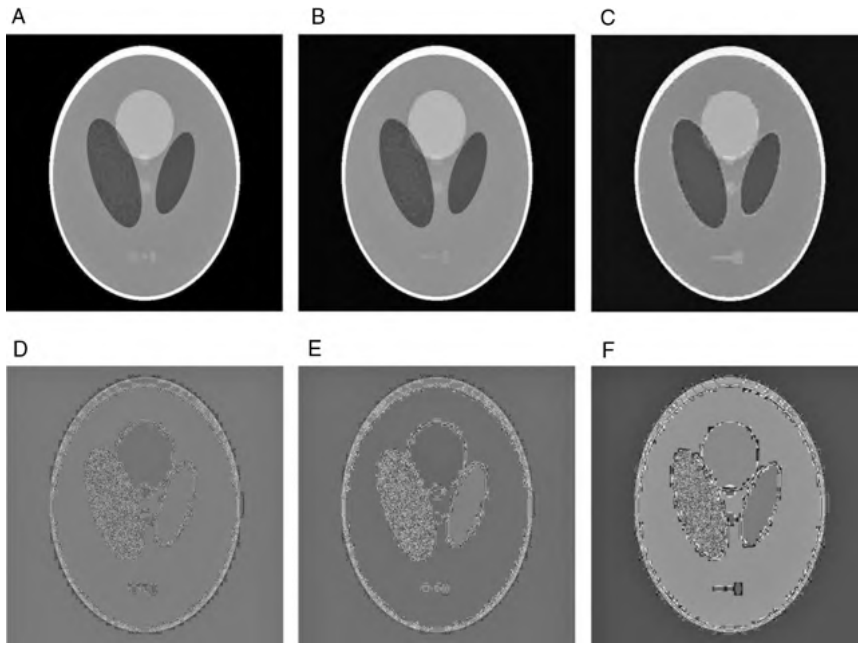


FIGURE 12.11 Demonstration of JPEG blocking artifacts. The original image is the Shepp–Logan head phantom with some artificially added noise in the left ventricle. Shown are decompressed images after 97.2% reduction ($q = 25\%$, A), 98.4% reduction ($q = 15\%$, B), and 99% reduction ($q = 5\%$, C). Below are the corresponding difference images of the original image (D–F). With increasing compression rates, the outlines of the 8×8 blocks become more and more visible, particularly near edges. The difference images make the lowpass character of JPEG compression particularly visible, because they show strongly the noise component and high values near edges. Lossless compression achieves about 92% compression rate with this image.

decompressed images show progressive degradation and the appearance of blocking artifacts after compression with rates well above 95% (notice the very obvious blocks near the "tumors" in the lower part of Figure 12.11C). The root-mean-square errors [square root of Equation (12.9)] of the images in Figure 12.11 to the original image are 6.0, 7.2, and 11.1, respectively. Due to the nature of DCT quantization, multiple compression and decompression operations at the same quality level do not further degrade the image.

The JPEG-typical blocking artifact becomes particularly predominant in pure binary (black-on-white) images, where JPEG compression causes new black pixels to appear. Other compression schemes prove superior for this special case. Such an example is the proprietary Cartesian Perceptual Compression method (http://www.cartesianinc.com/Tech/) or the freely available DjVu scheme.[2]

The introduction of the wavelet transform and its rapidly growing use in image processing applications provided a big advance in lossy compression. Wavelet-based compression was adopted in the JPEG-2000 standard and proved vastly superior to DCT compression at high compression rates. Wavelet-based JPEG-2000 compression[17] follows the same general outline of the earlier JPEG compression (Figure 12.9). However, the DCT step is replaced by a DWT step (discrete wavelet transform); the quantization step and the subsequent coding steps differ fundamentally. In addition, more consideration went into integer-only algorithms that do not cause rounding errors.

JPEG-2000 compression begins with a color model conversion. Whereas the YCbCr model [Equation (12.10)] is still valid within the JPEG-2000 standard, a modified YUV model based on integer-only operations provides an alternative color model transformation with minimal rounding errors (a maximum loss of 2 bits in the computation of Y):

$$\begin{aligned} Y &= \left\lfloor \frac{\mathrm{R} + 2\mathrm{G} + \mathrm{B}}{4} \right\rfloor \\ U &= \mathrm{B} - \mathrm{G} \\ V &= \mathrm{R} - \mathrm{G} \end{aligned} \tag{12.8}$$

The chrominance components U and V are not downsampled, because it is more straightforward to drop the finest level of the wavelet transform instead. The subdivision of the image into blocks of $n \times n$ pixels, where n does not necessarily have to be 8 or 16, is optional. This subdivision (*tiling*) is allowed to reduce memory requirements if necessary, but larger tiles, up to the size of the image itself, reduce the appearance of blocking artifacts.

The discrete wavelet transform is performed as described in Section 4.1. The JPEG-2000 standard allows us to choose between two wavelets. Both are from the Cohen–Daubechies–Feauveau family of wavelets. The first, defining a lowpass filter with nine coefficients and a highpass filter with seven coefficients, often referenced as CDF 9/7, is used for lossy compression. The second, using five coefficients for the lowpass and three coefficients for the highpass, is known as LeGall 5/3 and is used for lossless compression. The LeGall 5/3 wavelet filter uses integer coefficients and

therefore avoids the rounding errors associated with floating-point coefficients. The subsequent quantization step is similar to the JPEG quantization. Contrary to JPEG quantization, only one value of Q for each wavelet subband is allowed. Dividing the wavelet coefficients by the subband's value for Q and rounding to the nearest integer reduces the number of bits needed for each subband, and also sets small coefficients to zero. This step is the actual compression step associated with data loss. The value of Q determines the image quality after decompression. In the case of lossless JPEG-2000 compression, $Q = 1$ is chosen.

Subsequent entropy coding is performed by lossless arithmetic compression, a scheme known as *MQ coding*,[17] which is a specific form of context-based encoding with prediction (see Figure 12.5). The final data stream is organized to allow access to image subregions. The coding scheme called *EBCOT* (embedded block coding with optimized truncation)[27] is built around the idea of dividing each subband into rectangular blocks of coefficients and performing the encoding independently on each block. EBCOT allows limited random access to the bit stream.

Another feature of JPEG-2000 images is their ability to handle regions of interest. It is possible to define regions of the image where lower degradation or even lossless compression is applied. Particularly in the context of medical imaging, it is advantageous to use a lower compression in diagnostically relevant regions. A digitized x-ray mammogram, for example, could be compressed at a high rate, yet the region where microcalcifications are suspected remains uncompressed.

Examples of JPEG-2000 compressed images are given in Figure 12.12. The images are directly comparable to Figure 12.11 with similar compression rates. The block artifacts that characterized JPEG compression are much less dominant in the JPEG-2000 scheme, and the blurring occurs over a wider region. The root-mean-squared errors are consistently lower with JPEG-2000 compression than with JPEG compression, with values of 3.95, 5.85, and 7.38 in Figure 12.12A, B, and C, respectively.

Several other lossy compression schemes exist, but none of them have received the attention that JPEG-2000 received. Closely related to JPEG-2000 is the Progressive Graphics File (PGF) file format.[26] It uses the same basic principles as JPEG-2000, that is, lossy compression based on the wavelet transform and quantization of the wavelet coefficients. In contrast to JPEG-2000, PGF was optimized for speed. PGF features a somewhat lower compression efficiency over JPEG-2000, but it compresses and decompresses about five times faster. BTC or block truncation coding[18] uses a fundamentally different approach. The idea behind block truncation coding is the observation that image values show only small local variations. For BTC, the image is subdivided into small regions (typically 4×4 pixels), and for each block, the mean and standard deviation are computed. Any pixel that is larger than the local mean is represented by a "1"; all pixels below the mean are represented by a "0." Therefore, a 4×4 block can be coded in four bytes (mean, standard deviation, and 16 bits of image data), and the compression rate is at least 75%. The disadvantage of BTC is a severe loss of image information at relatively modest compression rates, and BTC never achieved appreciable popularity.

Apparent self-similarity of image regions was used by Barnsley and Hurd in their concept of fractal image compression.[1] The idea behind fractal image compression is

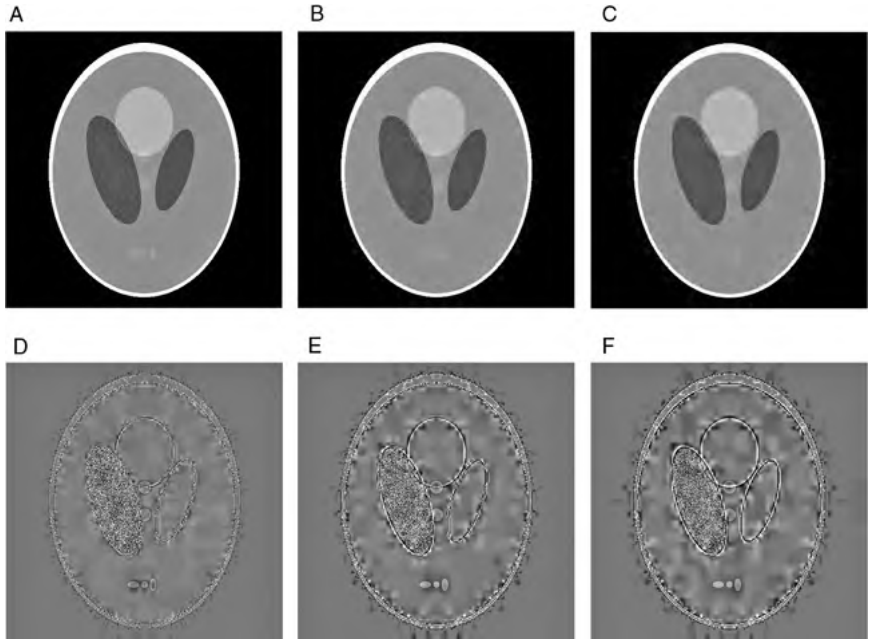


FIGURE 12.12 Demonstration of JPEG-2000 wavelet-based compression. The images are comparable to Figure 12.11 and have similar compression rates. Shown are decompressed images after 97.2% reduction (A), 98.4% reduction (B), and 99% reduction (C). Below are the corresponding difference images to the original image (D–F). JPEG-2000 compression shows less localized blurring and the absence of the 8×8 processing blocks seen in the JPEG DCT compression scheme.

the observation that some regions of the image are similar to others, but possibly on a different scale. In a general form, this representation was introduced as the Hutchinson operator in Section 10.1. More specifically, the image is searched for regions that can be described as the attractor of some iterative affine transformation. If this is the case, the image region is fully described by the parameters of the transformation. Fractal compression is patented, and details of the compression method are not readily available. Fractal compression probably enjoys its greatest popularity as the compression method of choice in the Microsoft Encarta encyclopedia. Notably, fractal image compression gives rise to fractal interpolation, a method that provides a more natural-based approach to zooming into an image.

Up to this point, still-image compression was discussed. Compression rates in video sequences are generally much higher than in still images, because the difference from one frame to the next is relatively small. It is therefore possible to predict the next frame with high accuracy. Consequently, the residuals of each predicted frame are small and can readily be compressed. The combination of within- and between-frame compression gives rise to lossy compression rates of 99.9% and higher. The most common standard is the MPEG (Moving Picture Experts Group) compression

standard. It defines both audio and video coding. The video compression is based on the DCT and builds to some extent on techniques introduced with the JPEG standard.

12.4. BIOMEDICAL EXAMPLES

Apart from the compression rate, the main concern that arises in biomedical imaging is to what extent a lossy compression affects the efficacy of an image to support a diagnosis. In other words, the optimum balance between storage and bandwidth constraints on one side and image detail preservation on the other side needs to be found. An objective measure of the difference between the original image $I(x,y)$ and the image degraded by compression and decompression $C(x,y)$ can be given by various methods. Popular metrics are the mean-squared error (MSE) and the peak signal-to-noise ratio (PSNR):

$$\mathrm{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[I(x,y) - C(x,y)]^2 \tag{12.9}$$

$$\mathrm{PSNR} = 10\log_{10}\frac{I_{\max}^2}{\mathrm{MSE}} \tag{12.10}$$

In Equation (12.10), $I_{\max}$ is the maximum possible value for the bit depth: for example, 255 for an 8-bit image. The square root of the MSE value in Equation (12.9) is also known as the root-mean-squared error (RMSE). Shiao et al.[24] suggested two more metrics. These are based on local windows rather than single pixels. The first is the Q index,[30] where a distance metric Q is calculated for each window:

$$Q = \frac{4\,\mathrm{cov}\,(I,C)\mu_I\mu_C}{\left(\sigma_I^2 + \sigma_C^2\right)\left(\mu_I^2 + \mu_C^2\right)} \tag{12.11}$$

where cov(I,C) indicates the covariance between images I and C inside the local window, μ_I and μ_C the mean values of the windows in images I and C, and σ_I and σ_C indicate the standard deviations of the values in the windows of I and C. Whereas sliding and tiled windows provide a quality map $Q(x,y)$, Wang and Bovic average the Q-values over all windows to obtain a single metric for the entire image.[30] The second metric is the Moran peak ratio.[7] The local Moran coefficient A is computed through

$$A = \frac{N\sum_{j=1}^{mn}\sum_{i=1}^{mn}\delta(i,j)(I_i - \mu)(I_j - \mu)}{2(2mn - m - n)\sum_{i=1}^{mn}(I_i - \mu)^2} \tag{12.12}$$

where m and n are the pixel width and height of the window, the delta function $\delta(i,j)$ assumes the value 1 for $i = j$ and 0 otherwise, μ is the average image gray value inside the local window, and $N = mn$. Local values for A need to be computed for both images I and C, and the histogram distributions of the Moran coefficients $A(x,y)$

are compared between both images. The Moran peak ratio is defined as the ratio of the highest peak values of the A-histograms of both images. Shiao et al.[24] analyzed the degradation of the images as measured with the Q index and Moran peak ratio with increased compression ratio, and compared DCT-based JPEG compression to wavelet-based JPEG-2000 compression. It was found that JPEG-2000 usually outperformed JPEG compression. The study also highlighted the difference between the application of windowed similarity metrics with sliding or tiled windows. Particularly JPEG compression, itself window-based, caused unusual trends in the similarity metrics at higher compression rates. A study that critically examines the usability of objective metrics was presented by Fidler et al.[12] In this study, CT images of the head were examined. The difference between compressed–decompressed images and their originals was quantified with the mean-squared-error metric. In addition, subjective perception of the degradation of test patterns was included. Two compression modes were compared, one with a constant quality factor and correspondingly constant MSE, and one with a constant compression rate. The authors suggest that a compression rate target may produce unreliable results, depending on the image entropy, and recommend using a variable compression rate with a targeted maximum MSE.

Neither MSE and PSNR nor windowed similarity metrics provide information on how the compression has affected the diagnostic value of the image. This effect can be shown in Figures 12.11 and 12.12, where the subtle "tumors" in the lower part of the phantom become less visible in the JPEG-2000 compressed Figure 12.12, although the RMSE is lower than in Figure 12.11. Whereas similarity metrics are useful for obtaining an objective value of quality loss, the diagnostic value needs to be assessed separately with the aid of diagnostic experts. An example are microcalcifications in digital x-ray mammograms. Microcalcifications are very small features that easily get blurred by the lowpass character of all lossy compression schemes. Chan et al.[6] compared the diagnostic value of digitized mammograms compressed with a pyramidal compression scheme and with a DCT-based compression scheme. With similar diagnostic value (determined through ROC analysis), the DCT-based compression provided higher compression rates, but 90% compression could not be exceeded.

Similar studies exist for other modalities and other sites. Bramble et al.[3] examined the quality of compressed digitized x-ray images when used to determine subperiosteal resorption of the phalanx. Image quality loss was simulated by quantization of the Fourier coefficients, a technique related to, but different from, the quantization of DCT coefficients. After Fourier transform, the Fourier coefficients were rounded to the nearest integer with 8, 7, or 6 bits; and final compression was performed with Huffman coding. The gold standard was the review by three radiologists, and comparisons were performed using the receiver operating characteristic (ROC). Average compression ratios of 94% (8 bits), 96% (7 bits), and 99% (6 bits) were achieved, but the ROC curve of the 6-bit compressed image differed significantly from the 12-bit image. For this task, therefore, a compression ratio of 96% was considered acceptable, and 99% compression rate caused unacceptable image degradation.

Following the same idea, Cosman et al.[9] examined the influence of compression on the diagnostic accuracy of lung CT images. The focus of this study, however,

was the development of suitable statistical methods for the observer-based analysis of compression-based image degradation. CT images of the chest were examined for the presence of nodules in the lung or the mediastinum. Three observers examined uncompressed images at 12 bits per pixel and compressed images with compression rates between 78 and 95%, that is, from 2.6 bits/pixel down to 0.57 bit/pixel. These compression rates may differ from conventional JPEG compression, since the authors used a predictive tree-structured vector quantization algorithm where a codebook is generated by training with a representative image. Although such a compression technique can provide better compression rates at the same level of image degradation, the compression algorithm was never adopted as a standard and is therefore not readily interchangeable between platforms.

A study that encompasses not only DCT- and wavelet-based compression but also the much less popular fractal compression method was presented by Ricke et al.[22] on digital x-ray images of the thorax and of a phantom. Radiologists were tasked with identifying abnormalities (lung nodules in the chest x-ray and detail texture in the phantom), and ROC analysis was performed to examine the influence of the compression method on diagnostic accuracy. For DCT-based JPEG compression, minor degradation was noted at a compression rate of 95% and extensive loss of quality at 97.5%. For fractal compression, the same levels of degradation were observed at 97% and 98.9%, and for wavelet compression at 98.8% and 99.1%. At similar compression rates, diagnostic accuracy was similar for wavelet- and fractal-based compression but markedly lower for DCT-based compression. However, at low compression ratios, DCT-based compression proved superior.

In these studies, there appears to be a compression limit around 95% where image degradation causes unpredictable results. Irrespective of the compression method, image archiving systems should not approach this limit, although higher compression rates appear technically feasible. In fact, on a recent consensus conference,[16] much lower compression rates were recommended: computed tomography at 80% for the brain and 87.5% for all other CT images; 90% for digital radiography, 93% for digital mammography, and 86% for MRI. These compression rates were found independent of the compression method.

As digital images—even x-ray images—become more and more digital, the question of the impact of digitization was raised.[20] Despite very high spatial resolution and a high image depth of 16 bits, this study revealed only a moderate match between diagnoses on film and on a digital workstation. However, there were only three radiologists in this study, and the moderate match was attributed primarily to one of the three radiologists. This finding may indicate that preferences, habits, or training have an influence. Nonetheless, digital radiology workstations become progressively more popular, and they provide a number of advantages over plain film x-ray. Examples include the ability to modify brightness and contrast, to zoom in, and to perform measurements rapidly. For modern image modalities such as MRI and CT, three-dimensional display and image fusion are possible. The choice of a suitable workstation is critical, however, and some requirements for an "appropriate" workstation have been reviewed by Krupinski and Kallergi.[14]

REFERENCES

1. Barnsley MF, Hurd LP. Fractal Image Compression. Natick, MA: A.K. Peters, 1993.
2. Bottou L, Haffner P, Howard PG, Simard P, Bengio Y, Cun YL. High quality document image compression with DjVu. J Electron Imaging 1998; 7(3):410–425.
3. Bramble JM, Cook LT, Murphey MD, Martin NL, Anderson WH, Hensley KS. Image data compression in magnification hand radiographs. Radiology 1989; 170(1 Pt 1): 133–136.
4. Cannavo MJ. Successful PACS implementation: the importance of planning and maintenance. Biomed Instrum Technol 2000; 34(4):269–273.
5. Carrino JA, Unkel PJ, Miller ID, Bowser CL, Freckleton MW, Johnson TG. Large-scale PACS implementation. J Digit Imaging 1998; 11(3 Suppl 1):3–7.
6. Chan HP, Lo SC, Niklason LT, Ikeda DM, Lam KL. Image compression in digital mammography: effects on computerized detection of subtle microcalcifications. Med Phys 1996; 23(8):1325–1336.
7. Chuang KS, Huang HK. Assessment of noise in a digital image using the join-count statistic and the Moran test. Phys Med Biol 1992; 37(2):357–369.
8. Cohen MD, Rumreich LL, Garriot KM, Jennings SG. Planning for PACS: a comprehensive guide to nontechnical considerations. J Am Coll Radiol 2005; 2(4):327–337.
9. Cosman PC, Davidson HC, Bergin CJ, Tseng CW, Moses LE, Riskin EA, Olshen RA, Gray RM. Thoracic CT images: effect of lossy image compression on diagnostic accuracy. Radiology 1994; 190(2):517–524.
10. Cowen AR, Parkin GJ, Hawkridge P. Direct digital mammography image acquisition. Eur Radiol 1997; 7(6):918–930.
11. Duchene J, Lerallut JF, Gong N, Kanz R. MicroPACS: a PC-based small PACS implementation. Med Biol Eng Comput 1993; 31(3):268–276.
12. Fidler A, Skaleric U, Likar B. The impact of image information on compressibility and degradation in medical image compression. Med Phys 2006; 33(8):2832–2838.
13. Gillespy T III, Rowberg AH. Dual lookup table algorithm: an enhanced method of displaying 16-bit gray-scale images on 8-bit RGB graphic systems. J Digit Imaging 1994; 7(1):13–17.
14. Krupinski EA, Kallergi M. Choosing a radiology workstation: technical and clinical considerations. Radiology 2007; 242(3):671–682.
15. Liu BJ, Huang HK, Cao F, Zhou MZ, Zhang J, Mogel G. Informatics in radiology (infoRAD): a complete continuous-availability PACS archive server. Radiographics 2004; 24(4):1203–1209.
16. Loose R, Braunschweig R, Kotter E, Mildenberger P, Simmler R, Wucherer M. [Compression of digital images in radiology: results of a consensus conference.] Rofo 2009; 181(1):32–37.
17. Marcellin MW, Gormish MJ, Bilgin A, Boliek MP. An overview of JPEG-2000. Proc DCC 2000; 523–541.
18. Mitchell OR, Delp EJ, Carlton SG. Block truncation coding: a new approach to image compression. Proc IEEE Int Conf Commun 1978; 1:12B.1.1–12B.1.4.
19. Ortiz AO, Luyckx MP. Preparing a business justification for going electronic. Radiol Manage 2002; 24(1):14–21.

20. Pudas T, Korsoff L, Kallio T, Uhari M, Alanen A. Influence of film digitization on radiological interpretation. Br J Radiol 2005; 78(935):993–996.
21. Rice RF. Some Practical Noiseless Coding Techniques. Technical Report JPL-79–22. Pasadena, CA: Jet Propulsion Laboratory, 1979.
22. Ricke J, Maass P, Lopez HE, Liebig T, Amthauer H, Stroszczynski C, Schauer W, Boskamp T, Wolf M. Wavelet versus JPEG (Joint Photographic Expert Group) and fractal compression: impact on the detection of low-contrast details in computed radiographs. Invest Radiol 1998; 33(8):456–463.
23. Santa-Cruz D, Ebrahimi T, Askelöf J, Larsson M, Christopoulos CA. JPEG 2000 still image coding versus other standards. Proc SPIE 2000; 4115:446–454.
24. Shiao YH, Chen TJ, Chuang KS, Lin CH, Chuang CC. Quality of compressed medical images. J Digit Imaging 2007; 20(2):149–159.
25. Smith-Bindman R, Miglioretti DL, Larson EB. Rising use of diagnostic medical imaging in a large integrated health system. Health Aff (Millwood) 2008; 27(6):1491–1502.
26. Stamm C. PGF: a new progressive file format for lossy and lossless image compression. J WSCG 2002; 10:421–428.
27. Taubman D. High performance scalable image compression with EBCOT. IEEE Trans Image Process 2000; 9(7):1158–1170.
28. Umbaugh SE. Computer Imaging: Digital Image Analysis and Processing. Boca Raton, FL: Taylor & Francis, 2005.
29. Wallace GK. The JPEG still picture compression standard. Commun ACM 1991; 34(4):30–44.
30. Wang Z, Bovic AC. A universal image quality index. IEEE Signal Process Lett 2002; 3:81–84.
31. Weinberger MJ, Seroussi G, Sapiro G. LOCO-I: A low complexity, context-based lossless image compression algorithm. Proc IEEE Data Compress Conf, Utah 1996; 140–149.
32. Weinberger MJ, Seroussi G, Sapiro G. The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Trans Image Process 2000; 9:1309–1324.
33. Welch TA. A technique for high-performance data compression. Computer 1984; 17:8–19.
34. Ziv J, Lempel A. A universal algorithm for sequential data compression. IEEE Trans Inform Theory 1977; 23(3):337–343.
35. Ziv J, Lempel A. Compression of individual sequences via variable-rate coding. IEEE Trans Inform Theory 1978; 24(5):530–536.

13

IMAGE VISUALIZATION

Two alternatives exist at the end of an image analysis chain. The image may be quantified; and one or a few numbers describe the features in the image. Or, the image is viewed by an observer, and a subjective conclusion is drawn. Image visualization techniques are methods to present the contents of an image in such a way that the observer can extract from it the maximum of information. Image visualization can be divided broadly into two different sections: the visualization of two-dimensional data and the visualization of volumetric image data.

13.1. GRAY-SCALE IMAGE VISUALIZATION

The simplest possible case is a gray-scale image with byte-sized image values: that is, image values in the range 0 to 255. Since most display units accept this range of values, image data can, without further manipulation, be stored in the display pixel buffer that is connected to the display. However, the results may not always be satisfactory. First, a translation of an image value into screen brightness is not always appropriate, and the physical meaning of the pixel value must be considered. In most instances, the translation of higher image values into lighter shades of gray is valid. Cameras and other light-intensity detectors accumulate the number of photons and provide a voltage that is proportional to the number of photons collected during exposure. This voltage is digitized, possibly scaled into the image value range, and stored as image value. Most display units behave in the same way. A higher image

Advanced Biomedical Image Analysis, By Mark A. Haidekker

value is translated into a higher luminescent value. A model in which higher image values translate into lighter shades of gray is often referred to as *min-is-black*, which indicates that the smallest image value corresponds to black and, by implication, the largest image value is white. Conversely, the *min-is-white* model translates the smallest value into white and the largest value into black. Most notably, the software program NIH Image, predecessor of ImageJ, used this model. It is rooted in the history of NIH image, which was originally designed to perform gel analysis. In a gel, a dark blot means higher exposure (or higher concentration), hence the translation of higher image values into darker shades of gray. Both models can be translated into the other with $I_2 = I_{\max} + I_{\min} - I_1$, where $I_{\max}$ and $I_{\min}$ are the highest and lowest image values, respectively, and I_1 and I_2 are the pixel-by-pixel image values of the respective model.

In Section 2.1, some techniques to enhance contrast were introduced. For visualization purposes, the actual image data are not modified. Rather, image data are translated in the display pixel buffer for display purposes only. One example was the gamma function to enhance contrast in the dark regions of the image. Another example that is very common in medical image modalities (e.g., built-in screens in ultrasound or CT devices) and in medical imaging software is the contrast and brightness adjustment referred to as *window/center*. The window/center function is a generalized linear translation of a brightness range from I_1 to I_2. For the purpose of this chapter we assume that a typical general-purpose display unit accepts values from 0 to 255. The window/center function can be described by

$$I_D = \begin{cases} 255 & \text{for } I > I_2 \\ 255 \cdot \dfrac{I - I_1}{I_2 - I_1} & \text{for } I_1 \leq I \leq I_2 \\ 0 & \text{for } I < I_1 \end{cases} \tag{13.1}$$

where I_D is the display brightness value and I is the original image value. The brightness range I_1 to I_2 can be computed from window and center values, w and I_c, with $I_1 = I_c - w/2$ and $I_2 = I_c + w/2$. The definition in Equation (13.1) can be modified to feature the window (intensity range) $w = I_2 - I_1$ and the center intensity $I_c = (I_1 + I_2)/2$. The window/center function allows to expand contrast in an image value range of interest, and it clamps the values above and below the window. This is associated with a complete loss of contrast in the clamped value range. The effect of the window/center function in Equation (13.1) is demonstrated in Figure 13.1: A CT image of the chest with a wide image value range (−1000 HU to more than 1000 HU) is remapped to the display value range (0 to 255) by using three different settings for w and I_c optimized for the lung interior, the soft tissue, and overall contrast, respectively. In addition to the window/center function, any of the contrast enhancement functions described in Sections 2.1 and 2.2 can be applied. This includes histogram stretching, histogram equalization, and nonlinear contrast enhancement functions, such as the gamma function.

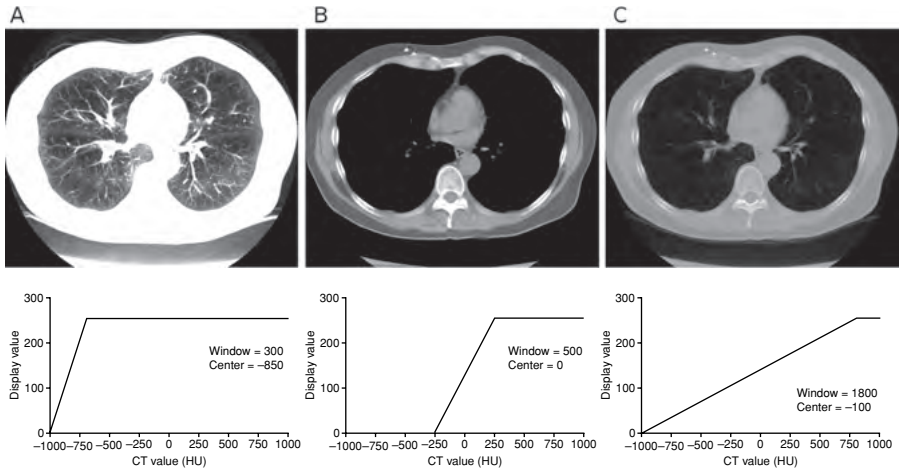


FIGURE 13.1 Effect of the window/center contrast adjustment. CT images typically have a wide range of image values, from −1000 (air) to more than +1000 (compact bone). The image values need to be remapped to match the display value range from 0 to 255. The remapping function in Equation (13.1) was used with different window and center values. A narrow window ($w = 300$) combined with a center value near air ($I_c = -850$ HU) enhances contrast in the lung tissue but leaves no contrast in soft tissue, some of the blood vessels, and bone (A). An almost equally narrow window with a higher center ($w = 500$, $I_c = 0$) provides good contrast in the soft tissue region, but the inside of the lung and the bone region have no contrast (B). A wide contrast window ($w = 1800$) with a center similar to that in part B provides contrast in all regions, but local contrast is low (C).

The question of contrast expansion arises whenever the image value range is lower than the display value range. Examples are binary images that contain only the values 0 and 1 and multithresholded images with few discrete values (e.g., 0 for background and 1, 2, and 3 for three features, respectively). In most applications, such an image would be displayed as almost-black on black with no discernible contrast. Contrast expansion to 0 and 255 in the first example and to 0, 85, 170, and 255 in the second example provides images with optimum contrast. However, when images are automatically contrast-expanded, the viewer loses the ability to compare two images directly, because the images values may have been scaled by different contrast functions.

Display devices with very steep contrast curves pose a different challenge. Examples are printers: With the exception of specialized halftone printers (photo printers), a printing device produces a black-and-white image—ink or no ink. In such a case, the image is either discretized into dots of variable size, or the printer prints a fine raster of dots, where the probability of a dot being either black or white depends on the gray value of the pixel. This process is called *dithering*. Although dithering plays only a small role in biomedical imaging (medical centers often use high-quality film printers for hard copies of biomedical images), it is mentioned here for completeness.

13.2. COLOR REPRESENTATION OF GRAY-SCALE IMAGES

A very popular visualization method for gray-scale images is false coloring. The human eye is much more sensitive toward different colors than toward different shades of gray. To generate a false-colored image, mapping tables are needed that map the image values of the gray-scale image to the components of a color image.

Color can be defined in a color model. A very common model is the RGB model (red–green–blue), where a color shade is created by adding the three base components red, green, and blue. Depending on the intensity of the individual component, a large number of different hues and shades can be created. Computer monitors use the additive RGB model. Typical computer monitors, either cathode-ray tube-based or liquid-crystal-based, have closely spaced dots with the base colors for each pixel (Figure 13.2). Since the pixel size is small, the eye cannot discern the individual color components and perceives a color that corresponds to the sum of the three components. A pixel, for example, where the green and red components are turned on, while the blue component is turned off, appears as yellow. If each component can be turned either on or off, $2^3 = 8$ different colors can be produced. Most computer monitors accept the intensity values of 0 to 255 for each base color, allowing a total of 256^3, or 16.7 million, colors.

The RGB model can be seen as a cube defined by a three-dimensional coordinate system with the R, G, and B axes. The cube fills the space from 0 to 1 along each axis, where 0 corresponds to a color component fully turned off and 1 corresponds

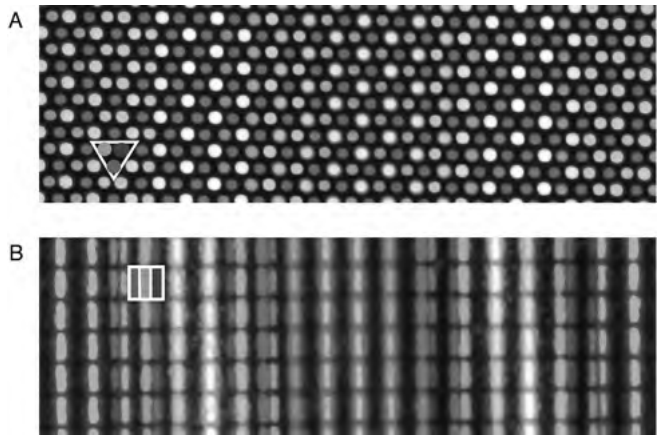


FIGURE 13.2 Demonstration of additive color generation. The images are magnified sections of photographs of a CRT screen (A) and a LCD screen (B), both displaying a color gradient blue–green–red–green–blue. In each image, one pixel is delineated (white triangle and square, respectively). Each pixel is approximately 0.25 mm in size. It can be seen that each pixel is composed of three components: red, green, and blue. Note that the camera that was used to take these photos has a high sensitivity for green, and the brightest green segments are overexposed. (*See insert for color representation of the figure*.)

to the color component at its maximum brightness. Each color is a point in this three-dimensional space. Several other color models exist: for example, the HSV (hue–saturation–value) model and the CMYK model (cyan–magenta–yellow–black) model. The HSV model provides a different color interpretation. It is often represented by a cylinder. The top ring of the cylinder covers all possible colors at their highest saturation (this range is often referred to as the *color gamut*), and each color corresponds to an angle θ in cylinder coordinates. By convention, red is placed at 0°, 120° is green, and 240° is blue. Mixed colors exist at 60° (yellow), 180° (cyan), and 300° (magenta). Any color is defined by a point in the HSV cylinder. The radial distance from the central axis corresponds to the saturation S (i.e., the color purity), and the elevation inside the cylinder is the brightness (i.e., the value V). The value is often referred to as intensity I, and the color models HSV and HSI are synonymous. A graphical representation of the RGB and HSV color models is shown in Figure 13.3. A conversion from the RGB to the HSV model can be performed with

$$
\begin{aligned}
\mathrm{V} &= C_{\max} \\
\mathrm{S} &= 1 - \frac{C_{\min}}{C_{\max}} \\
\mathrm{H} &= \begin{cases}
\dfrac{60(\mathrm{G}-\mathrm{B})}{C_{\max}-C_{\min}} & \text{for } C_{\max} = \mathrm{R} \\
120 + \dfrac{60(\mathrm{B}-\mathrm{R})}{C_{\max}-C_{\min}} & \text{for } C_{\max} = \mathrm{G} \\
240 + \dfrac{60(\mathrm{R}-\mathrm{G})}{C_{\max}-C_{\min}} & \text{for } C_{\max} = \mathrm{B}
\end{cases}
\end{aligned}
\tag{13.2}
$$

where $C_{\max}$ is the highest value of the color components R, G, and B, and $C_{\min}$ is the lowest value of the color components.

The CMYK model is found predominantly in printers and chemical color photography, where color layers subtract colors from white light. Correspondingly, the colors C, M, and Y can be obtained by subtracting R, G, and B from white. Since any color where C = M = Y produces an approximation of gray, it can be subtracted from the C, M, and Y components. Let $K = 1 - C_{\max}$ in the definition of Equation (13.2). In this case, the C, M, and Y components are computed with

$$
\begin{bmatrix} \mathrm{C} \\ \mathrm{M} \\ \mathrm{Y} \end{bmatrix} = \begin{bmatrix} 1-K \\ 1-K \\ 1-K \end{bmatrix} - \begin{bmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{bmatrix} \tag{13.3}
$$

In the CMYK model, one of the color components is always zero, because it is substituted by a K offset, which produces a better approximation of gray than the C = M = Y mix.

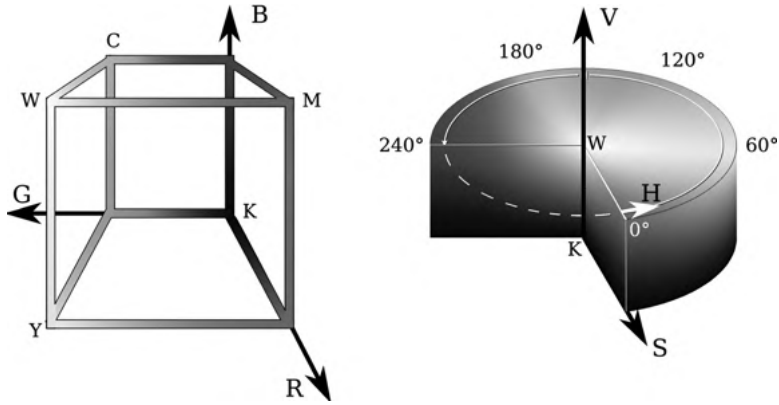


FIGURE 13.3 Graphical representations of the RGB and HSV color models. Each color represents one point inside the RGB cube, and its position is determined by the value of its R, G, and B components in Cartesian coordinates. Here, only the edges of the filled cube are shown. The edges of the cube give an example of the color gradients with black (k) and white (w) at diagonally opposite ends, and with the additive colors cyan (c), magenta (m), and yellow (y) at off-axis corners. The HSV model uses a cylinder coordinate system with the hue (H) as the angular component, the saturation (S) as the radial component, and the value (V, also known as intensity or brightness) as the axial component. The central axis of the cylinder covers the gray-value gradient from black to white. (HSV pie from http://en.wikipedia.org/wiki/HSL_and_HSV.) (*See insert for color representation of the figure*.)

The luminance–chrominance models are often used in image compression because the color component can be compressed more than the luminance component. The YCbCr model was introduced in [Equation (12.5)]. The luminance component $Y \approx 0.3R + 0.6G + 0.1B$ reflects approximately the sensitivity of the human eye toward the individual color components. A number of similar definitions exist. One example is the integer–YUV model in Equation (12.8). In the context of image visualization, these color models play a minor role, with the exception of the Y (luminance) component, which is useful for the conversion of color images to gray scale.

In the context of image compression (Section 12.3), the sensitivity of the human eye toward color and gray shades was discussed. In any local area of an image, the eye can distinguish approximately 20 shades of gray. The sensitivity of the eye toward different colors is much higher. A popular method to make a gray-scale image more accessible to the observer is *false coloring*, that is, a visualization method where each gray value is converted to a unique color. Because of the large number of possible colors, false-color schemes contain considerable redundancy, and a large number of false-color schemes can be designed. Two examples can be readily constructed. In both examples, the entire colors gamut (with the exception of magenta, which is often considered an unpleasant color) is covered, whereby dark gray shades are converted to blue, intermediate gray shades to green, and light shades to red. If the

image intensity is denoted I and the image value range is 0 to 255, the first color scheme translates I into the red, green, and blue color components z_R, z_G, z_B through

$$z_R = \begin{cases} 0 & \text{for } I < 128 \\ 1 + 2(I - 128) & \text{for } I \geq 128 \end{cases}$$
$$z_G = \begin{cases} 2I & \text{for } I < 128 \\ 1 + 2(255 - I) & \text{for } I \geq 128 \end{cases} \tag{13.4}$$
$$z_B = \begin{cases} 255 - 2I & \text{for } I < 128 \\ 0 & \text{for } I \geq 128 \end{cases}$$

The second example, often known as the *rainbow* scheme, extends the range where a color component is maximized and follows

	$I < 64$	$64 \leq I < 128$	$128 \leq I < 192$	$I \geq 192$
z_R :	255	$511 - 4I$	0	0
z_G :	$4I$	255	255	$1023 - 4I$
z_B :	0	0	$4I - 512$	255

(13.5)

where, for simplicity, the color components are provided in table form.

Both examples are very similar, but the color purity (saturation) is higher in the second example, because it is derived from the HSV model with the S and V components set to their maximum values and the H component linearly related to image intensity. Both false-color schemes have about equal luminance for all hues of color. Therefore, there are no dark color shades in any image that uses these two color schemes. Although this is sometimes desirable (e.g., when annotations or black contour lines need to be added), other color schemes are designed to assign dark color shades to dark gray shades and bright or white color shades to light gray shades. One example is a color scheme known as *Fire* in software packages such as NIH Image and ImageJ. Low image values are translated into black and dark blue shades; as the image values increase, the luminance increases while the hue changes over magenta to red, then yellow, and eventually white. In analytical form, such a color scheme can become quite complex and frequently is stored as a table [a lookup table (LUT)]. While rendering with lookup tables is very fast, the source image range is restricted by the number of table entries unless the image values are scaled or interpolated before the lookup table is applied. Typically, lookup tables have 256 entries for each color (red, green, and blue).

Figure 13.4 shows a gray-scale image (in this example the cross section of a mouse femur acquired with micro-CT) and three representations that were false-colored with different lookup tables. It can be seen from the graphs of the lookup tables in Figure 13.4 that the slopes of the individual color components are mostly greater than 1. With steep slopes, apparent contrast can be gained by changing colors rapidly between close image values. The *Terra* color scheme takes this idea to the extreme. Whereas the Rainbow [Equation (13.5)] and Fire schemes use smooth gradients, the Terra scheme uses steep transitions with plateaus in between. The consequence is the

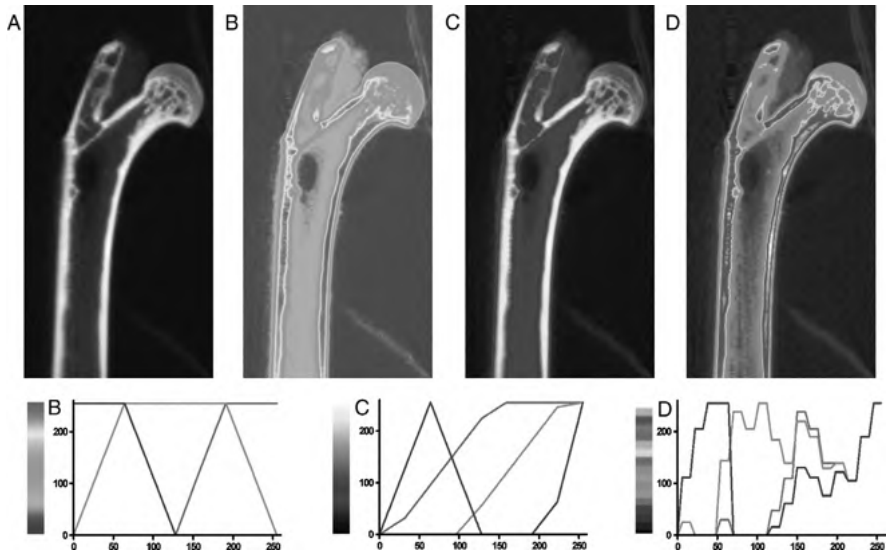


FIGURE 13.4 Gray-scale image of a micro-CT cross section of a mouse femur (A) and three false-color representations (B–D). In (B), the color scheme called Rainbow [Equation (13.5)] was applied, in (C) the color scheme called Fire, and in (D) a color scheme called Terra. The graphs underneath the color images are the corresponding lookup tables (the x-axis is the image gray value and the y-axis is the color component value). Whereas smooth transitions were desired in images B and C, the Terra color scheme in image D introduces abrupt transitions, which cause the appearance of contour bands. (*See insert for color representation of the figure*.)

appearance of bands of equal color that lead to the formation of pseudocontours in the image.

Another false-color scheme, used frequently in microscopy applications, changes the darkest shades of gray to blue and the lightest shades to red. Ideally, this color scheme is based on the histogram where all pixels below a certain percentile q (e.g., $q < 0.05$) are turned blue and all pixels above the percentile $1 - q$ are turned red. All other pixels retain their gray shades. In a simpler version, coloring is simply determined by a threshold, for example, $T = 0.05(I_{\max} - I_{\min})$. Pixels are colored blue when $I < I_{\min} + T$ and colored red when $I > I_{\max} - T$, whereas all other pixels retain their gray shades. The purpose of this color scheme is to more easily identify under- and overexposed pixels and quickly adjust exposure settings. When large image areas appear blue, for example, the technician increases exposure until the red and blue areas are approximately balanced. This idea can be extended to highlight thresholded regions with single and multiple thresholds when the thresholds are adjusted manually. The displayed image corresponds to the original image where the thresholded masks, appropriately colored, are superimposed over the image.

Another method of false coloring is to combine multiple (and related) gray-scale images into one color image by assigning the gray-scale components to color

channels. This visualization method has been used in various places in this book. In Figure 2.8C, for example, the original image occupied the blue and green channels (giving the original gray scale a cyan hue), whereas the zero crossings of the edge detection operator were added in red (i.e., the contrasting color). In Figure 2.8D, blue and green were used for the edge space and line space outputs of the Frei–Chen operator. Since the line space output had a fairly low contrast, the blue component was added in a nonlinear fashion to highlight the extreme values of the line space. In Figure 2.13A, the YCbCr model was used and the original histology section was assigned to the Y channel, thereby defining the detail contrast. The Cb component was derived from the fluorescent image and the Cr component from the local contrast (range operator). The composite image highlights areas of high fluorescence in blue–green hues and areas of high local contrast variance in red–orange hues. The advantage of this color scheme is the ability to retain detail information in the Y channel, while the chromaticity channels may contain blurred information with lower spatial detail.

False coloring is a complex topic, and often some experimentation is needed to achieve the visualization goal. There is virtually no limit in the number of possible color mapping schemes, but selection of an unsuitable color scheme may actually make the data less understandable. The popular rainbow scheme, for example, exhibits wide bands where the hue changes only minimally (most prominent in the cyan band in Figure 13.4B). Perceived contrast may be reduced. To select or design an appropriate color map, it is recommended that a visualization goal or strategy first be defined.[31] Moreover, the optimum color map also depends on the type of data: for example, the dominant spatial frequencies. At low spatial frequencies, the eye is more sensitive toward hue contrasts, and at high spatial frequencies, the eye is more sensitive toward luminance (intensity) contrasts.[31] This phenomenon is exploited in color image compression (see Section 12.3), where the chrominance information undergoes higher compression rates than the luminance information. According to Bergman et al.,[2] visualization goals fall broadly into three categories: isomorphic representation, segmentation, and highlighting. Isomorphic representation is probably the most common task. Suitable color maps follow the luminance of the original gray-scale image, as in the example of the Fire lookup table (Figure 13.4C). In an image with low spatial frequencies, the Rainbow color map would also be acceptable, the advantage of the latter being that black annotations or contour lines could be added.

Segmentation color maps create a structure with pseudocontour bands. A typical example is the Terra color map in Figure 13.4D. Depending on the spatial frequencies, the number of different color bands can be kept variable. With higher spatial frequencies, fewer color bands should be used. Color bands are not necessarily uniform but can be adapted to the information. A broad background peak in the histogram can be represented by a single band, and the feature is highlighted by several bands distributed over the intensity range of the feature of interest. Finally, highlighting-type color maps are related to segmentation color maps, but they have very few bands. In the simplest case, a highlighting color map consists of two bands of complementary colors, and the appearance is much like that of a thresholded image. A different highlighting color map can be thought of as consisting of two components: one isomorphic map, such as a color map that progresses smoothly from black over different

shades of blue, then cyan and green to light green. The highlighted intensity range would then be superimposed as a band of red.

13.3. CONTOUR LINES

Contour lines are known from topographical maps; they are lines that connect points of equal elevation. Contour lines help us understand the shape of the terrain. Analogously, contour lines in medical images may be useful in visualizing the shape of features. In gray-scale images, contour lines connect pixels of equal intensity value. For this reason, the term *isocontour line* is frequently used in the context of images. Usually, contour lines are generated for a number of intensity values, analogous to the different elevation levels in a topographical map. Traditionally, contour lines are represented by thin black lines. In dark regions of a gray-scale image, those contour lines would become difficult to discern. Although it is possible to switch to white contour lines in dark areas of a gray-scale image, most often contour lines are used superimposed over a false-colored image, specifically with a false-color scheme of high luminance such as the rainbow scheme in Equation (13.5). Isocontour lines can readily be extracted from the image. Some authors propose to use the gradient image,[28] but multilevel thresholding appears to be a more straightforward approach. A very fast isocontour line extraction algorithm could threshold the image at the different isocontour levels and mark boundary points as contour line pixels. A more refined approach takes into consideration the fact that the isocontour line generally passes between two pixels. Consider Figure 13.5, where individual pixels are displayed and the image values in the area between the pixels have been determined by interpolation. By applying a threshold, all pixels can be classified as either above the threshold (white) or below (black). The isocontour line will intersect any edge that is bounded by one pixel above and one pixel below the threshold. In Figure 13.5, the isocontour line is indicated as a white dashed line, and the intersection with the edges are marked with an **x**. These intersections can be found by interpolation, and connecting these intersections with straight lines leads to a polygonal approximation of the isocontour line (the thick gray line in Figure 13.5). A simplified algorithm could mark each pixel as a contour line pixel that (1) touches an intersecting edge and (2) lies below the threshold. This approach guarantees 8-connectivity of the resulting isocontour line. Isocontour lines are very sensitive to noise, and a noisy image can have highly irregular and widely meaningless isocontour lines. In this case, moderate blurring of the image is recommended before isocontour line extraction. For low-resolution images (such as 256×256 pixels or lower), resolution should be increased by interpolation. As a consequence, isocontour lines appear thinner and less dominant.

13.4. SURFACE RENDERING

A very attractive visualization technique for two-dimensional gray-scale images is to render them as a three-dimensional elevation landscape. Image values are interpreted

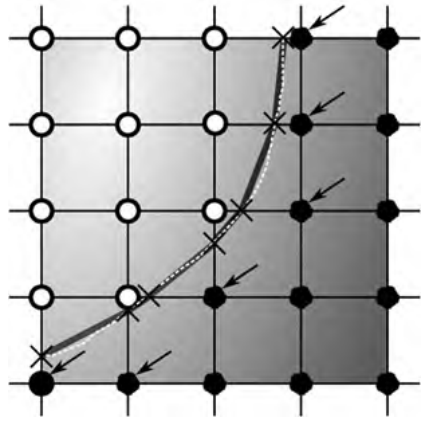


FIGURE 13.5 Finding an isocontour line. Pixels can either lie above the isocontour threshold (white circles) or below (black circles). Any edge that is bounded by one white and one black circle intersects with the contour line. With interpolation, the actual isocontour line (white dashed line) may be approximated by connecting the intersections of the isocontour line with the edges, marked x. The resulting polygon (thick gray) is a good approximation of the isocontour. A less refined approximation that does not need interpolation is generated by marking one pixel of each intersecting edge (arrows).

as the landscape elevation, and the shape of a feature as well as the relative height of different features can often be more easily appreciated in an elevation landscape than in the original gray-scale or even false-color representation.

The surface rendering procedure consists of three key steps. First, the surface is tesselated (subdivided) into triangles; second, the triangles are colored; and third, the triangles are projected onto a view plane. The projection step can best be imagined by considering an observer looking through a transparent window (which later coincides with the computer screen) onto a landscape that stretches behind the window. Let the observer be at location V in three-dimensional space. Each point of the landscape is seen by the observer at V, which means that each point of the landscape is connected with V by a straight line that intersects with the window. For the actual projection, each of these lines is computed, and at the intersection point with the window, the color of the corresponding point in the landscape is marked. For different projections, equations can be derived that determine where a point $P = (x, y, z)$ of the landscape projects onto the window at $Q = (x_w, y_w)$.[29] One relatively straightforward example is the isometric projection, where P and Q are related through the equation (see Chapter 11 for a comprehensive overview of linear spatial transformations)

$$[x_w \quad y_w \quad 0 \quad 1] = [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} -f\cos\alpha & -f\sin\alpha & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13.6}$$

Here α is the oblique angle of the projection and f is a parameter that determines the stretching of the depth coordinate. Equation (13.6) assumes that the coordinate system of the projection window has its origin at the bottom left corner and that the depth axis is the x-axis, pointing at the observer. In this case, the base (the floor of the landscape) is spanned by the x and y axes of the landscape, and the z-axis is the elevation. This arrangement makes intuitive sense when the image is considered as an image value (height z) as a two-dimensional function of x and y. Figure 13.6 explains how projection equation (13.6) can be derived.

It can be seen from Figure 13.6 and Equation (13.6) that three vertices of a triangle with arbitrary location and orientation in the landscape can be projected onto the projection window, and the result will again be a triangle. The corners of a rectangle (a quad, or, even more generally, any polygon) do not necessarily lie in a plane, but those of a triangle always do, because three points in space define a plane. The triangle is the preferred primitive because it is always planar. Any triangle within the three-dimensional space of the landscape can therefore be unambiguously projected onto the projection window by computing the projected two-dimensional vertices and filling the projected triangle with the same color as that of the original triangle.

To render (to project) a three-dimensional elevation landscape, it needs to be divided into triangles. Each group of four adjoining pixels $I(x, y)$, $I(x+k, y)$, $I(x+k, y+k)$, and $I(x+k, y)$ form two triangles. Integer values larger than 1 can be used for k to reduce the resolution for faster rendering and to achieve some smoothing of the rendered image. In the simplest case, each triangle is now assigned a uniform color. The color is determined by three factors: the base color, the illumination,

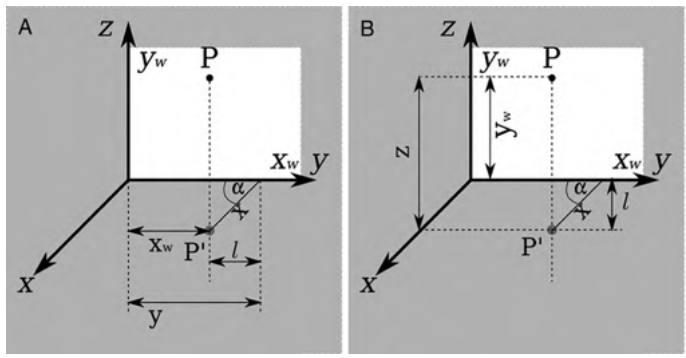


FIGURE 13.6 Derivation of the isometric projection equation for the window coordinate x_w (A) and y_w (B). The white rectangle symbolizes the projection window. The coordinate system of the landscape (x,y,z) has the same origin as the window coordinate system (x_w,y_w). Therefore, the z-axis and the y_w-axis coincide; so do the y-axis and the x_w-axis. The dashed vertical line projects point P in three-dimensional space to the projection point P' in the xy plane. From there it can be seen that the distance l equals $x \cos \alpha$ in part A and $x \sin \alpha$ in part B. With the help of l, the transformation equations $x_w = y - x \cos \alpha$ and $y_w = z - x \sin \alpha$ readily emerge. (From ref. 29.)

and the orientation of the triangle with respect to the light source and the observer. The simplest form of illumination is to render a triangle's color without change, that is, to assume a homogeneously distributed ambient illumination. However, this results in a flat and unappealing representation. In many cases, a point light source is placed at the same location as the observer, and light reflection from the surface is modeled. A frequently employed method to determine the appearance of a surface to the observer is the application of a comprehensive illumination model, known as the *simple illumination and reflection model* (see Foley et al.[9] for an overview). The simple illumination and reflection model contains ambient light and multiple light sources for illumination and models diffuse and specular reflection. Despite its name, the simple illumination and reflection model is of appreciable complexity and goes beyond the scope of this book. However, some of the basic concepts will be introduced in simplified form. Let the observer's viewpoint V be (x_v,y_v,z_v). Each triangle has a surface normal $N = (x_N,y_N,z_N)$, and diffuse light reflected from the light source to the observer dims with the angle θ between V and N. More precisely, it can be described by

$$I_S = I_0 \cos\theta = I_0 \frac{NV}{|N| \cdot |V|} \tag{13.7}$$

where I_S is the shading intensity and I_0 is the light source intensity. For simplicity, the normalization $I_0 = 1$ can be chosen. The addition of specular reflection gives the surface a metallic sheen and is modeled with a steep drop-off in intensity when the observer's position deviates from the angle at which the light from the source is reflected. Specular intensity is additive, and the shading intensity of a surface with diffuse and specular reflected light and ambient light can be computed with

$$I_S = I_0 + I_1 \cos\theta + I_2 \cos^n \theta \tag{13.8}$$

where I_0 is the ambient light intensity, I_1 and I_2 determine the balance between diffuse and specular light, and the exponent n determines how fast the specular intensity drops off. A low value for n (typically between 2 and 5) would be suitable for the appearance of dull plastic; higher n values (typically between 5 and 10) create the impression of shiny plastic or varnished surfaces; and very high n values (above 10) create a distinctly metallic or glassy impression.

The base color may now be selected by any of the methods described in Section 13.1 or 13.2. In the simplest case, the base color for all triangles could be set to white. Alternatively (and more frequently), a false-color scheme is applied. The resulting base color, determined, for example, by consulting the lookup table for the value at the average intensity of the three vertices, needs to be attenuated (i.e., multiplied) by I_S. Therefore, triangles that face the observer are brightest, and light emitted from triangles that are oriented at an angle to the observer are dimmer. Triangles that are at 90° to the observer are infinitely thin, and no diffuse light is reflected to the observer. If the angle exceeds 90°, the triangle is not drawn in the projection. At high levels of specular reflection, the visual impression can be improved further by

desaturating the color for small θ or by separating specular reflection and adding it to the diffuse reflection with a white base color.

Finally, the sequence in which the triangles are drawn matters, because triangles that are in the front of the scene occlude triangles in the back. A simple method to resolve the problem of occluded faces is known as *painter's algorithm*.[8] The scene is drawn from back to front, and triangles that are in the back of the scene are painted over by triangles that are drawn later (hence "painter's algorithm") and consequently, are removed from the projected image.

Additional flexibility can be gained by allowing the observer's position to change relative to the scene, and by creating a light source with a variable position. A variable observer position would violate assumptions for the isometric projection, and Equation (13.6) no longer holds. Rather, a more complex model of an axonometric projection[29]

$$[x_w \quad y_w \quad 0 \quad 1] = [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} -\sin\theta & -\cos\phi\cos\theta & 0 & 0 \\ \cos\theta & -\cos\phi\sin\theta & 0 & 0 \\ 0 & \sin\phi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13.9}$$

or perspective projection must be used. In Equation (13.9), ϕ is the elevation (0° in the zenith) and θ is the azimuth of the observer (and therefore the projection window) relative to the coordinate system of the scene. The added flexibility increases the complexity of the calculations, particularly when flexible light sources are involved. Fortunately, implementation of the projection is not necessary, since powerful three-dimensional rendering libraries exist. The most notable library is OpenGL, derived from Silicon Graphic's Graphics Library (GL), which was eventually made available under the terms of an open-source license. OpenGL can be understood as a software library that interfaces with the graphics hardware of the computer. Graphics hardware includes the pixel buffer (where the projected pixels are stored for viewing) but also any possible hardware components that perform mathematical operations. As can be seen from Equations (13.6) and (13.9), a lot of vector and matrix multiplications are necessary to render a complex scene. Today's computer graphics cards come with the ability to perform thousands of these matrix operations simultaneously. While the development of powerful graphics cards is driven by the high demands placed on the rendering hardware by today's computer games, any application that uses three-dimensional rendering benefits from this development. OpenGL performs most of the tedious functions of three-dimensional rendering, including rotation and translation of the scene relative to the observer, placement of light sources and illumination of the scene (OpenGL handles the complete *simple illumination and reflection model*), clipping of out-of-scene objects, hidden face removal, and eventually, the projection and generation of a frame buffer (*frame buffer* is the specific term for the viewable pixels inside the projection window). OpenGL can produce even more realistic scenes than the model above by interpolating the shading of the triangles (Gouraud shading instead of flat shading), as well as atmospheric effects such as fog.

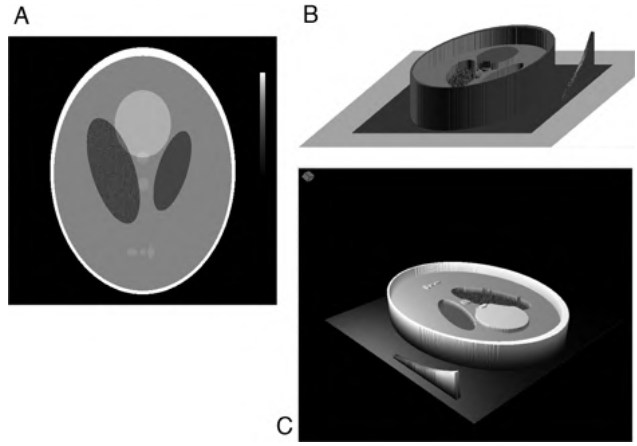


FIGURE 13.7 Examples of elevation landscape rendering. The source image (A) is the Shepp–Logan head phantom, with some additive noise in the left ventricle. A scale bar was placed to the right of the phantom. Rendering B was performed using the home-made isometric view that served as this chapter's example (Figure 13.6), and rendering C was performed with OpenGL. The red cube in the top left corner of image C is the light source. Elevation landscape rendering is particularly useful for recognizing shapes: From the original image, the intensity function in the scale bar at the right image edge cannot readily be determined. In the landscape renderings, however, the curved nature becomes obvious, and an anti-gamma function can be recognized. (*See insert for color representation of the figure*.)

The input to OpenGL is a tesselated scene, that is, a massive amount of triangles or quads with their associated colors and the position of the observer and light sources.

Figure 13.7 shows renderings of an elevation landscape. Shown is the Shepp–Logan head phantom with some additive noise in the left ventricle. The image in Figure 13.7B was created using the isometric view technique explained above [notably with the projection of Equation (13.6) and Figure 13.6], whereas Figure 13.7C was created using the OpenGL library. To illustrate the effect of illumination, the light source was rendered as a red cube, and both diffuse and specular reflection components were considered. Elevation landscape rendering is particularly useful to recognize shapes. An intensity scale bar is provided at the right margin of the original image. It is not easy to determine whether intensity increases linearly or in a nonlinear fashion. From the landscape renderings, however, it is immediately visible that it is an anti-gamma function that lowers contrast in the dark regions and enhances contrast in the bright regions.

13.5. VOLUME VISUALIZATION

Many modern medical imaging modalities provide three-dimensional images, either the intensity as a function of three spatial dimensions $I(x,y,z)$ or the intensity as a

function of two spatial dimensions and time, $I(x,y,t)$. Accordingly, display systems capable of projecting a three-dimensional image are under development.[3] However, this research area is still in its infancy, and it will take years until three-dimensional displays become mainstream equipment in medical image visualization. Therefore, the main challenge to display a three-dimensional data set on a two-dimensional display device remains. Traditionally, volumetric imaging devices such as CT, MRI, PET, SPECT, and three-dimensional ultrasound allowed to print the output image on film on a slice-by slice basis, that is, a tiled array of images $I(x,y)$ where each tile represents one slice in z. The special treatment of the z-dimension continues with the anisotropic voxel size; frequently, voxels are square in the xy-plane but have a larger interslice distance Δz. A clinical CT image, for example, could have an in-plane resolution of 0.2×0.2 mm and a slice thickness of 2 mm. However, in true three-dimensional rendering applications, no spatial dimension is different from the others as a consequence of the projection principles that were introduced in the preceding section. The visualization of three-dimensional images is fully based on the same principles of projection that were used to project and display elevation landscapes, but additional considerations come into play as the complexity of the three-dimensional image is markedly higher than that of a two-dimensional image.

13.5.1. Maximum-Intensity Projection

The maximum-intensity projection (MIP) is a simple, yet effective way to create a projection of a three-dimensional volume. A maximum-intensity projection along the z-axis is generated with the equation

$$I_{\text{MIP}}(x,y) = \max_z I(x,y,z) \tag{13.10}$$

where $\max_z$ symbolizes the maximum value of the image at one specific coordinate x,y for all z slices. The maximum-intensity projection creates the impression of a view through transparent parts of the volume onto the brightest elements of the volume. The maximum-intensity projection is not limited to projections along the z-axis. In fact, ray tracing can be applied to obtain the maximum-intensity value along angled paths or even diverging paths to create the illusion of perspective. A particularly elegant way of creating an illusion of a three-dimensional scene is to compute multiple maximum-intensity projections around a pivot point. Let us use the coordinate systems of the scene (i.e., the object to be projected) and the projection window in Figure 13.6. Let us also assume that the pivot point is in the center of the volume, and the viewpoint rotates along the azimuthal direction θ; that is, all projections occur parallel to the xy-plane. The maximum-intensity projection now follows:

$$I_{\text{MIP}}(x_w, y_w, \theta) = \max_s I(x,y,z) \tag{13.11}$$

where x_w and y_w are the coordinates of the projection window where $y_w = z$, and $\max_s$ symbolizes the maximum value along a path $s(\theta)$, where s is described by the

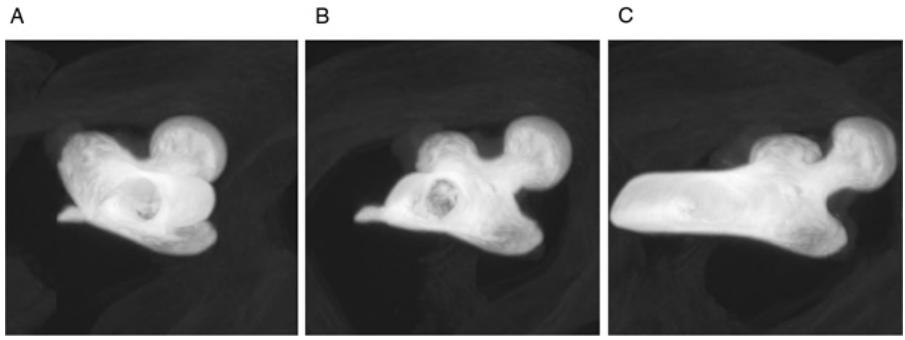


FIGURE 13.8 Maximum-intensity projections of a three-dimensional CT scan of a mouse femur (see Figure 13.4). The center image (B) was created from a projection parallel to the z-axis, while the other images were created by rotating the projection path by -18° (A) and $+18^\circ$ (C) around the y-axis.

equation of a straight line,

$$s: \; x \cos\theta + y \sin\theta = x_w \tag{13.12}$$

In practice, each path could be described by a start point P_1: x_1, y_2, z and an endpoint P_2: x_2, y_2, z where x_1 and y_1 lie on the projection plane outside the image volume, and x_2 and y_2 lie on the other side of the image volume. The line connecting P_1 and P_2 can be traversed efficiently by using the Bresenham algorithm and the maximum value along that line determined. Once the projections I_{MIP} (x_w,y_w) are computed for all θ, they can be displayed successively in a videolike fashion. Figure 13.8 shows maximum-intensity projections of the mouse femur CT image (see Figure 13.4) at three different angles (-18°, 0°, and $+18^\circ$) of rotation.

13.5.2. Volumetric Rendering of Apparently Solid Objects

If the outer surface of an object can be extracted by segmentation, the object surface can be tessellated, placed in a scene, and rendered in a similar fashion as the elevation landscape in Section 13.4. When the outer surface is a closed contour, the object gives the appearance of a solid, opaque object. Two steps are necessary to create a surface rendering: First, the object contour needs to be found, and second, the surface that was obtained in the first step needs to be tessellated. After the tessellation step, coloring, shading, and projecting the triangles are performed exactly as described in Section 13.4. In many cases, the object of interest has a different image intensity than that of the surrounding background. One example is the mouse femur, introduced in Figure 13.4. If this is not the case, image processing steps need to be taken to segment the object so that the object has a higher intensity than background. Once this is achieved, a very popular algorithm, called the *marching cubes algorithm*,[18] performs both steps in one sweep over the data: It finds

triangles that lie on an isosurface of the object and therefore tessellates the surface for rendering.

The marching cubes algorithm can best be explained in two dimensions, in analogy to the isocontour lines in Figure 13.5. In two dimensions, each square is bounded by four pixels that lie either above or below the isocontour threshold. Therefore, $2^4 = 16$ different configurations are possible. The isocontour line will intersect any square where the four nodes belong to different classes. Furthermore, any edge that connects nodes of different classes intersects the isocontour line. These intersections can be found by interpolation, and connection of these intersections with straight lines leads to a polygonal approximation of the isocontour line (thick gray line in Figure 13.5). In three dimensions, the squares become cubes, and there exist $2^8 = 256$ different configurations of node pixels. Also, an intersecting surface is found rather than a line. The surfaces that emerge from the intersections of the isosurface with the cube can have between three and six corners, and multiple surfaces can intersect the cube. A total of 15 unique configurations exist, with the remainder being rotated or mirrored configurations. Any surface with more than three corners gets subdivided into triangles. At the end, the surface is approximated by triangles where all corners of the triangles are found at intersection points of cube edges with the isosurface. The triangles, more precisely the tessellated isosurface, is now ready for rendering.

The marching cubes algorithm was further refined to improve efficiency,[23,33] to improve robustness and accuracy,[17] to eliminate ambiguous configurations that exist in the original algorithm,[20,22,27] and to address topological consistency.[26] However, the basic principle remains the same.

An example of a three-dimensional surface rendering is shown in Figure 13.9. In the CT image of the mouse femur introduced in Figure 13.4, the bone has a higher CT value than the background. The marching cubes algorithm can therefore be supplied with a threshold value that defines the isosurface. Note that the isosurface continues inside the bone, where the bone marrow has a lower CT value than the compact bone (Figure 13.4A). In Figure 13.4B it can be seen that the purely intensity-based segmentation element of the marching cubes algorithm is not ideal in this example. The femoral head shows signs of a threshold selection that was too high. In this case, presegmentation with an adaptive threshold or a slice-by-slice application of Otsu's method would have yielded better results.

A scene may contain multiple objects. In medical images, these objects are often related to each other: for example, different organs superimposed over the general anatomy. Figure 13.10A shows such an example. The visualization goal was to show the brain in its correct position inside the skull. The basis was a three-dimensional MR image of the head. Two different segmentation steps were applied to the image. First, the entire head was segmented and solidly filled. When the scene is rendered, only the outer surface of the head is shown, since the solidly filled head has no interior surface. Second, the brain was segmented with a modified region-growing process. Both head and brain were stored in separate images for visualization and placed in the scene to be rendered independently, yet while retaining their relative spatial orientation and position. To create Figure 13.10A,

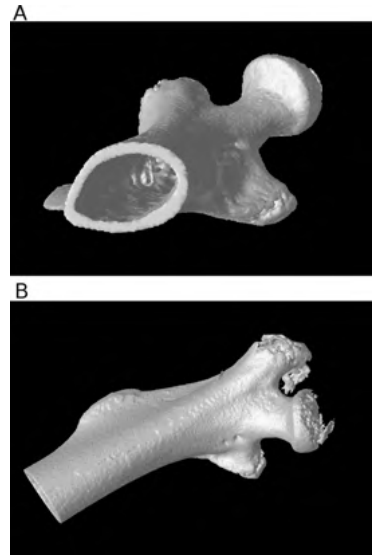


FIGURE 13.9 Three-dimensional renderings of the mouse femur CT image. Two different rotations are shown: the first (A) coinciding approximately with the MIP in Figure 13.8B, and the second (B) rotated to show the bone lengthwise. In part B, segmentation errors near the femoral head can be seen.

a prismatic section was removed from the rear of the skull image. In this section, the brain (which would normally be surrounded by the opaque skull) becomes visible.

In Figure 13.10B, a different concept was used: transparency. Up to this point, only solid, opaque objects were discussed. When rendering a scene, objects may be

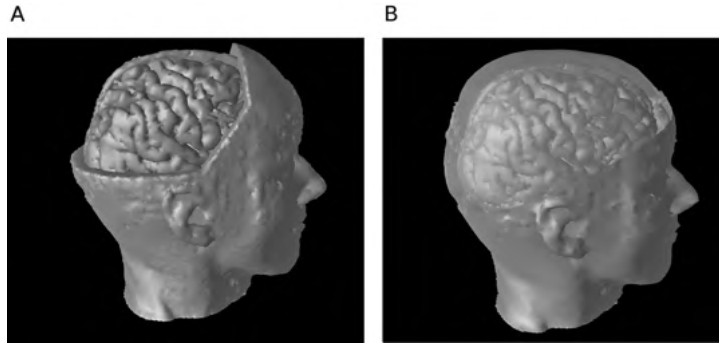


FIGURE 13.10 Visualization of the brain inside the head. The brain and the head were prepared as separate objects and placed in the scene for rendering. In image A, a section of the head is removed in the head image, whereby the brain is exposed in the removed section. In image B, a clip plane separates the upper section of the skull form the lower, and transparency was used in the upper section. (*See insert for color representation of the figure.*)

assigned a specific level of transparency in addition to the color. In color images, the transparency channel is often represented as a fourth channel, in addition to red, green, and blue. This channel is termed *alpha* by convention, and the color model that includes the alpha channel is RGBA. Alpha may assume any value between 0 and 1, where 0 is completely transparent and 1 is completely opaque. When the scene is rendered, any ray passing through an object with partial transparency gets assigned a weighted sum of the transparent object's color and the color of the object that lies farther back in the scene. The process of using the alpha value could be illustrated with the painter's algorithm. Let us assume that from back to front, three triangles (colored red, green, and blue) exist in the scene. The rearmost triangle has an alpha value of 1, the middle triangle of 0.5, and the foremost triangle of 0.2. With the painter's algorithm, the rearmost triangle would be painted first in red. Next, the middle triangle is painted. Wherever it occludes the red triangle, the colors are mixed: 100% red mixed with 100% green by using an alpha value of 0.5 becomes 50% red and 50% green, a color that has a dark yellow tint. In those regions where the green triangle occludes part of the (black) background, the color is 50% green. Let us write those colors in vector format as [0.5, 0.5, 0] and [0, 0.5, 0], respectively. The blue triangle is painted last. It adds a blue component, namely, 20% blue, and attenuates the color of the occluded objects by 20%. The first color now becomes [0.4, 0.4, 0.2] and the second color becomes [0, 0.4, 0.2]. This technique was used in Figure 13.10B, where the head object was split into two objects (lower and upper part), and the upper part was assigned a white color with an alpha value of 0.4. The brain was colored red to provide more contrast. The upper part of the head now appears semitranslucent and allows us to see the brain through it.

13.5.3. Volumetric Rendering with Transparency

A common technique to render solid objects is to use the gray-value information as alpha. Dark sections of the image (e.g., the air surrounding the patient) would be almost fully translucent, whereas bright objects, such as bone in a CT image, would appear not only white, but also fully opaque. Regions with intermediate brightness would appear like fog, partly obscuring the bright objects in the scene. In some respects, this technique is related to the maximum-intensity projection. However, in contrast to the maximum-intensity projection, solid rendering interprets each voxel as a solid cube with its unique color and transparency. Color can be added to the entire scene or to individual objects in the scene with false-color schemes. One example is the superposition of MR images with PET images in radionuclide studies: Areas of high radionuclide activity appear bright in PET images and are often vividly colored. Once PET and MR images are registered, the gray-scale MR image can be superimposed over the PET image by placing both images in the scene. Finally, the MR image is made partly translucent. The resulting rendered scene allows the observer to conveniently identify the regions of high activity inside the MR volume.

13.6. INTERACTIVE THREE-DIMENSIONAL RENDERING AND ANIMATION

As mentioned before, development of three-dimensional graphics is driven primarily by computer games, which require faster and faster three-dimensional rendering techniques. Volumetric medical image visualization benefits from this trend.[15] In fact, three-dimensional visualization evolves in two important directions: more sophisticated and more realistic (photorealistic) rendering, and virtual reality. With today's powerful computer graphics hardware, it is possible to create a scene projection within 0.01 to 0.1 s. It is therefore possible to display tens to hundreds of projections per second in a continuous sequence, thus creating the illusion of a movielike animation. With a technique called *double-buffering*, seamless transitions from one projection (in the context of animated movies, projections are called *frames*) to the next is possible. Double-buffering means that one frame is displayed while the next frame is rendered in the background. Once the new frame is completely rendered, it is displayed, thereby replacing the old frame. Meanwhile, the next frame is prepared. Two pixel buffers are necessary: One is displayed while the other is used for rendering in the background; then the roles are exchanged. The time needed for rendering one frame depends strongly on the number of triangles to be rendered and the rendering quality (e.g., shading interpolation), but also on the graphics acceleration hardware used. At frame rates above about 25 frames per second, the animation is perceived of as smooth. Three-dimensional rendering becomes interactive when the viewer can influence the scene by means of keyboard commands, mouse movements, or motions with specialized pointer devices. Although the mouse is the most widely used device, specialized three-dimensional tracking devices exist that provide information on six degrees of freedom (an example of its application is a virtual bronchoscopy workstation[11]).

To provide interactive features, the render window needs to record and interpret pointer or mouse movements. When the mouse is dragged (i.e., moved while a button is pressed) horizontally over the projection window, for example, the scene can be rotated around the z-axis (see Figure 13.6 for a definition of the coordinate system of the scene). When the mouse is dragged vertically, the scene is rotated around the y-axis. The projection window has only two dimensions, and rotation around the x-axis needs a different definition. One possibility is to use the second mouse button for the x-axis. Alternatively, and possibly more intuitively, the curvature of the mouse trace can be used to allow x-axis rotation: If the mouse is dragged along a straight line, the scene is rotated around the y- or z-axis, whereas if the mouse is dragged in a circular motion, rotation takes place around the x-axis. Very often, the scene is thought of as embedded in a sphere, and the mouse or pointer action moves, pushes, or rotates the sphere. In this fashion, the dynamics of the mouse gesture can be used to rotate the scene or to pan or zoom the projection. As an optional extension, inertia can be modeled with the sphere. If inertia is enabled, the sphere would continue rotating until another mouse gesture changes the spin. With rendering in real time, the scene becomes animated.

Animation becomes more general when the camera position can be controlled relative to the scene. One of the challenges of this approach lies in the degrees of freedom: Both camera position and direction—six degrees of freedom—need to be manipulated such that the scene is rendered in a meaningful way. Frequently, the software helps with this task by keeping the camera direction centered on the scene or by keeping the camera direction tangential to the path. The camera movement can now be manipulated interactively or with the help of the computer. The computer could, for example, identify a cavity and keep the camera centered inside the cavity. Such methods have been used in virtual colonoscopy, a biomedical example that is covered in Section 13.7.

For any given scene, the camera can, alternatively, be programmed to follow a fixed path. Many demonstration projects, for example, for the Visible Human Project,[1] make use of this technique. Interactive operation of the camera is needed only while the path is being determined initially. The camera path can be discretized by vertices and interpolated between vertices. For real-time animation, the camera path is traversed at a predefined speed. Movies can now be generated by saving the rendered frames at constant time intervals and converting the sequence of frames into a movie.

13.7. BIOMEDICAL EXAMPLES

Visualization is a very complex topic which includes not only the computer science aspects of image analysis but also the physiology of vision and the psychological aspects of perception.[37,38] In fact, a skillful visualization involves elements of art as much as it involves elements of science. In the context of medical imaging, however, there is a risk that information gets lost or distorted. Moreover, the complex computational processes associated with visualization may contain errors and flaws, and an incorrect rendering may in extreme cases lead to an incorrect diagnosis.[8] Careful planning, execution, and validation of the visualization is necessary. In fact, Thompson et al.[35] argue very strongly for a five-step process that begins with the formulation of a visual model of the data, a "vision" with clearly defined visualization goals rather than a trial-and-error approach that could lead to unsuitable data representation.

In addition to these aspects, data preprocessing may require considerable effort. In one example, the author was asked to provide an animated three-dimensional rendering of two major blood vessels (the aorta and the vena cava) from a CT image of a primate. The rendering was intended to serve more artistic than scientific purposes. The basis was a three-dimensional CT image of the primate, taken after injection of contrast agent to enhance contrast of the blood vessels. Figure 13.11A shows one CT slice. The two main challenges become immediately apparent. First, the contrast agent was distributed unevenly in the vena cava and created a U-shaped region of very high contrast. Second, the aorta did not contain any contrast agent at the time of acquisition, and segmentation was difficult due to low contrast. Consequently, the three-dimensional reconstruction provided by the built-in algorithms of the CT scanner (Figure 13.11B) was deemed insufficiently clear.

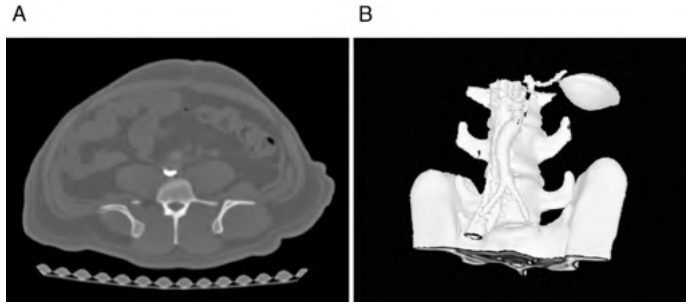


FIGURE 13.11 Visualization of the blood vessels in a three-dimensional data set of a primate. One representative CT slice is shown in part A. Whereas bone shows good contrast, the aorta exhibits poor contrast with the surrounding tissue, and the vena cava contains unevenly distributed contrast agent (U-shaped white region above the vertebra). A three-dimensional reconstruction provided by the CT scanner's built-in software was not considered to have enough quality.

A visualization goal was formulated to have the two blood vessels (colored red for the artery and blue for the vein) in spatial relationship to the spine, following the basic idea of Figure 13.11B. The primary challenge was the segmentation of the blood vessels. It was possible to segment the spine by intensity-based region growing, but any intensity-based method was precluded in the vena cava because of the low contrast and the presence of unevenly distributed contrast agent. However, the U-shaped contrast agent section provided two pieces of information: the location of the vein with an offset, and part of the inner wall: namely, from the outer curve of the U. After segmentation of the contrast agent region, the shape was used to complete a circle in each slice. The selection was validated by examining the Fourier coefficients of the trace of the circle's center through the CT slices, where high-frequency components indicated discontinuities in the segmented aorta. By removing the high-frequency coefficients, a smoothed trace was created and the vena cava reconstructed from the circles. The same a priori knowledge of a circular contour was used in the aorta, and by using a smoothed trace and the intensity gradient, the aorta could be extracted. Near the bifurcation some manual intervention was required because the shape deviated from the circle. Eventually, three segmented objects were available: the spine, the aorta, and the vena cava. These were arranged in a scene and rendered. Repositioning of the camera made it possible to provide 360 frames for a full rotation, and the frames were combined into an animated movie. Three frames are shown in Figure 13.12.

A rapidly evolving imaging technique that is discussed intensely in both the medical literature and in popular articles is CT-based colonography (see Rockey[30] for a recent review). Conventional colonoscopy involves inserting a fiber-optic camera system into the colon to diagnose polyps and colorectal cancer. CT-based colonography provides cross-sectional images of the abdomen. Beforehand, the colon is cleaned and gas-filled for optimum contrast. Images are normally evaluated on radiographic

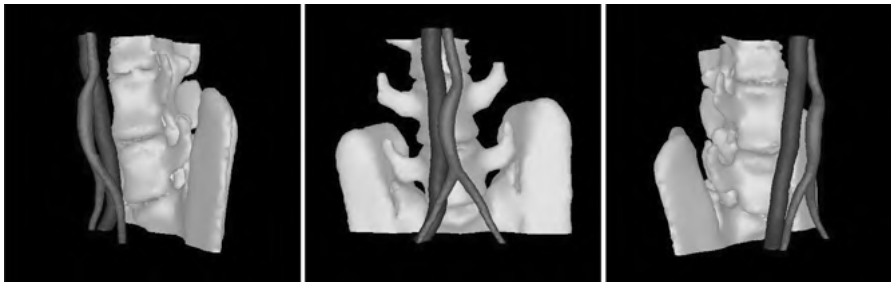


FIGURE 13.12 Three frames from the primate blood vessel animation. Aorta and vena cava are boldly colored (red and blue, respectively), while the spine was assigned a color with less saturation. Although some a priori knowledge was used for segmentation, the course of the blood vessels and the unusual shape of the aorta are clearly visible. (*See insert for color representation of the figure*.)

film on a slice-by-slice basis. With this method, large polyps can be detected with similar sensitivity and specificity as those for conventional colonscopy, but sensitivity diminishes somewhat with smaller polyps.[24] Virtual colonoscopy is a new method that relies strongly on three-dimensional visualization techniques.[41] Because of the high air/tissue contrast, the inner lumen of the colon is easily segmented and the centerline of the lumen determined. The centerline can help guide the camera, and the camera may be positioned and rotated by the radiologist to allow the optimum view of suspicious areas. A computer-aided system to provide a panoramic view and to cover as much of the inner surface as possible was proposed by Hassouna et al.[10] under the name *virtual flyover*. False coloring of the inner surface can be performed to provide a better illusion of endoscopy. Alternatively, the wall may be colored depending on its curvature: Long ridges or valleys are normal for the colon, but convex "bumps" are suspicious and can be highlighted in a different color. Comprehensive systems for shape-based classification of the colon wall are under development and promise to further aid the radiologist in the detection of polyps and suspicious lesions.[5,16]

Visualization plays a key role in virtual reality, a comprehensive simulation environment that becomes more and more popular in medical education. An overview of the early developments in medical application of virtual reality can be found in a review paper by Chinnock.[4] In the last 15 years, virtual reality has evolved into a mainstram area of medical science. Examples include the virtual reality simulation of robot-assisted colorectal surgery[19] as a typical example of a training simulation environment, the simulation of a virtual arm to study a patient's response to simulated parts of his own body,[34] and the recent development of a virtual patient to train medical students in routine doctor–patient interaction.[7] In the last example, the virtual reality environment has reached a point where speech recognition and scripted responses allow limited interaction between the medical student and the virtual patient. An example of a virtual reality training environment is shown in Figure 13.13 in a simulated ophthalmologist's exam room.[14] A large-size projection screen creates a life-size simulation and enhances the feeling of immersion. With a three-dimensional

FIGURE 13.13 Training of medical students with a virtual patient. The setting is an ophthalmologist's office, and the patient can be asked to perform various tasks, such as telling the number of fingers indicated, or following the hand with his eyes. In this simulation, the hand is controlled by the medical student by means of a three-dimensional positioning device. Simple clicks make it possible to extend a specific number of fingers. (Courtesy of Dr. Kyle Johnsen.) (*See insert for color representation of the figure*.)

positioning device, the medical student can control the virtual arm that is displayed inside the scene. The simulated patient can be asked to perform various tasks, such as following the arm with his eyes or telling how many fingers are raised. Diseases such as the inability of the patient to follow the hand with one eye can be simulated. With a click, the virtual hand can be brought to hold an ophthalmoscope. Close up, the ophthalmoscope can be seen to illuminate and magnify the virtual patient's eye.

A number of excellent examples for visualization techniques can be found in conjunction with the Visible Human Project.[1] From the early stages of the Visible Human Project, researchers started to develop segmentation and visualization techniques to render the anatomy of the Visible Human. Müller et al.[25] present a vision of the operating room of the future, where the integration of images from multiple modalities with image processing and visualization has the potential to improve diagnosis and treatment. One example of a Visible Human visualization is the development of an interactive brain atlas, a simulation and visualization tool that allows the user to move through the slices of the brain and to create and render a three-dimensional reconstruction of the brain.[36] Two basic Web-based applications exist, a Web-based interactive atlas[21] and a Web-based interactive image server.[12] Both focus on providing images from the Visible Human data set in an organized manner. The interactive atlas comes with annotations and makes it possible to choose between corresponding photographies, CT, and MR images. The image server makes use of interpolation and allows cross-sectional images to be created along arbitrary virtual cutting planes. Schiemann et al.[32] present methods to segment organs from the Visible Human data set and to perform realistic three-dimensional rendering. Ishimaru et al.[13] developed a virtual fly-through animation through the temporomandibular joint of the Visible Human as one step in the direction of virtual arthroscopy. The fly-through technique

has evolved beyond the Visible Human model and is now being applied to actual patients with coronary fly-through visualizations,[39,40] virtual laryngoscopy for tumor detection in CT images,[42] and virtual endoscopy for aneurysm surgery planning in the head,[6] to name a few examples from a large body of literature.

Visualization of medical image data marks one important endpoint of the image analysis chain. Visualization techniques cover the entire range from false coloring of gray-scale data to virtual reality. Software tools that help with visualization tasks are presented in Chapter 14. Those tools provide comprehensive help but are limited to fairly standard visualization tasks. Although software tools are flexible enough to accommodate the most common visualization goals, advanced visualization goals soon exceed the abilities of available software programs. Individual experimentation and programming play an even more important role, with the introduction of artistic elements. As such, visualization is clearly the subjective endpoint of the image analysis chain, whereas quantification, as described for example in Chapters 8 through 10, is the objective endpoint. Both endpoints have their respective values in medical imaging.

REFERENCES

1. Anonymous. The Visible Human Project. http://www.nlm.nih.gov/research/visible/visible_human.html, 2003. Accessed Oct. 2009.
2. Bergman L, Rogowitz B, Treinish L. A Rule-Based Tool for Assisting Colormap Selection. Washington, DC: IEEE Computer Society, 1995: 118–125.
3. Blundell BG, Schwarz AJ. Volumetric Three-Dimensional Display Systems. New York: Wiley, 2000.
4. Chinnock C. Virtual reality in surgery and medicine. Hosp Technol Ser 1994; 13(18):1–48.
5. Chowdhury TA, Whelan PF, Ghita O. A fully automatic CAD-CTC system based on curvature analysis for standard and low-dose CT data. IEEE Trans Biomed Eng 2008; 55(3):888–901.
6. Colpan ME, Sekerci Z, Cakmakci E, Donmez T, Oral N, Mogul DJ. Virtual endoscope-assisted intracranial aneurysm surgery: evaluation of fifty-eight surgical cases. Minim Invasive Neurosurg 2007; 50(1):27–32.
7. Deladisma AM, Cohen M, Stevens A, Wagner P, Lok B, Bernard T, Oxendine C, Schumacher L, Johnsen K, Dickerson R, Raij A, Wells R, Duerson M, Harper JG, Lind DS. Do medical students respond empathetically to a virtual patient? Am J Surg 2007; 193(6):756–760.
8. Elvins TT. A survey of algorithms for volume visualization. ACM SIGGRAPH Comput Graph 1992; 26(3):194–201.
9. Foley JD, van Dam A, Feiner SK, Hughes JF. Illumination and shading. In: Introduction to Computer Graphics. Reading, MA: Addison-Wesley, 1994.
10. Hassouna MS, Farag AA, Falk R. Virtual fly-over: a new visualization technique for virtual colonoscopy. Int Conf Med Image Comput Comput Assist Interv 2006; 9(Pt 1):381–388.
11. Heng P-A, Fung PF, Leung K-S, Sun H-Q, Wong T-T. Virtual bronchoscopy. Int J Virtual Reality 2000; 4(4).

12. Hersch RD, Gennart B, Figueiredo O, Mazzariol M, Tarraga J, Vetsch S, Messerli V, Welz R, Bidaut L. The visible human slice Web server: a first assessment. Proc SPIE 2000; 3964:253–258.
13. Ishimaru T, Lew D, Haller J, Vannier MW. Virtual arthroscopy of the visible human female temporomandibular joint. J Oral Maxillofac Surg 1999; 57(7):807–811.
14. Kotranza A, Johnsen K, Cendan J, Miller B, Lind DS, Lok B. Virtual multi-tools for hand and tool-based interaction with life-size virtual human agents. Proc IEEE Symp 3D User Interfaces 2009; 23–30.
15. LaViola JJ Jr. Bringing VR and spatial 3D interaction to the masses through video games. IEEE Comput Graph Appl 2008; 28(5):10–15.
16. Li J, Huang A, Yao J, Liu J, Van Uitert RL, Petrick N, Summers RM. Optimizing computer-aided colonic polyp detection for CT colonography by evolving the Pareto fronta. Med Phys 2009; 36(1):201–212.
17. Lopes A, Brodlie K. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. IEEE Trans Vis Comput Graphics 2003; 9(1):16–29.
18. Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. Pro 14th Annu Conf Comput Graph Interact Tech 1987; 163–169.
19. Marecik SJ, Prasad LM, Park JJ, Pearl RK, Evenhouse RJ, Shah A, Khan K, Abcarian H. A lifelike patient simulator for teaching robotic colorectal surgery: how to acquire skills for robotic rectal dissection. Surg Endosc 2008; 22(8):1876–1881.
20. Matveyev SV. Approximation of isosurface in the Marching Cube: ambiguity problem. Proc IEEE Conf Vis 1994; 288–292.
21. McNulty JA. LUMEN Cross-Section Tutorial. http://www.meddean.luc.edu/lumen/meded/grossanatomy/x_sec 1999. Accessed Oct. 6, 2009.
22. Montani C, Scateni R, Scopigno R. A modified look-up table for implicit disambiguation of marching cubes. Visut Comput 1994; 10(6):353–355.
23. Montani C, Scateni R, Scopigno R. Discretized marching cubes. Proc IEEE Conf Vis 1994; 281–287.
24. Mulhall BP, Veerappan GR, Jackson JL. Meta-analysis: computed tomographic colonography. Ann Intern Med 2005; 142(8):635–650.
25. Müller W, Grosskopf S, Hildebrand A, Malkewitz R, Ziegler R. Virtual reality in the operating room of the future. Stud Health Technol Inform 1997; 39:224–231.
26. Natarajan BK. On generating topologically consistent isosurfaces from uniform samples. Vis Comput 1994; 11(1):52–62.
27. Nieslon GM, Hamann B. The asymptotic decider: resolving the ambiguity in marching cubes. Proc IEEE Conf Vis 1991; 83–91.
28. Nishimura T, Fujimoto T. Fast contour line extraction algorithm with selective thresholding according to line continuity. Syst Comput Jpn 1994; 25(3):101–110.
29. Olfe D. 3D transformations. Computer Graphics for Design: From Algorithms to Autocad. Upper Saddle River, NJ: Prentice Hall, 1995; 290–322.
30. Rockey DC. Computed tomographic colonography. Curr Opin Gastroenterol 2009; 25(1):55–58.
31. Rogowitz B, Treinish L. Using perceptual rules in interactive visualization. Human Vision, Visual Processing and Digital Display V. Proc SPIE, 1994; 2179:287–295.
32. Schiemann T, Tiede U, Hohne KH. Segmentation of the visible human for high-quality volume-based visualization. Med Image Anal 1997; 1(4):263–270.

33. Shu R, Zhou C, Kankanhalli MS. Adaptive marching cubes. Vis Comput 1995; 11(4): 202–217.
34. Slater M, Perez-Marcos D, Ehrsson HH, Sanchez-Vives MV. Towards a digital body: the virtual arm illusion. Front Hum Neurosci 2008; 2:6.
35. Thompson D, Braun J, Ford R. OpenDX: Paths to Visualization. Missoula, MT: Visualization and Imagery Solutions, Inc., 2001.
36. Toh MY, Falk RB, Main JS. Interactive brain atlas with the Visible Human Project data: development methods and techniques. Radiographics 1996; 16(5):1201–1206.
37. Treisman AM, Gelade G. A feature-integration theory of attention. Cogn Psychol 1980; 12:97–136.
38. Tufte ER. The Visual Display of Quantitative Information. Cheshire, CT: Graphics Press, 1983.
39. van Ooijen PM, de Jonge G, Oudkerk M. Coronary fly-through or virtual angioscopy using dual-source MDCT data. Eur Radiol 2007; 17(11):2852–2859.
40. van Ooijen PM, Oudkerk M, van Geuns RJ, Rensing BJ, de Feyter PJ. Coronary artery fly-through using electron beam computed tomography. Circulation 2000; 102(1):E6–E10.
41. Vining DJ. Virtual colonoscopy. Semin Ultrasound CT MR 1999; 20(1):56–60.
42. Yan Y, Luo S, McWhorter A. Virtual laryngoscopy: a noninvasive tool for the assessment of laryngeal tumor extent. Laryngoscope 2007; 117(6):1026–1030.

14

IMAGE ANALYSIS AND VISUALIZATION SOFTWARE

The purpose of this chapter is to introduce the design ideas behind several image analysis software packages, as well as practical tips to getting started, so that the reader can apply the topics of this book in practice. For this purpose, the software should be readily and freely available, which leads to an introduction of the concepts of Free Software in the following section. Two image analysis software packages are covered in detail as representative examples. Because of the complexity of visualization tasks, software for visualization is covered separately.

The focus of the software examples in this chapter lies on those that are released under a *Free Software license*. Although there are literally hundreds of image analysis software programs available in the marketplace, Free Software has a distinct advantage: It gives the user the right to view and modify the source code. Free Software is rooted in a specific philosophy that distinguishes between free "as in beer," software for which no license fees are paid, and free "as in speech", software that comes with a number of specific freedoms, foremost among them the freedom to obtain the source code. A comprehensive overview of one popular philosophy of Free Software is presented by Stallman[10]: Free Software should be understood as a matter of liberty and not price. The freedoms granted with Free Software are (1) the freedom to run the program for any purpose and on as many computers as desired, (2) the freedom to study how the program works and adapt it to your needs, (3) the freedom to redistribute copies so that you can help other computer users, and (4) the freedom to improve the program and release your improvements to the public. For the second and fourth freedoms, access to the source code is a precondition. In the

Advanced Biomedical Image Analysis, By Mark A. Haidekker

early days of the computer, access to the source code was taken for granted. Most computer-controlled devices were delivered with complete program listings and even controller microcode. In the late 1970s, the concept of closed-source software began taking hold, and since then software has been distributed predominantly in the form of *binaries*, that is, precompiled machine code that is unreadable for humans. Closed-source software became dominant because of the increasing number of software patents, copyright claims, and trade secrets. However, software is different from other patentable intellectual property. Software is generally not monolithic but rather, can fall into several categories of protection.[5] Most notably, there exists a dichotomy between the source code and the compiled executable (binary) form. Initially, source code was interpreted by the courts as a form of literary expression and protected as such. However, there are many different ways to write code that produces the same results when compiled and executed in binary form. Conversely, almost identical code can produce vastly different results when executed on a computer. The courts have responded to this dilemma by adopting a more stringent application of the protection to the functional aspects of software.[5] As one consequence, trivial functional steps became patented. Examples include European patent EP 1,022,648, which deals with the progress bar; U.S. Patent 4,398,249, which covers the natural order recalculation of a spreadsheet; or U.S. Patent 6,935,954, which protects the notion of hallucinating computer game characters. In the software patent dispute, the European Patent Office even admits that patent law would allow protecting "activities which are so familiar that their technical character tends to be overlooked, such as the act of writing using pen and paper."[4] This trend of a growing number of disruptive trivial patents is aggravated by the complexity of modern software programs, which may infringe on any number of patents without the programmer even being aware of the infringement. Free Software aims to prevent a collision with disruptive patents and maintain the freedoms described above.

In a scientific setting, software with unpublished source code poses a serious drawback, as many data processing steps (e.g., image analysis, statistical analysis) cannot be reproduced by others. In fact, not even the users of closed-source software can know how their software has processed the data. Bugs may remain undiscovered, and unless software users take tedious steps of validating their software, incorrect data may enter the scientific literature and remain undiscovered. There is a stark contrast between the meticulous disclosure of the methods in scientific articles and the black-box nature of the data processing and analysis components in the same articles. For this reason, a strong case must be made in favor of Free Software in scientific applications not only with respect to the liberties listed above, but also with respect to transparency and reproducibility of scientific data.

As a consequence of restrictions involved with closed-source software, restrictive licenses, software patents, trade secrets, and copyright issues, a movement has emerged that supports and develops Free Software. Such software is often developed by a community of programmers who collaborate through the Internet, and the software is also available for free download through the Internet. Open-source software tends to come with a lot of features, because many developers contribute to it. In addition, the software tends to have a high standard of quality because of the inherent

peer review of open-source software. Popular Free Software usually has a very active community that helps with support, and most user questions are quickly resolved in Internet forums or Wiki-style knowledge repositories. One famous example for major community-developed software is the GNU/Linux operating system, which is renowned for its numerous features and for its stability and reliability. Free Software is often released under legally binding licenses that ensure the freedoms described above but also ensure that Free Software remains free. The GNU general public license (GPL) is one of the most widely used licenses, and the philosophy of the GPL is described in the preamble of the license agreement. The full text is available on the Web at http://www.gnu.org/licenses/gpl-3.0.html.

14.1. IMAGE PROCESSING SOFTWARE: AN OVERVIEW

For image processing a plethora of software programs exist. Numerous basic tasks can be done with programs such as Adobe Photoshop or its Free Software equivalent, the GNU image manipulation program (the GIMP, http://www.gimp.org). Among many functions to manipulate individual pixels and to annotate images, the GIMP can perform several of the basic image processing tasks described in Chapter 2. Examples are Gaussian blurring, noise filtering, speckle reduction, unsharp masking, sharpening, edge detection, morphological dilation and erosion, contrast enhancement, and false coloring. By using the gray-scale curve, manual thresholding is possible. Furthermore, regions can be selected by shape, color, or intensity, and operations can be restricted to the region selected. However, any image processing or analysis task that goes beyond the basic operations becomes progressively difficult with basic programs for pixel editing.

Camera manufacturers and manufacturers of imaging devices (e.g., microscopes) usually provide image processing software with a number of image processing operators and quantitative functions. Sometimes, the software is included in the hardware costs and the customer does not pay extra license fees. Often, however, the basic package includes a crippled version of the software, and an extra license fee must be paid to obtain the activation key to unlock the majority of the image processing functions. As an alternative option, the buyer of the imaging device may consider staying with the basic or crippled version that contains the necessary hardware drivers and supplement the package with a high-level free image analysis package. In the following section we describe three possible realizations of image processing software: menu-driven software, scripting languages, and visual image processing networks.

14.1.1. Menu-Driven Software

The most common type of image analysis program is menu-driven. The user is presented with a small control window that contains the main menu. From the menu, functions like opening an image file or applying an operator to a loaded image file can be accessed. The main advantage of menu-based image processing software is the low learning curve. Because of the menus, functions are easy to find even for users

FIGURE 14.1 Main control window of ImageJ.

who are not familiar with the software. On the other hand, the repeated application of a sequence of operations on different images quickly becomes tedious and prone to mistakes. For this reason, most menu-based image processing programs offer some sort of macro scripting language that allows automating complex tasks. The scripting languages of menu-based software are predominantly an alternative way to use the menu-based functions, but usually allow some flow control, such as loops and conditional execution.

One representative example, ImageJ (Figure 14.1), is an image processing and analysis package supported by the National Institutes of Health and is widely used in scientific and medical research applications. Its functionality and ease of use makes it suitable for entry-level and intermediate users. Moreover, ImageJ allows advanced users to extend the capabilities with macro scripts or plugins. ImageJ has a long development history. The predecessor version was known as NIH Image and written exclusively for Mac OS in a specific Macintosh dialect of the Pascal programming language. The original development goal was to create a tool for gel densitometry, and some specialized functions still exist that give testimony to this history. The restriction to Mac OS limited the use of NIH Image, and Scion Corp. ported the software to Windows, translated it to the C/C++ language, and extended its functionality to support their image acquisition boards. Scion Image is available free of charge but without the source code. Around 1997, development of the Java-based ImageJ began. ImageJ was intended to replace NIH Image and run on all major platforms, including Linux, Mac OS, and Windows. Java was chosen as the programming language because of its platform independency. After more than a decade of development with contributions from many programmers, ImageJ has become a very mature and stable software. The functionality of ImageJ can be extended by plugin modules, and more than 400 user-provided plugins are linked from the ImageJ Web page. ImageJ comes with comprehensive documentation, examples, and tutorials on the Web, and there is a Wiki-style information and documentation portal.

14.1.2. Scripting Languages: Matlab Image Processing Toolbox, Octave, Scilab, and IDL

Image processing functions can be built on top of high-level scripting languages such as Matlab. Matlab is a widely used high-level programming language for mathematical problems, most notably, numerical methods and matrix operations. Since an image can be interpreted as a large matrix, the extension of a matrix algebra toolbox by image processing functions is a natural step. Since Matlab is a high-level scripting

language, the functions in the Matlab image processing toolbox are accessible through commands similar to program statements. To use the image processing toolbox effectively, basic understanding of software programming is necessary. Matlab supports the construction of graphical user interfaces, and image processing applications can be controlled by a graphical user interface, but a graphical user interface needs to be programmed by the developer as well. GNU Octave (http://www.octave.org) is a Free Software equivalent to Matlab, that has a compatible scripting interface. At this time, the Octave image processing functions are not as comprehensive as those offered by Matlab, but many basic operators are present, and Octave is being actively developed and extended. Scilab (http://www.scilab.org), a high-level mathematical scripting language similar to Matlab, offers a lower level of compatibility with Matlab than with Octave. However, several sophisticated extensions for Scilab exist. For image processing and analysis, the Scilab Image Processing toolbox (http://siptoolbox.sourceforge.net) provides numerous image processing functions that are added to Scilab's scripting language. Finally, the nonfree Interactive Data Language (IDL) is a high-level scripting language for data and image analysis. Similar to Matlab's image processing toolbox, basic programming knowledge is required to develop image analysis applications. However, IDL offers a much larger number of built-in functions, thus allowing faster development of advanced image analysis applications. The main advantage of using a high-level scripting language for image processing is the availability of common programming-language elements (e.g., high-level operators and flow control with loops and conditions). The disadvantage is a relatively steep learning curve for users unfamiliar with the underlying language. An example for image processing steps driven by a script language is shown in Figure 14.2. In this example, Octave was used. Matlab commands would look somewhat similar.

14.1.3. Image Processing Networks: Khoros and MeVis-Lab

Khoros and MeVis-Lab are examples of the graphical representation of an image analysis chain that is developed and visualized as a connected network of operators. This type of software offers a canvas, and the user can place image processing operators on the canvas. Each operator has input and output tabs. The user can drag a line from an output tab to an input tab of a different operator, thereby establishing a sequence of image processing operations. This graphical approach allows rapid development of an image processing chain, and the operation of the chain can be grasped quickly, due to its graphical nature. A screenshot with sample image processing network is shown in Figure 14.3. The advantage over the scripting approach is the menu-driven selection and placement of operators. The advantage over the conventional menu-based approach is the repeatability of the functions: The image processing network can be reused with merely a different file name given to the input operator. Khoros and MeVis-Lab are nonfree software. The Khoros license comes with a developer kit that allows proficient programmers to extend the abilities of Khoros. A restricted version of MeVis-Lab can be downloaded from its home page (http://www.mevislab.de) free of charge, but the full version needs to be licensed. A Free Software alternative is not known at this time.

FIGURE 14.2 Image processing with a scripting language, in this example with Octave. The screenshot shows QtOctave, a graphical environment for Octave, and two images that were generated with a few commands. The commands can be seen in the QtOctave window and perform the following steps: (1) load an image file and store in the variable `im`; (2) convolve the image with a 5×5 matrix of 1/25 (box smoothing filter) and store the result in `sm`; (3) compute the gradients in the x and y directions; (4) take the maximum of the gradients in the x and y directions to approximate Sobel-like behavior; (5) display the smoothed image `sm`; (6) and (7) display the gradient image `D` in a new window.

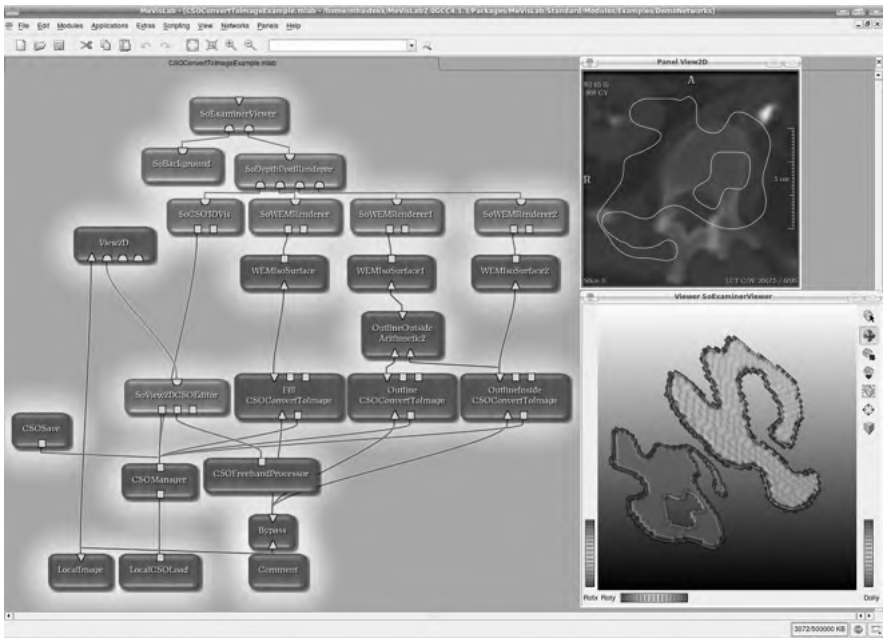


FIGURE 14.3 Sample screenshot from a graphical image processing network. This example shows a network created in the MeVis-Lab software to visualize contour lines. Data flow in the network runs somewhat counterintuitively from the bottom up, with the LoadImage operator in the lower left corner and the SoExaminerViewer, a rendering module, in the topmost position. To the right of the network, two display windows are open, one corresponding to the View2D operator, and the other to the SoExaminerViewer.

14.2. IMAGEJ

There are strong similarities between the functionality offered by various image analysis programs. With the example of ImageJ, the scope of a typical image analysis program is highlighted in this section. At the same time, we discuss briefly how to put ImageJ to use. ImageJ can be downloaded from its home page, http://rsbweb.nih.gov/ij/. From the same page, documentation and user-provided code can be accessed. Java files are available for Linux (both 32- and 64-bit versions), Mac OS (both 32-bit and 64-bit versions), and Windows. In all cases, installation is a straightforward process that can be completed within a few minutes, aided by the fact that the binaries are bundled with Java, and no extra Java installation is necessary. The executable file (usually linked to the program start icon) is a simple script that starts Java and tells it to execute the ImageJ Java binary `(ij.jar)`, which in turn opens the main window (Figure 14.1).

14.2.1. Image Files, Formats, and Types

An image can be opened in ImageJ from the `FILE` menu. In addition, the file menu offers functions to create empty images or images with a gradient and to open a sample image stored on the NIH server. ImageJ can open images in a number of commonly used formats. Formats supported include GIF, BMP, PGM, and JPEG as well as the DICOM medical image format and the FITS format (flexible image transport system), used predominantly in astronomy. Of particular interest is the capability of ImageJ to open stacks: Multiple images of the same size can be opened and assembled into a stack, that is, a three-dimensional image $I(x,y,z)$ or $I(x,y,t)$. A stack opens in a single window, and a scrollbar underneath the window allows us to browse through the stack. Related functions are the conversion of multiple open images to a stack, and vice versa. Another useful function is the import of raw data. Many image formats are composed of a header followed by the image data. The header contains image metainformation such as image size and bit depth, and the actual image data are a sequence of bytes that need to be interpreted according to the metainformation. With the raw format import function, the header can be skipped and the raw image data accessed directly. The user needs to specify the image format (size, bit depth, data type). This option works only for uncompressed image data.

Most image processing software supports gray-scale and color images with 8 bits per channel. Some software packages that are more quantitatively oriented also allow gray-scale values with 16 bits per channel. These software packages are often associated with high-end scientific cameras that support 10, 12, or 16 bits per pixel or are associated with imaging devices (CT, MR) that have a higher depth than 8 bits per pixel. In the example of ImageJ, 8- and 16-bit integers and 32-bit floating-point image values are supported. Unlike ImageJ, some image processing software that comes with imaging devices such as microscopes, cameras, or OCT devices provide a function to view the image stream from the camera in real time. This function facilitates focusing and exposure adjustment.

14.2.2. Manual Image Editing and Annotation

Many image processing programs allow manipulating individual pixels or regions of pixels. Some programs offer manual shearing and perspective distortions, and some programs even allow general rigid-body transformations. Most often, rigid-body transformations are either performed with a manually entered transformation matrix or with the help of fiducial markers. These functions create the basis for image registration. Most image processing programs also come with an annotation module that allows us to draw simple elements such as arrows, lines, circles, ellipses, or rectangles. In addition, labeling is possible. The annotation elements are usually merged with the pixels of the image, and the original image information that was overwritten by the annotation cannot be recovered. Exceptions are programs such as the GIMP, which makes it possible to arrange the elements on different layers. Until the layers are merged, the information remains separate and can be restored. However, layered image formats are usually specific to the individual program, and layer merging is necessary to export the image in a format that is universally readable.

In ImageJ, the `EDIT` menu allows access to some of these functions, and button shortcuts allow fast access to region select and annotation functions. The `IMAGE` menu of ImageJ contains additional functions to manipulate the image. This includes functions to crop, expand, rotate, and rescale the image, to adjust contrast and color balance, and to change the pixel depth (bytes per pixel). Also, color images can be decomposed into a three-slice stack with either the red, green, and blue information or the hue, saturation, and value (brightness) information in separate slices of the stack.

14.2.3. Global Image Manipulation and Filtering

The main functionality of ImageJ can be found under the menu items `PROCESS` and `ANALYZE`. The `PROCESS` menu offers the typical image processing operators, starting with `SMOOTH` (convolution with a 3×3 Gaussian kernel), `SHARPEN` (convolution with a sharpening kernel as described in Section 2.3), `FIND EDGES` (Sobel operator), and contrast-enhancement functions. Two contrast-enhancement functions are offered: histogram stretching with the option to stretch past the data value range allowed (to achieve a user-selectable percentage of saturated pixels) and histogram equalization. Histogram equalization is applicable only to integer-type images (8 and 16 bits/pixel) and is not implemented for floating-point and RGB images. A number of spatial-domain filters are implemented, including a generalized convolution with a user-editable kernel, a convolution with a Gaussian kernel of selectable standard deviation, unsharp masking, and local variance filters. Furthermore, several rank filters are offered: maximum (gray-scale dilation), minimum (gray-scale erosion), median, and mean filter. The mean filter differs from the median filter, as it replaces the central pixel by the mean value of the neighborhood rather than the median value. The effect is very similar. A number of image math functions are implemented: monadic operators (operations that act on one image) such as raising the image values to the power

of a constant γ, taking the exponential or logarithm of image values (note that log 0 is arbitrarily set to zero), and taking the reciprocal of the image values; and dyadic operators (operations that need two equal-sized images) such as addition, subtraction, multiplication, division, and bitwise logic operators. A `MAKE BINARY` function exists that applies Otsu's threshold. Morphological operators that act on binary images are erosion, dilation, opening, closing, outlining, filling of holes, skeletonization, and watershed segmentation. Interestingly, ImageJ occasionally switches to a min-is-white representation (it announces this by declaring its use of an inverting LUT), and threshold-based segmentation may assign low image values to foreground and high image values to background (yet keep binarized low image values black and high image values white by merit of the inverting LUT). The sometimes confusing switch from dark to white background is rooted in the history of NIH Image, which was designed as a densitometry software. The `PROCESS` menu further offers the Fourier transform and Fourier-based filters. The `FFT` function performs by default the fast Hartley transform and returns one real-valued frequency-domain image. By selecting specific options, the actual complex-valued Fourier transform can be computed. A flexible Fourier-based bandpass filter is offered that can also serve as highpass and lowpass filters, and, optionally, can remove periodic components parallel to the x and y axes simply by setting to zero the values along the u and v axes. The resulting FFT filter mask can be displayed, which is a helpful feature for debugging a FFT bandpass filter. Also, a general FFT filter is possible where a second image serves as the filter function. In practical experiments, the FFT filter functions failed when the source image was an integer-type image, and conversion to 32 bits/pixel was necessary.

The `PROCESS` menu offers several additional functions, such as the artificial addition of image noise (a function useful to study noise reduction filters), the overlay of a shifted image to create a shadow effect, and finally, a background flattening function. Background flattening is realized with the *rolling ball algorithm,*[11] which can best be described when the image is interpreted as an elevation landscape where the background represents the highest elevation points. A sphere that rolls over the landscape in scan lines touches the landscape at certain points, and the elevation at those points is considered the local background. Pixels that have never been touched by the ball (normally, feature pixels) are interpolated from nearby known background pixels, and the resulting background image is finally subtracted from the original image. This algorithm tends to produce blocklike artifacts in images where the background is not clearly defined. In such a case, the FFT bandpass filter may be a more suitable choice.

One of the problems of ImageJ becomes apparent at this point: When several images are opened at the same time, the image on which an operator acts is not always predictable. In practical use, it may be useful to have ImageJ open as few images as possible (ideally, a single image) and, instead, run multiple instances of ImageJ. Clearly, multiple images need to be opened when images need to be converted to a stack, to a color image with R, G, and B channels, and when image math needs to be performed. In all other cases, keeping only a single image window open avoids ambiguity.

14.2.4. Image Measurements

Image processing software most often allows us to obtain numerical values for individual pixels and pixel values along a straight or curved line (intensity profile). Most image processing software also makes it possible to zoom in on the image to make individual pixels visible. Furthermore, programs allow us to examine the histogram and obtain statistical information on the image value distribution. The `ANALYZE` menu in ImageJ contains operations to measure elements of an image, most notably gray-value analysis and histograms. Gray-value measurement and histograms can be restricted to a region of interest. With an option dialog, additional metrics can be enabled, such as median, mode, standard deviation, and higher-order statistics (skew, kurtosis). A versatile function exists for particle analysis. Particle analysis is useful, for example, in counting cell colonies or characterizing fluorescently labeled cells. Particle analysis requires a binarized (segmented) image. A summary with feature count, size, area fraction, and average size is obtained together with a table that lists for each feature the area, gray-value statistics, bounding box, centroid, and several selectable shape parameters. From the detailed results table, a histogram of size distributions (and other distributions, such as perimeter or circularity) can be obtained. ImageJ offers the extraction of intensity profiles along straight lines and rectangular selections. For rectangular selections, the profile tool displays the gray value, averaged along vertical scan lines, as a function of the horizontal position. Finally, ImageJ offers functions to calibrate size and optical density. Images can be calibrated through the `IMAGE PROPERTIES` function, where the pixel size can be entered. After calibration, all measurements use the calibration value. For the optical density, a stair-step calibration phantom can be used, and the average gray value is then related to known density in a calibration table. Additional built-in tools include fractal box counting and a some of gel-specific analysis tools. These are very well-developed functions and among the strongest features that make ImageJ suitable for quantitative image analysis.

14.2.5. Visualization Tools

The most frequently offered visualization tools are linear and nonlinear gray-scale enhancement, color lookup tables, and rendering of an image as three-dimensional elevation landscape. The ImageJ `IMAGE` menu offers access to these functions. ImageJ's built-in viewer for three-dimensional elevation landscapes creates a fixed view that is independent of hardware-acceleration features. ImageJ comes with a large number of color lookup tables for false coloring. A convenient feature is a function to inset a color scaling bar into an image.

14.2.6. Macro Scripting Language and Plugin Extensions

Two important components of an image analysis program determine its versatility through automation and extensibility: a macro scripting language and the option to

create extensions, called *plugins* in ImageJ. The macro language is usually some form of high-level language, normally interpreted and not compiled, that defines sequences of operations and offers some additional flow control (conditional execution, loops, and function calls). Plugins are normally written in the same language as the software itself, in this case Java, and the programmers are required to follow a specific format to properly integrate the plugin with the main software.

The macro language of ImageJ is a mix of Pascal and C syntax elements combined with custom elements. The Pascal-style elements are rooted in the history of ImageJ because the macro language of NIH Image was highly Pascal-like. Apparently, the macro language interpreter received a moderate overhaul when it was integrated in ImageJ. Fortunately, a macro recorder exists that records image processing operations as they are applied and allows editing of the resulting script. The macro language is highly interactive, allowing us to create interactor elements (text boxes, input windows, sliders, dialogs). As all programming languages, the ImageJ macro language knows variables. Variables can contain numerical values, strings, or arrays—all at the same time—since macro variables are not restricted to a single type. Images cannot be accessed directly. All calls to image operators act on the active window, and an image can be activated by its title. This strategy requires that all images be opened in their respective windows for a macro to access them. The ImageJ macro language is well documented (http://rsb.info.nih.gov/ij/developer/macro/macros.html) and easy to use for anybody, even someone with minimal programming experience. To help further with developing macros, a repository with more than 300 macro examples exists (http://rsb.info.nih.gov/ij/macros/). As such, the macro language is ideally suited for simple automation of repetitive tasks. Yet the macro language is powerful enough to allow the implementation of relatively complex analysis tasks, such as segmentation of small bones for x-ray densitometry.[6] Since the macro language is interpreted, however, macros execute comparatively slowly. For the implementation of complex algorithms, the plugin architecture is better suited.

ImageJ plugins are compiled in Java code. *Compilation* means that the code is translated into machine language for execution. Conversely, an interpreter (such as the macro interpreter) reads a human-readable text file and parses the text for instructions. The parsing process is relatively slow, and inside nested loops the time used for parsing can add up considerably. Compiled code does not have this disadvantage and runs more efficiently. In addition, plugins can be developed with the full functionality of a programming language and are not restricted by the scope limitations of the macro script language. On the other hand, writing a plugin requires considerable programming experience, notably with the object-oriented Java language. A book that covers the basic image processing with special consideration of ImageJ as the processing tool was published recently.[3] The book contains a detailed tutorial on how to write ImageJ plugins, and the authors have also provided a tutorial written by W. Bailer to create plugins. It is accessible on their Web site (http://www.imagingbook.com/index.php?id=102). The plugin concept is fundamental to ImageJ, because most of the menu commands of the ImageJ package are written as plugins. For this reason, parts of the ImageJ source code can be used as templates to implement new plugins. Each plugin contains a number of components: a plugin

class that implements the user interface for the plugin, a plugin filter class that contains a variable to hold the image to work on and a setup method to initialize the plugin filter, and an image processor class that performs the actual image processing function. Each plugin must provide a certain minimum set of methods to work with ImageJ. Detailed instructions on how to write an ImageJ plugin are beyond the scope of this book. The interested reader is referred to the online documentation that is listed in this section.

14.2.7. Additional Functions and Operations

Depending on the area of application, image processing software offers additional functions, many of which are covered in this book. Whereas the basic functions listed above are common to most image processing programs, the software packages differ most in the implementation of advanced functionality. Often, a program evolved from a specific task, and many operators related to this task are present. Other programs perform modality-specific functions. For example, CT and MR scanner software often offer sophisticated segmentation algorithms. Microscope software sometimes offers particle tracking. Built-in software in medical imaging devices is programmed to aid the radiologist in specific measurement tasks, such as determination of the head size in the ultrasound image of an embryo, or the segmentation and subsequent bone density measurement in vertebral regions of CT and DEXA (dual-energy x-ray absorptiometry) to diagnose osteoporosis.

14.3. EXAMPLES OF IMAGE PROCESSING PROGRAMS

In this section, some comprehensive image processing and analysis tools and some specialized visualization software tools are introduced. These software tools have been selected as representative examples of Free Software and can be downloaded from the maintainer's Web sites. In all cases, the source code is available, and the user can examine, expand, and modify the software. All examples belong in the category of menu-driven image analysis software, as this is the category with the largest number of representatives available. The examples have also been selected because they are under active development. To complement the list below, two examples of nonfree image processing programs, ANALYZE and e-Film, are given. Mayo Clinic's very comprehensive image analysis tool ANALYZE (http://mayoresearch.mayo.edu/mayo/research/robb_lab/analyze.cfm) is available for a considerable license fee. The license fee can be waived when a collaborative research proposal has been accepted by the Biomedical Imaging Resource of Mayo Clinic. The lean DICOM viewer e-Film (http://www.cedara.com/OEM_Solutions/Products/eFilmLite/index.aspx) is often provided on a CD together with image data to enable viewing the study data without having to resort to a full-featured image analysis program. The licensing schemes of these nonfree programs provide an interesting contrast to the Free Software philosophy.

AMIDE (AMIDE's A Medical Imaging Data Examiner) AMIDE is a specialized software package for medical image analysis and visualization with a particular focus on three-dimensional data sets, particularly the merging of multiple three-dimensional volume data sets. AMIDE is capable of reading a number of different medical image formats, including DICOM, ACR/NEMA, NIFTI, and Concorde PET data. AMIDE has very powerful capabilities to define two- and three-dimensional regions of interest, including freehand and isocontour regions. Multiple volume three-dimensional sets can be brought to registration by means of fiducial markers and rigid-body transformations. The fiducial markers need to be set manually for each data set. Data sets can be blended or overlaid. AMIDE has strong visualization and rendering options, with transverse, coronal, and sagittal sectioning, volume rendering (full volume with intensity-based transparency), and the generation of fly-through and time-series movies that can be exported in movie file formats. On the other hand, AMIDE has only very basic image processing and quantitative analysis capabilities. These include nearest-neighbor and trilinear interpolation, anisotropic filtering (Gaussian smoothing and median filters), thresholding, image math between data sets, three-dimensional line profiles with Gaussian curve fit, and image statistics. AMIDE can be obtained from http://amide.sourceforge.net and is available for Linux, Windows, and Mac OS.

BioImage Suite The BioImage Suite (http://www.bioimagesuite.org), maintained by Yale University, is a three- and four-dimensional image analysis tool for image processing and visualization. Functions include basic image processing (such as standard image filtering, interpolation, rotation), voxel classification (histogram-based, Markov random fields, exponential-fit methods), and deformable models (segmentation of different anatomical structures with snakelike deformable models). A particular focus lies on MR data analysis, fMRI, and image registration. Examples for unique features are MR bias field correction, specialized rigid registration schemes, specialized measurement functions for diffusion-weighted MRI, and cardiac deformation analysis. Plans have been announced to combine BioImage suite with Harvard's Slicer.

BioImageXD BioImageXD is a recent development that has a focus on microscope images.[7] BioImageXD has a strong rendering component, which is based primarily on the Visualization Toolkit (VTK) described further below. Rendering functions include false coloring, rendering of three-dimensional stacks, and creation of animated movies from two- and three- as well as four-dimensional data, that is, sequences of volumetric images taken at different points in time. This software is also very flexible when it comes to reading proprietary microscope image files, such as the image formats used by Zeiss, Olympus, and Leica. These formats include two-dimensional images and three-dimensional stacks. BioImageXD is still under development, and the image processing and analysis functions are not well developed at this time. However, for microscope image processing, BioImageXD offers several very attractive functions that are difficult to find elsewhere. Most notably, the software features

intensity correction for photobleaching and the analysis of fluorescence co-localization. Furthermore, plans for new features include deconvolution of three-dimensional image stacks, that is, the subtraction of blurred out-of-focus planes from the in-focus plane. Once implemented, deconvolution imaging can produce very high quality images from nonconfocal microscopes. BioImageXD is available in source code, and binaries for Windows and Mac OS exist, whereas Linux support is only rudimentary. BioImageXD can be accessed on the Web at http://www.bioimagexd.net.

CVIPTools (Computer Vision and Image Processing Laboratory Tools) The CVIPTools are developed at the Computer Vision and Image Processing Laboratory of Southern Illinois University at Edwardsville. Its long development history explains the unique four-layered internal structure of the software: The lowest layer (presumably also the oldest layer) is composed of a number of library programs that contain the actual image processing code. The next layer is a command-line layer to control the library modules and to combine image processing steps into a script. The third layer adds the functionality needed for a graphical user interface, and the top layer is the graphical user interface itself. With the graphical user interface, the CVIPTools become a cohesive, menu-oriented image processing and analysis package. The CVIPTools are possibly one of the most comprehensive software packages available for image analysis. The CVIPTools cover the complete basic functionality described in Chapters 2 through 4, with many enhancement and restoration filter options in the spatial and frequency domains. For image segmentation, CVIPTools provide numerous algorithms, such as histogram-based intensity thresholding, split-and-merge segmentation, fuzzy c-means clustering, and the Hough transform. Particularly noteworthy are feature extraction functions by shape and texture (see Chapters 8 and 9). In addition, CVIPTools contain several advanced image compression algorithms, including fractal and wavelet compression. In CVIPTools, many operators of similar functionality are implemented. An example is that of edge detectors, where CVIPTools features the Kirsch, pyramid, Sobel, Prewitt, Roberts, and Frei–Chen operators. The user can study the effect of these functions and study their subtle differences. A complementary book[13] is available that covers the CVIPTools operators. A CD-ROM accompanies the book, which contains CVIPTools binaries. Unfortunately, binaries are provided for Windows only. Users of other operating systems can download binaries from the CVIP Web page, http://www.ee.siue.edu/CVIPtools, which also allows download of the source code. It appears that the available non-Windows binaries are outdated and do not run on more recent operating system versions. Knowledge of the compilation process and the files that control the process (makefiles) is recommended for users who intend to compile the software. The source code is fairly well documented and can serve educational purposes even if not compiled.

IMAL (Image Measurement and Analysis Lab) IMAL is a comprehensive image analysis package developed primarily by Thomas Nelson. It was developed originally at the National Institutes of Health, where the software was known as TNImage. Development continues at the Blanchette Rockfeller Neurosciences Institute at Johns

Hopkins University. IMAL provides a very comprehensive suite of image processing functions that cover most functions in Chapters 2 and 3. IMAL also provides a number of convenient annotation and drawing functions. Furthermore, IMAL comes with a number of measurement functions. These include length, area, size, and angle, all of them with the option to calibrate the data. In addition, IMAL helps with densitometry measurements (strip and spot densitometry): for example, in the analysis of gels. The wavelet transform functions are a particular strength. In addition, IMAL provides functions for fiducial marker-based image registration. IMAL is one of the very few programs that uses higher bit depths than 8 bits/pixel when needed. As such, IMAL is particularly suited for quantitative image analysis and may surpass ImageJ in this respect, because ImageJ reduces images to 8 bits/pixel when opened, even if higher image depths have been read. IMAL comes with a full user manual. IMAL uses some UNIX-specific libraries, most notably Motif (a library for creating graphical user interfaces) and is available for flavors of UNIX and Linux only. Windows users may use one of the Unix emulators, such as Cygwin. For Macintosh users, a similar solution with the emulator MI/X applies. The main Web page for IMAL is http://www.brneurosci.org/imal.html. On this page, links make it possible to download the source code or precompiled binaries, including static binaries (i.e., self-contained binaries that have all libraries built-in). Although static binaries can be large, any problems with mismatched libraries are avoided, and the software runs on a wide variety of operating systems.

Slicer Harvard's Slicer software (http://www.slicer.org) is a software suite for registration and visualization. Slicer allows slice by slice and three-dimensional views, segmentation, false coloring, and overlay views in two and three dimensions. Slicer offers some registration functions. A unique module is the FE-Mesh module, which allows us to extract mesh information for finite-element models from image data.

VisIt VisIt is an open-source visualization package developed at the Lawrence Livermore National Laboratory. Unlike VTK (described below) and OpenDX (described in Section 14.5), VisIt comes with a graphical user interface and allows the user to control most visualization tasks from a menu. VisIt covers all functions described in Chapter 13. In addition, VisIt can plot vector fields and has a special module for rendering chemical molecules. VisIt is ideal for beginners because of its ease of use, yet it contains powerful visualization functions that probably cover all common visualization tasks. VisIt is available for Linux, Windows, and Mac OS and can be downloaded from the maintainer's Web page at https://wci.llnl.gov/codes/visit/home.html.

VTK (Visualization Toolkit) The Visualization Toolkit (VTK) is a suite of libraries for volume visualization. By itself, the toolkit provides the rendering functionality, but it does not come with a user interface. It is necessary to describe the visualization task in a programming language (either C++ or Tcl). Considerable effort is required to master VTK, and the help of a book[9] is recommended. VTK acts as a layer between

the preprocessed and segmented volume data and the computer's rendering software, OpenGL. As such, it performs many of the tasks that were introduced briefly in Sections 13.4 and 13.5. VTK is often combined with a parallel development, ITK (the Insight Toolkit), which performs the image segmentation and classification. Both programs have emerged from the Visible Human Project. The home page for VTK is http://www.vtk.org and for ITK, http://www.itk.org. Because of the complexity that VTK presents to the user, several third-party products have been developed that provide a graphical user interface and contain implementations of the most important visualization tasks. Examples include the VTK Designer by VCreate Logic and Amira by Visage Imaging.

14.4. CRYSTAL IMAGE

Crystal Image was developed and written by the present author. As with many similar software packages, initial software development was done to fill the need for a small number of highly specialized algorithms. During the initial development stages of Crystal Image in the year 2000, ImageJ was in its infancy, and NIH Image was not a feasible alternative because it was restricted to Macintosh computers. Scion Image was not open source and could therefore not be modified or extended. The decision was made to start an image processing package from scratch. Furthermore, the decision was made to use the Linux operating system for development because of the availability of a multitude of development and debugging tools and the availability of suitable libraries: GUI libraries (GTK, the Gimp Toolkit) and image display libraries (Imlib). Initial development efforts showed that software development would be faster in this environment than under Windows with Visual C++, which was the alternative at that time. Since then, Crystal Image has grown to become a comprehensive quantitative image analysis package that features most of the functionality of ImageJ, including a macro language and plugin capability. The algorithms that are provided throughout this book were implemented and tested in Crystal Image before they were converted to pseudocode for publication. In this respect, Crystal Image exceeds the functionality of ImageJ. However, Crystal Image still needs to be considered a work in process. Development efforts were focused primarily on functionality. For this reason, documentation is lacking, and Crystal Image does not have an installer. Some initial design restrictions have not been removed. Most important, at the time of inception the four image types that ImageJ supports were considered sufficient: 8-bit, 16-bit, 24-bit color, and 32-bit floating-point. A slightly inconsistent implementation of the color format makes it sometimes awkward to use, and a later need for additional formats (48-bit color, 64-bit double-precision floating point) could not be accommodated because that would necessitate changes throughout the code base. Other restrictions, such as the inability to open more than one image at a time, were later considered desirable and therefore kept (in this case to avoid the ambiguity that occurs with ImageJ when multiple images are open). Despite these development shortcomings, Crystal Image contains a number of powerful features that other image analysis programs do not offer.

Any image analysis package contains several core elements: a definition of an image object (the actual image data supplemented by metainformation such as image type and size), the ability to open image files in various formats, the ability to display such an image, and the ability to save an image or modified image. Processing and analysis functions build on this core functionality. Independently, code for a GUI is maintained and linked to the core functionality. Similar to ImageJ, Crystal Image opens a small menu window and displays the opened image in a separate window. Also similar to ImageJ, Crystal Image organizes its functions into menu items, and the menu controls the individual processing operators. In addition, Crystal Image features a macro interpreter that controls the image processing operators in an automated fashion. In the diagram of the Crystal Image internal structure shown in Figure 14.4, it can be seen how the file operators, the image processing operators (and by extension, the plugins), and the visualization tools interact with the main image structure.

The menu control of Crystal Image is similar to that of ImageJ. The `FILE` and `PROCESS` menus are comparable. The `MEASURE` menu corresponds to ImageJ's `ANALYZE` menu. Functions under ImageJ's `EDIT` menu in Crystal Image are divided more stringently between functions that modify properties of an image (`EDIT`) and functions for visualizing the image or aspects of the image (`VIEW`). Two menu items exist that have no correspondence in ImageJ: `SEGMENTATION` and `SPECIAL`.

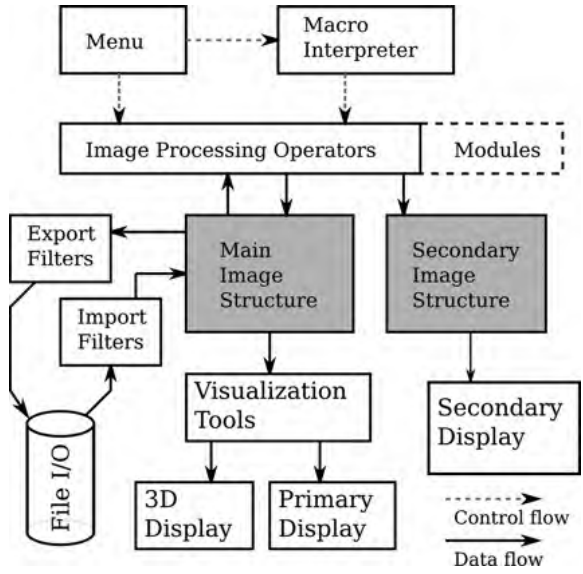


FIGURE 14.4 Internal structure of Crystal Image. Crystal image maintains one single image structure on which to operate. The operators modify this image under the control of either menu commands or the macro interpreter. An auxiliary (secondary) image with reduced functionality exists to help some interactive tasks. Both primary and secondary images can be viewed, but extensive visualization tools exist for the primary window only.

Under the SEGMENTATION menu, several different segmentation methods have been collected, including intensity-based region growing and hysteresis thresholding, snakes (active contours), watershed segmentation, and clustering. The SPECIAL menu contains several image processing operations that were included in earlier development stages, such as small bone densitometry and analysis of different CT images. Currently, the SPECIAL menu is the default location where plugins register their functions.

For the file operations, Crystal Image uses the TIFF format predominantly. The TIFF format is extremely flexible and allows reading and writing of all image types (8 bits/pixel unsigned, 16 bits/pixel signed, 32 bits/pixel floating point, and 24 bits/pixel RGB color) either as two-dimensional slices or as a stack of slices (three-dimensional images). In addition, Crystal Image can read any ImageMagick format (GIF, PNG, BMP, and other pixel formats), but these default to 24-bit RGB. Furthermore, a legacy image format specific for Crystal Image exists: the vsrt (volume short) and vflt (volume float) formats. These formats are extensions of the PBM (portable bitmap) format to 16 and 32 bits/pixel with an option to store three-dimensional image data. The advantage of this format is the simple way to create an import filter. Crystal Image can also import and export raw image formats and ASCII data and export images in OpenDX native two- and three-dimensional formats. The last function is a particularly interesting way to establish an easy and direct link to the powerful OpenDX 3D visualization system. However, Crystal Image comes with a its own flexible, yet easy-to-use visualization tools: Any image can be displayed with modified brightness/contrast/gamma, false-colored, provided with isocontour lines, and displayed as a three-dimensional elevation surface.

There is a large overlap between the operators that ImageJ and Crystal Image provide. Furthermore, Crystal Image provides numerous advanced image analysis operators that are presented throughout this book and that are not available in ImageJ. Some highlights of Crystal Image operators covered in this book are multiple functions for background removal, adaptive filters (local adaptive lowpass filter, tri-state median filter, anisotropic diffusion filter, and the adaptive bilateral filter), several fractal dimension operators (local Hoelder exponent and a number of estimators for the global fractal dimension, such as the box-counting method, Minkowsky method, mass dimension, blanket method, dispersion method, and the frequency-domain rose plot), the measurement of run lengths, and a wavelet module that offers the wavelet transform and wavelet-based denoising. Shape and texture analysis functions are implemented, the latter based on the co-occurrence matrix.

Image math functions are implemented as well, but the restriction that only one image can be opened at a time had to be overcome. For this purpose, the image math menu offers a push function that stores the present image in the background (a one-level stack) and then makes it possible to perform dyadic operations on the currently opened image with the previously pushed image. A similar idea was used to implement the undo function. Whereas ImageJ does not offer a consistent undo function, Crystal Image offers an undo operation for every operation that changes the main image. There are two user-selectable undo options: a fast one-level undo in memory (single-level undo stack) and a 10-level undo, where the undo stack is kept on the hard disk.

The macro language built into Crystal Image differs fundamentally from ImageJ and is directed more toward the scripting languages of Matlab and Octave. The model language was BASIC (beginner's all-purpose symbolic instruction code), and the macro language has been nicknamed ImageBasic. The fundamental language element is the assignment

$$a = \mathrm{f}(b, c, d, \ldots); \tag{14.1}$$

where the variables a, b, c, d, ... may be numerical values, strings, or images. In many cases, f is an operator, b is the image to operate on, and c, d, ... are parameters for the operator. An example of such an expression is

$$i2 = \text{hysteresis\_threshold}\,(i1, t, t + 50); \tag{14.2}$$

This is a statement that executes the hysteresis threshold operator with image *i1* and the lower and upper thresholds t and *t+50*, respectively. The result—the segmented image—is stored in the variable *i2*. It can be seen that the arguments (in this example, *t+50*) may be mathematical expressions. The input image itself may also be an expression such as *i1*mask* (multiplication of two images) or *img-offs* (subtraction of images). In addition, flow control statements are available: *if-else*, the *for* loop, the *while* and *repeat-until* loops, and function calls, albeit without formal parameters. This means that all variables are global. A particularly powerful feature is the *foreach* loop, which allows execution of the loop for all files in a directory that matches a specified file mask.

One additional mode exists, the immediate execution mode. In this mode, macro commands can be entered and are executed interactively. This is a convenient method of accessing macro functions and automating menu functions. Short loops are possible. In addition, the commands are logged and can be saved to a text file, where they can be converted to a full macro with minimum effort.

An example macro that highlights some of the features and provides an example of the syntax is given in Algorithm 14.1, which shows most of the elements of a macro. Macro files are plain text files. Each macro starts with the keyword `macro`, followed by the macro name. The macro name is the name of the menu item that is added to the `MACROS` menu of Crystal Image. The new menu entry allows to activate the macro. The symbol # indicates a comment. All text from a # symbol to the end of the line is ignored by the macro interpreter. In the next two lines, two images are loaded and stored in the variables a and b. In this example, a noise-free image and a noisy image of the same object are loaded for comparison. The macro tests how effectively the adaptive tristate median filter can remove the noise. In the first section, the mean-squared distance between images a and b is computed; then the image b is subjected to a conventional median filter (result stored in c), and the distance to a is computed and printed. In the second half, the adaptive tristate median filter is applied within a `for` loop that steps the threshold from 0 to 255 in increments of 5. The mean-squared distance is computed for each threshold and stored in an array.

```
macro tsmf_analysis:

   a = load ("liver_clean.tif");        # unspoiled original
   b = load ("liver_shotnoise.tif");    # with 10% shot noise
   display (b);                         # Show b in main window

   nerr = (a-b)/a;                  # normalized error per pixel
   x = measure (nerr*nerr,0);       # mean-squared error
   print ("Mean-squared distance of the originals: ",x);
   c = morph (b,"median",8);        # the other extreme
   nerr = (a-c)/a;                  # normalized error per pixel
   x = measure (nerr*nerr,0);       # mean-squared error
   print ("Mean-squared distance to rank-filtered: ",x);

   # Now examine the thresholds in the TSMF

   dim (xydata[55,2]); # array storage

   i=0;
   for (T=0,255,5); # FOR loop from 0 to 255 in five increments
      c = convolve (b,"TSM",T);     # Apply the TSM filter
      nerr = (a-c)/a;               # normalized error per pixel
      x = measure (nerr*nerr,0);    # mean-squared error
      print ("Threshold ",T," Mean-squared distance ",x);
      xydata[i,0]=T;
      xydata[i,1]=x;         # Store the t-x pair for graphing
      i = i+1;
   endfor;

   graph (i, xydata);        # Display the data points x over t

endmacro;
```

Algorithm 14.1 Example of a Crystal Image macro. This example analyzes the effect of the adaptive tristate median filter with various thresholds.

After the loop is finished, the distance values are plotted over the threshold values. A screen shot of the macro run is shown in Figure 14.5.

Analogous to ImageJ, the option to create plugin code (called *modules*) exists. Crystal Image modules need to be written in C, the native language of Crystal Image. A given structure needs to be followed. Modules are compiled into dynamically shared objects (DSOs) and loaded by Crystal Image upon startup. Module development requires advanced knowledge of the C programming language. To facilitate module development, a module template (file `0module.c`) is provided that contains the necessary module infrastructure and extensive comments to explain how a module works.

Crystal Image can be run from the accompanying DVD in three forms, either from the live DVD, by installing the kubuntu Linux operating system from the DVD, or

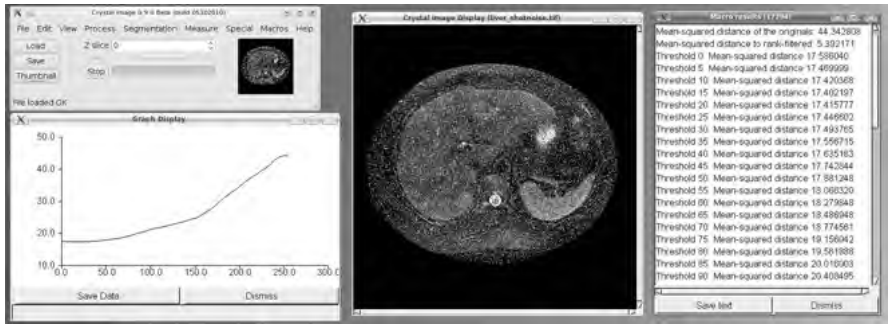


FIGURE 14.5 Example of a Crystal Image macro run. The macro is loaded and started from the macro menu in the control window (top left). Upon execution of the fourth line of Algorithm 14.1, the display window is opened. At the end of the macro run, the text window with the output of the `print` statements (right) and the graph display windows are opened. Both the values in the text window and the graph display show a mean-squared distance minimum at a threshold of 15, but this minimum is markedly higher than the distance of 5.4 between the median-filtered image and the original noisy image.

by compiling Crystal Image from the source code. In the latter case, a fully installed Linux computer must exist. Alternatively, it should be possible to install Crystal Image under Windows if an emulator such as Cygwin is installed (this is similar to IMAL). Instructions are provided in Appendix A.

14.5. OpenDX

OpenDX is an extremely powerful data visualization program, and arguably, the visualization tool with the best combination between user-friendly operation and flexibility. The software, known as the Visualization Data Explorer (DX), was originally developed by IBM as a commercial visualization product. In 1999, IBM released DX under an open-source license to "attract the creativity of the world's developer community [and to send] a strong message to both the technical and business community about IBM's commitment to open and non-proprietary standards."[1] Development on OpenDX has continued since then, and OpenDX has become a standard software in visualization. The OpenDX software, including source code, documentation, and samples, can be downloaded freely from the OpenDX Web site, http://www.opendx.org.

OpenDX is based on the principle of visual programming, similar to Khoros. A visual program is developed by placing operators on a canvas and connecting them with lines that represent data flow. A number of information elements need to be collected to obtain an object that can be rendered. The most important information elements are a data field (usually, an array of data) and color. The visual program in Figure 14.6 shows how operators collect information, starting with a file name, then the image itself. Additional operators add color, create the elevation landscape,

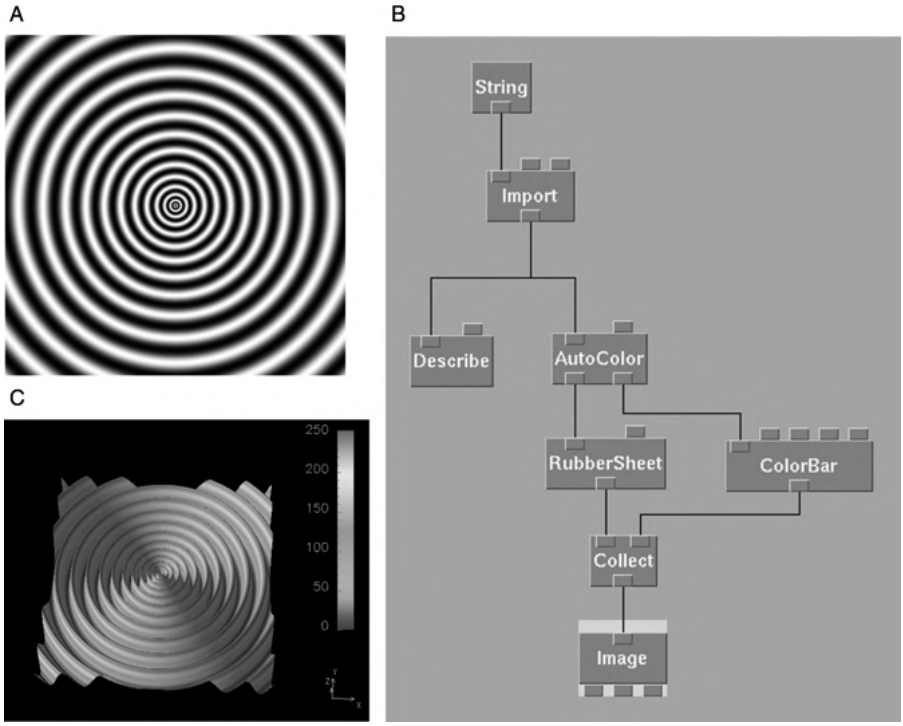


FIGURE 14.6 Example of an OpenDX visual program. The input image (A) exists in native DX format as created by Crystal Image. In the visual program (B), the `String` specifies the file name, and `Import` operator converts the image into a field. With `AutoColor`, color is being added to the nodes of the field. `Rubbersheet` creates the elevation map. Finally, the `Image` operator renders the object (C). The `ColorBar` operator is responsible for adding the color scale bar to image C, and the `Describe` operator describes the data object at the point where it is connected. (*See insert for color representation of the figure.*)

and finally, render the image. In addition, a color scale bar is added, and an operator was included that describes the data and provides helpful hints on how to render the data.

The design of a visual program is not straightforward, and users new to OpenDX might easily get confused by the number of operators available. Fortunately, OpenDX offers an "autopilot" mode where a default visual program is generated that can cover most standard visualization applications. With the `IMPORT DATA` function from the initial menu, this automatic mode is invoked. A Data Prompter window opens in which the data can be described. Once the data are described satisfactorily (format, such as image file or scattered data file; array dimensions and value range; data type), the click of a button creates a default visual program and renders the data. The visual programs created in this mode are quite complex, because they distinguish between two-dimensional scalar data (which are rendered as an elevation landscape),

three-dimensional scalar data (which are segmented using an isosurface and rendered as a three-dimensional object), and vector data (two or three dimensions, which are rendered as an array of glyphs). The visual program depends strongly on the type of input data. If the input data are an image, the visual program simply displays the image in a two-dimensional display window. In the case of a spreadsheet format (four columns: x, y, z, and value), a visual program is created that shows a cube in which glyphs are placed that are proportional in size to their value. In all cases, the visual program can be opened and examined, and this default visual program is helpful in learning how to create a custom OpenDX visual program.

Figure 14.6 shows one sample visualization: a false-colored elevation landscape rendering of a gray-scale image. Two more examples will provide a more general idea of how a visual program is created in OpenDX. The first example is a three-dimensional rendering of a segmented object such as the mouse femur used in Figure 13.8. In CT images, compact bone is easily segmented because of its high CT value. Three-dimensional volume rendering is possible whenever the image values of the object are distinct from the background image value (i.e., when segmentation by thresholding with a global threshold is possible). If this prerequisite is not met, the image needs to be thresholded by prior image processing. Once separation by image values is possible, isosurface extraction becomes possible. In this case, the segmented object is rendered as a solid represented by thin walls (i.e., its isosurface). The visual program and two sample renderings are shown in Figure 14.7. The sample object has two inhomogeneities. One is a section of reduced thickness which becomes clearly visible. The second is a section of reduced image intensity. A high isosurface value removes this section from the object and creates the impression of a split object. By reducing opacity in the color module it can be demonstrated that the rendered surface is indeed a thin wall. Furthermore, the first and last slices of the three-dimensional image are all background, so that the tube wall appears closed. Because of its surface-based rendering principle, OpenDX is not capable of rendering solid objects constructed from semiopaque cubes. The closest approximation is the rendering of a series of isosurfaces of the same object.

Another example visualizes a different data structure: a vector field. In such a case, glyphs are used to visualize the direction and magnitude of the vector field. OpenDX offers numerous glyphs, but the natural choice for the visualization of a vector field are small arrows. The example in Figure 14.8 visualizes fluid flow in a flow chamber. The flow velocity is represented by arrows, and the chamber geometry is merged with the image as an isosurface. To have a nontransparent back and a transparent front for the flow chamber, two images of the flow chamber outline were created, and the second (opaque) flow chamber was clipped.

The capabilities of OpenDX go vastly beyond these examples. OpenDX contains operators for mathematical operations, vector operations (such as divergence and curl of a vector field), calculation of input statistics and histograms, and interactive elements such as sliders and buttons. In addition, the visual programs may contain loop elements, which can be used to create animations: for example, by moving the camera with respect to the scene. OpenDX has a large number of import functions. Raw data or ASCII data can be described with the help of a `general` file, which

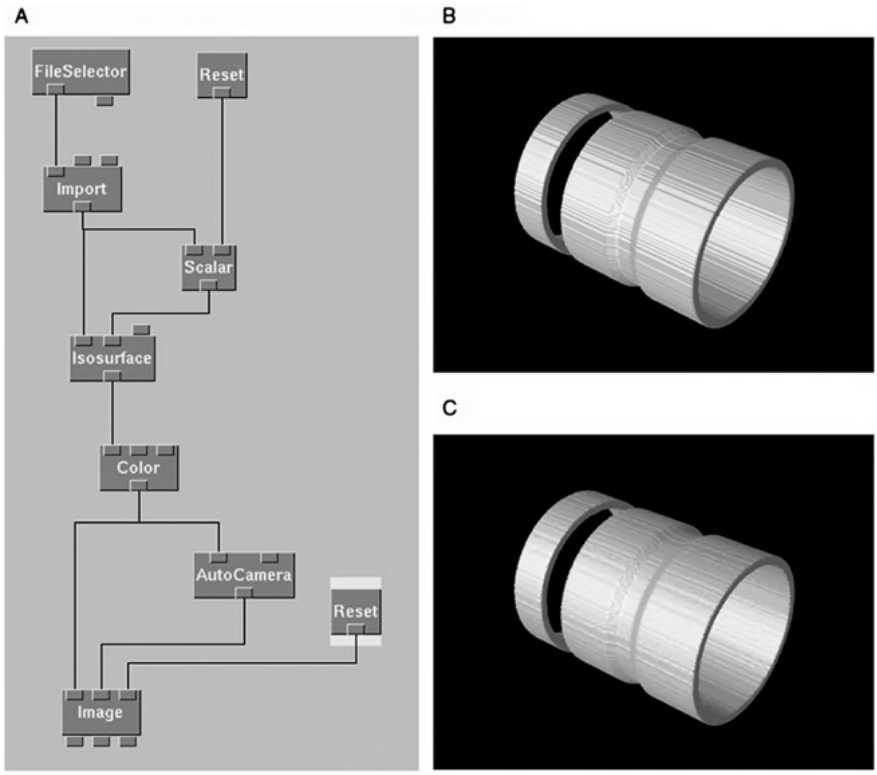


FIGURE 14.7 OpenDX visual program to render the isosurface of a three-dimensional object as a solid. Similar to Figure 14.6, the `Import` operator is used to read the file, but in this case the interactive `FileSelector` provides the file name string. The `Isosurface` operator extracts the isosurface, and the image value is provided by the `Scalar` operator. The Color operator adds color (in this case, 75% gray), and the `Image` operator renders the object. In extension to Figure 14.6, an `AutoCamera` module is provided, which replaces the built-in camera of the `Image` operator and provides additional flexibility. The Image operator renders image B. Bandlike structures that result from the pixel discretization are visible. By introducing a `SimplifySurface` operator between `Isosurface` and `Color` (not shown in the example visual progam), the surface can be smoothed (C).

is a text file that contains the detailed description of the data. A `general` file is created automatically when data have been described interactively with the import function.

The examples and lists in this chapter are not exhaustive, but they provide an idea of the capabilities of OpenDX. The examples also show that visual programs may gain in complexity very rapidly. The best way to gain an understanding of OpenDX visual programs is either to use a tutorial, such as the book *OpenDX: Paths to Visualization*,[12] or to peruse one of the many tutorial samples that are provided with OpenDX.

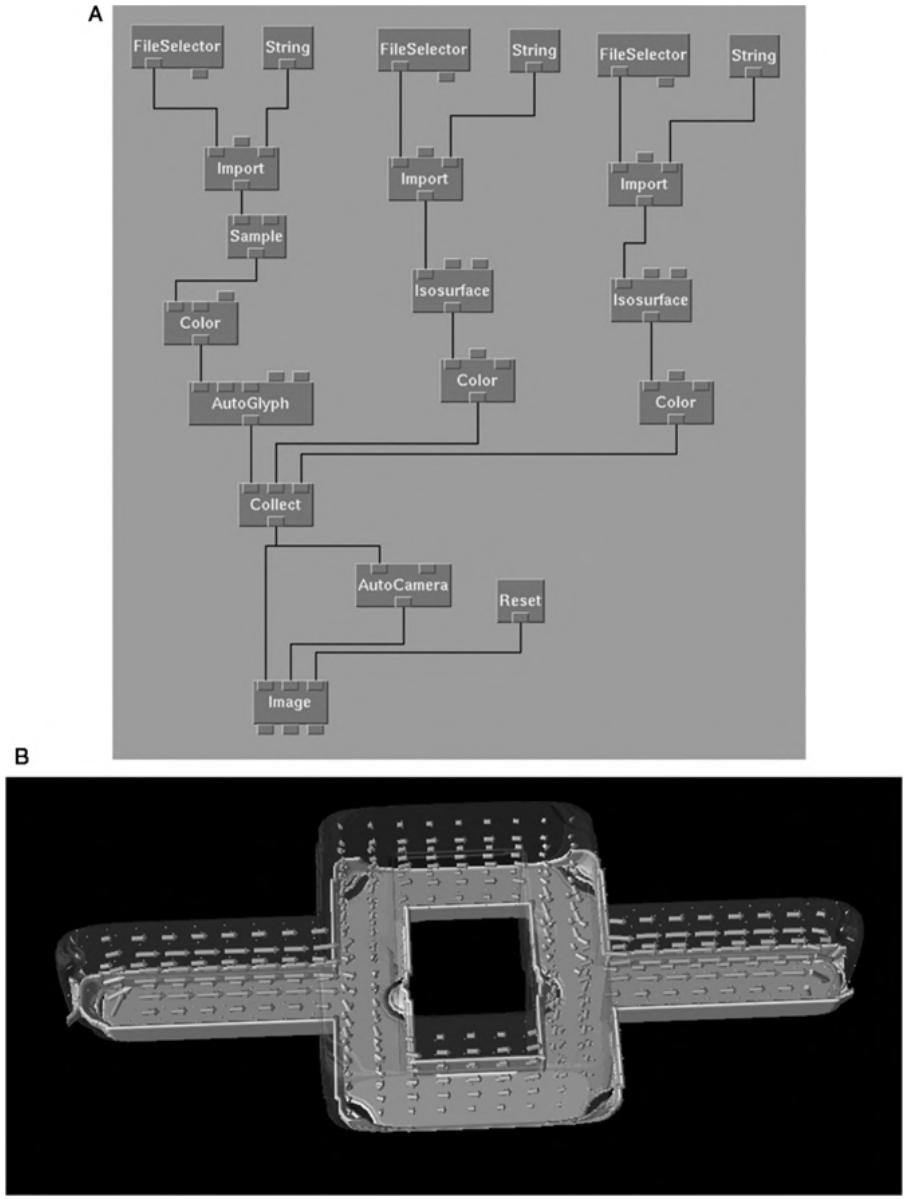


FIGURE 14.8 Visualization of a vector field. The visual program (A) is relatively complex because it merges three components: the vector field itself, visualized with the `AutoGlyphs` operator, and two scalar fields that contain the outline of the flow chamber, visualized as isosurfaces. The lower isosurface (about one-third of the chamber) was made nontransparent, whereas the upper part of the flow chamber was made semitransparent. (*See insert for color representation of the figure*.)

14.6. WAVELET-RELATED SOFTWARE

Numerous software packages for the wavelet transform and particularly for wavelet-based denoising can be found on the Web for free download. Most frequently, these software packages are collections of Matlab functions. Most of these functions will also work with Octave (Section 14.1.2). The arguably most comprehensive collection of functions is WaveLAB by David Donoho, which contains Matlab functions for orthogonal and biorthogonal wavelet transforms, translation-invariant wavelets, interpolating wavelet transforms, cosine and wavelet packets, and various denoising algorithms. The URL for WaveLAB is http://www-stat.stanford.edu/∼wavelab/. Denoising algorithms presented by Portilla et al.[8] can be found at http://www.io.csic.es/PagsPers/JPortilla/denoise/software/index.htm. Shihua Cai and Keyong Li have provided a collection of one- to three-dimensional wavelet transforms and denoising algorithms at http://taco.poly.edu/WaveletSoftware/index.html.

Wavelet transform functions are also available in C, C++, and Java. For example, Ian Kaplan provides several elementary functions in Java and C++, at http://www.bearcave.com/software/java/wavelets/index.html. Source code for a wavelet-based image compression package provided by Geoff Davis and written in C++ is available at http://www.geoffdavis.net/dartmouth/wavelet/wavelet.html. The digital signal processing group at Rice University provides a one- and two-dimensional wavelet processing toolbox at http://www-dsp.rice.edu/software/RWT/. Last but not least, an Internet forum exists at http://www.wavelet.org, where anybody involved in the design and application of wavelet software can discuss problems and provide answers.

14.7. ALGORITHM IMPLEMENTATION

Advanced image analysis often needs custom algorithms or custom modifications of existing algorithms. Existing image analysis programs offer many options with their scripting or macro languages. However, specialized algorithms may quickly go beyond the capabilities of these languages. In such a case it is best to implement the algorithm in a general-purpose programming language, such as Java, C, C++, or one of the many other languages that are available. Most algorithms that are provided in this book cannot be implemented readily within the constraints of a scripting or macro language. There are three general approaches to algorithm implementation with general-purpose programming languages:

1. Extension of existing image analysis programs (e.g., ImageJ plugin or Crystal Image module). In-depth knowledge of the inner workings of the host program is required, but the host program may already contain numerous functions that can be used by the algorithm.
2. Stand-alone processing utilities. These may be short and specific, and only minimal overhead may be required to read the input file and write the output file. Stand-alone utilities are often realized as command-line tools without a

graphical user interface. Scripting programs (Matlab, Octave) would fit in this category. With this option, goals may be achieved in the shortest time.

3. Integrated processing utilities, often with a graphical user interface (GUI). Although the overhead for this solution increases strongly over that of the second solution (GUI-based programs can easily reach and exceed an overhead of 50% of the code to provide user interaction), the developers have the maximum flexibility. In fact, the developers of BioImageXD state that it is often easier in the long run to begin development from scratch. Crystal Image has been created for the same reason.

Sometimes one development model may lead to another. An example is the CVIPTools suite, which began as a collection of stand-alone image processing tools and was later integrated into a cohesive graphical environment. Finally, it is possible to use existing image processing libraries. One popular example is the OpenCV library (http://opencv.willowgarage.com/wiki/, http://sourceforge.net/projects/opencvlibrary/) with about 500 library functions for image processing and computer vision. A book exists that introduces programming with OpenCV.[2] Libraries such as OpenCV combine a moderate initial learning effort with the advantage of flexible and efficient programming and make use of libraries that are generally highly optimized. It depends on the developers and the goals, which is the most suitable of the options listed. In this section we illuminate the fastest way to reach a working algorithm: the stand-alone module, aided by an existing image analysis program.

An image processing operator requires one or more image files as input, may require parameters to control the action of the operator, and usually outputs an image that is the result of the operation. In a stand-alone program, input and output images are typically disk files. Ideally, the program would have the ability to read and write image files. Image files contain the image data together with additional information, such as width and height, and the pixel interpretation. Libraries exist that help read and write an image, but the complexity of most image formats is high, and the import filter (e.g., for TIFF or DICOM images) can gain appreciable complexity. For the rapid development of a stand-alone algorithm, a raw format has the advantage of being independent of image libraries. Raw formats contain the image data without additional information, and the data can be represented either as ASCII numbers or as binary data. Two simple functions can handle raw data input and output, in the example of Algorithm 14.2 written in C. The functions are restricted to a specific data interpretation: in this example, 32-bit/pixel floating-point numbers.

The next step in the development process would be the creation of a program frame that integrates multiple functions (including the read and write function in Algorithm 14.2) into an entity that can be executed on a computer. In C, a function called `main` defines the entry point. The function `main` receives command-line arguments, and three elegant macros can be used to read and interpret command-line arguments. A possible realization of `main` is shown in Algorithm 14.3.

```
int read_raw (const char *fname, float *data, int x, int y, int z)
{
   FILE *F;                                  // input file handle
   int elems;

      data = malloc (x*y*z*sizeof (float)); // Allocate memory
      if (!data) return -1;                  // Exit if operation
                                             failed
      F = fopen (fname, "rb");               // Open file for read
      elems = fread (data, x*y*z, sizeof (float), F);
      fclose (F);
      if (elems!=x*y*z)
            return -1;                       // size mismatch error
      return 0;                              // success
}

int write_raw (const char *fname, float *data, int x, int y, int z)
{
   FILE *F;                                  // output file handle
   int elems;

      F = fopen (fname, "wb");               // Open file for write
      elems = fwrite (data, x*y*z, sizeof (float), F);
      fclose (F);
      if (elems!=x*y*z)
            return -1;                       // size mismatch error
      return 0;                              // success
}
```

Algorithm 14.2 Example of functions to read and write an image file in raw format. The functions receive the parameters for the image extent in the x, y, and z directions and a pointer to raw data storage in memory. Furthermore, the file name is given in `fname`. While `read_raw` makes sure that memory for the storage of image data is available, `write_raw` requires that data storage has been allocated properly: for example, by `read_raw`.

The only missing elements are some library definitions and the command-line handler definitions. The program can be typed into a text editor, starting with the following lines:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

#define nextargi (--argc,atoi(*++argv))
#define nextargf (--argc,atof(*++argv))
#define nextargs (--argc,*++argv)
```

```
int main (int argc, char **argv)       // argv are the command-line
                                       arguments
{
   int imgx, imgy, imgz;               // image dimensions
   float *data;                        // image storage
   float *target;                      // processed image storage
   char infname[1024], outfname[1024]; // file names for input and
                                       output
   float p1, p2;                       // two parameters
      if (argc<8)                       // Too few command-line
                                       arguments?
      {
           printf ("Command-line usage:\n");
           printf ("  %s (inf)(outf)(x)(y)(z)(p1)(p2)\n",argv[0]);
           printf ("  (inf) is the input file name\n");
           printf ("  (outf) is the output file name\n");
           printf ("  (x), (y), (z) are the image dimensions\n");
           printf ("  (p1) and (p2) are parameters\n");
           exit (0);
      }

      strcpy (infname,nextargs);       // Read input file name
      strcpy (outfname,nextargs);      // Read output file name
      imgx = nextargi;                 // Read image dimensions
      imgy = nextargi; imgz = nextargi;
      p1 = nextargf;
      p2 = nextargf;

      printf ("Reading image %s with dimensions %d, %d, %d\n",\
                  infname,imgx,imgy,imgz);
      if (read_raw (infname, data, imgx, imgy, imgz))
      {
            printf ("Error reading file\n"); exit (1);
      }

      target = malloc (imgx*imgy*imgz*sizeof (float));
      if (!target) exit (1);    // Memory allocation failed? Exit
                                immediately

      // The call to the image processing function goes here
      // We'll just copy the data for now

      memcpy (target, data, imgx*imgy*imgz*sizeof(float));

      if (write_raw (outfname, target, imgx, imgy, imgz))
      {
            printf ("Error writing file\n"); exit (1);
      }

      printf ("Program completed successfully\n");
      exit (0);
}
```

Algorithm 14.3 The function `main` provides the frame for execution of a program.

Next, the definitions for `read_raw` and `write_raw` are inserted, and finally, the function `main` follows. The completed file, call it `test.c`, can now be compiled (translated into machine language:

```
cc -Wall -g -o test test.c
```

where `cc` is the C compiler command with instructions to compile `test.c` into an executable binary file `test`, showing detailed warnings (`-Wall`) and providing debugging symbols (`-g`). The command

```
./test
```

should execute the new program, causing it to print a short message.

These instructions work for Linux and for the command-line terminal in Mac OS. Windows users should install Cygwin. Alternatively, Windows users can use Visual C++ and create a graphical user interface that acts as the equivalent of the command-line arguments, but the complexity is considerably higher.

The file `test.c` and a suitable raw file (`dot_blot.raw`) are provided on the accompanying DVD. Running `test` with the command-line parameters

```
./test dot_blot.raw output.raw 513 282 1 30 40
```

will create a new image, `output.raw`, that is the exact duplicate of the original image. At this time, the parameters `p1` and `p2` with command-line values of 30 and 40, respectively, are not used.

To complete the example, we will now implement an algorithm from this book: Algorithm 2.5 for hysteresis thresholding. Hysteresis thresholding requires two parameters, the lower and upper thresholds. We can use `p1` and `p2` as defined in Algorithm 14.3. Hysteresis thresholding takes place in two steps: (1) thresholding with the high threshold `p2`, and (2) region growing with the low threshold `p1`. To implement these steps, let us copy the previous example, `test.c` to a new file, `hthresh.c`, and modify the new file. First, remove the `memcpy` statement in Algorithm 14.3 and replace the entire line by

```
threshold (data, &target, imgx, imgy, imgz, p2);
```

This new function, `threshold`, is explained in Algorithm 14.4, which demonstrates how the pseudocode used in this book (in this case, Algorithm 2.5) translates into an actual programming language. The handling of data arrays deserves particular attention because it differs strongly between programming languages. In this case, a linear C array is used and its elements addressed by resolving the three dimensions into one linear address.

```
void threshold (float *data, float **target, int xm, int ym, int zm, float T)
{
   int x,y,z;

      for (z=0; z<zm; z++)
            for (y=0; y<ym; y++)
                  for (x=0; x<xm; x++)
                  {
                        if ( data[x+xm*(y+ym*z)] > T)
                              (*target)[x+xm*(y+ym*z)]=1;
                        else
                              (*target)[x+xm*(y+ym*z)]=0;
                  }
}
```

Algorithm 14.4 The first part of hysteresis thresholding, which creates the seed points. This algorithm thresholds the image passed through `data` with `T` and returns the result in `target`. This algorithm is the C implementation of Algorithm 2.5.

After inserting the function `threshold` and including a call to `threshold` in `main`, it may be worth compiling the code and testing whether the first part of hysteresis thresholding works. In fact, testing small sections of code saves development time in the long run, as bugs can be found more easily in small chunks of program code. Once the small section of new code is tested and validated, the developer can rely on it. After compiling it, the new code can be tested with

```
./hthresh dot_blot.raw output.raw 513 282 1 65 100
```

In the stand-alone strategy of programming, the output data are not visible. Here the help of image analysis programs comes in handy. In Crystal Image, the `FILE->IMPORT` function can be used to load the file `output.raw` by providing its dimensions, 512, 282, and 1, in the import dialog box. In addition, the data type of IEEE float gray scale needs to be selected. In ImageJ the `File->Import->Raw` function works in an identical fashion. The image should show white blobs on a black background.

At this point the region-growing part of the hysteresis thresholding operation is still missing and needs to be added to the code. Region growing is presented in pseudocode in Algorithm 2.4, and its translation into C is given in Algorithm 14.5. There are two notable differences between Algorithms 2.4 and 14.5. Whereas most of the algorithms in this book are designed for two-dimensional images, this example handles hysteresis thresholding in three dimensions (e.g., confocal image stacks). In three dimensions, connectivity can be defined by 6 or 26 neighbors. For simplicity, the 6-neighborhood was chosen. Since the testing of the six neighbors within a triple loop that spans all image pixels may lead to an unwieldy construct of if-statements to prevent memory access outside the image bounds, the neighborhood testing was moved into a separate function, `has_neighbor`. One additional advantage of this approach is easy extension toward a 26-neighborhood. To activate the new function,

```
int has_neighbor (float *target, int x, int y, int z, int xm, int ym, int zm)
{
    if (target[x+xm*(y+ym*z)]>0) return 0;
    if ( (x>0) && (target[x-1+xm*(y+ym*z)]>0) ) return 1;
    if ( (x<xm-1) && (target[x+1+xm*(y+ym*z)]>0) ) return 1;
    if ( (y>0) && (target[x+xm*(y-1+ym*z)]>0) ) return 1;
    if ( (y<ym-1) && (target[x+xm*(y+1+ym*z)]>0) ) return 1;
    if ( (z>0) && (target[x+xm*(y+ym*(z-1))]>0) ) return 1;
    if ( (z<zm-1) && (target[x+xm*(y+ym*(z+1))]>0) ) return 1;
    return 0;
}

void regiongrw (float *data, float **target, int xm, int ym, int zm, float T)
{
   int x,y,z, iter, grown;

    iter = 0;
    do
    {
        grown=0; iter++;

        for (z=0; z<zm; z++)
            for (y=0; y<ym; y++)
               for (x=0; x<xm; x++)
               {
                   if ((data[x+xm*(y+ym*z)] > T)
                       && has_neighbor (*target, x,y,z,xm,ym,zm) )
                   {
                       (*target)[x+xm*(y+ym*z)]=1;
                       grown++;
                   }
               }
        printf ("Region growing: Iteration %d, %d points grown\n",iter,grown);
   }
   while (grown>0);
}
```

Algorithm 14.5 The second part of hysteresis thresholding, which performs region growing with the lower threshold `T`. This algorithm needs the original data in `data` and the seed points in `target`. It returns the result in `target`. This algorithm is the C implementation of Algorithm 2.4.

it needs to be executed right after the thresholding part, and the following line needs to be inserted after the `threshold` function in `main`:

```
regiongrw (data, &target, imgx, imgy, imgz, p1);
```

Once the two functions from Algorithm 14.5 are included in the file `hthresh.c` and the file has been compiled, it can be tested again. The result should clearly differ from the intermediate test, where only high-level thresholding was performed. Figure 14.9 shows an example image run with the algorithm described in this section.

The code introduced above constitutes the lowest effort to program an algorithm in a programming language. In other languages, such as C++, Java, or Python,

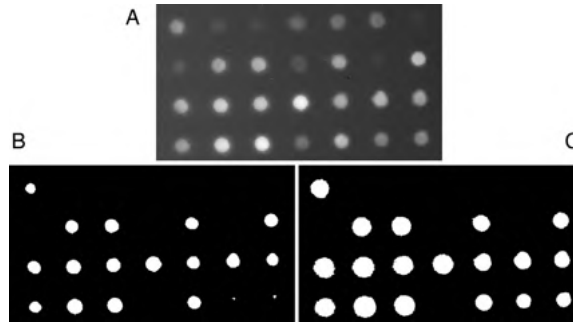


FIGURE 14.9 Images for the hysteresis thresholding example. The original gray-scale image A is an inverted image of gel blobs. Simple thresholding with $T = 100$ leads to binary image B, and hysteresis thresholding with $T_1 = 60$ and $T_2 = 100$ yields image C.

the approach is similar. The code could easily be extended to read and write an image format that includes the dimensions. Such a format is the PBM (portable bitmap) format. The natural extension is to include more information with the image structure. The definition of an image could, for example, contain the dimensions and the bits per pixel. A suitable structure definition in C would be

```
typedef struct
{
      void *data;             // pointer to image data
      int xmax, ymax, zmax;   // image dimensions
      unsigned char bpp;      // Bits per pixel
}
IMAGE;
```

While function calls become easier with such a structure, the additional information needs to be populated and maintained. In particular, if the algorithm allows more than one data format, the complexity grows rapidly. As complexity grows, some preliminary planning (i.e., software engineering) provides high payoffs in the long run. It is important that the scope of the project be known in advance and that a balance be found between fast implementation (for small projects with limited scope) and initial planning (for long-term projects with larger scope). With the known scope of the project, it is also important to weigh carefully the three options: integration into an existing project (e.g., an ImageJ plugin), a quickly developed stand-alone utility, or a more comprehensive package.

REFERENCES

1. Anonymous. OpenDX. http://www.opendx.org/about.html. Accessed July 1, 2009.
2. Bradski G, Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol, CA: O'Reilly, 2008.

3. Burger W, Burge MJ. Digital Image Processing: An Algorithmic Introduction Using Java. New York: Springer-Verlag, 2008.
4. European Patent Office BoA. Case number T 0258/03-3.5.1. http://legal.european-patent-officeorg/dg3/pdf/t030258ex.pdf, 2004. Accessed June 10, 2009.
5. González AG. The software patent debate. J. Intell Property Law Pract 2006; 1: 196–206.
6. Haidekker MA, Stevens HY, Frangos JA. Computerized methods for x-ray-based small bone densitometry. Comput Methods Prog Biomed 2004; 73(1): 35–42.
7. Kankaanpää P, Pahajoki K, Marjomäki V, Heino J, White D. BioImageXD: new open source free software for the processing, analysis and visualization of multidimensional microscopic images. Microsc Today 2006; 14(3): 12–16.
8. Portilla J, Strela V, Wainwright M, Simoncelli EP. Image denoising using scale mixtures of gaussians in the wavelet domain. IEEE Trans Image Process 2003; 12: 1338–1351.
9. Schroeder W, Martin K, Lorensen B. The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Clifton Park, NY: Kitware, Inc. Publishers, 2009.
10. Stallman RM. Free Software, Free Society. Boston: GNU Press, 2002.
11. Sternberg S. Biomedical image processing. Computer 1983; 16(1): 22–34.
12. Thompson D, Braun J, Ford R. OpenDX: Paths to Visualization. Missoula, MT: Visualization and Imagery Solutions, Inc., 2001.
13. Umbaugh SE. Computer Imaging: Digital Image Analysis and Processing. Boca Raton, FL: Taylor & Francis, 2005.

APPENDIX A

IMAGE ANALYSIS WITH CRYSTAL IMAGE

DISCLAIMER: CRYSTAL IMAGE COMES WITH ABSOLUTELY NO WARRANTY.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN

Advanced Biomedical Image Analysis, By Mark A. Haidekker

IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Installation

Crystal Image can be run from the accompanying DVD in three forms.

First, the DVD can be booted live. In this case, the kubuntu-Linux operating system runs independent from the hard disk and any operating system installed on the hard drives. Crystal Image is temporarily available within the Live DVD desktop and can be used instantly without modifications to the computer. When kubuntu is run from the live DVD, it is relatively slow because of the access times of a DVD drive, which are much longer than those of a hard drive. Once the computer is rebooted from its hard disk, it returns to the original operating system.

Second, kubuntu Linux can be installed directly from the DVD. Kubuntu is one of the ubuntu distributions maintained by Canonical Ltd. and is accessible through the home pages www.ubuntu.com or www.kubuntu.org. When kubuntu is installed from the DVD, the hard disk will be modified, and data may get lost. Ideally, kubuntu should be installed on an empty hard drive. After installation, the computer runs a full version of kubuntu with all its features and strengths, and the installation will include a functional copy of Crystal Image.

The third option is to compile Crystal Image form the source code. In this case, a fully installed Linux computer must exist. (Alternatively, it should be possible to install Crystal Image under Windows if an emulator such as Cygwin is installed.) This is the preferred solution for advanced users who wish to modify or extend the source code. Users who prefer to install the software from the source code tree need to compile the software. Compilation from source allows the software to be installed on different systems, such as newer versions of GNU/Linux, and possibly under Mac OS and Windows; in the latter case, an X window emulator such as Cygwin is required. In addition, the complete development system (gcc, the GNU C compiler, make, automake, and optionally the debugger gdb) needs to be installed. The Crystal Image source code can be downloaded from http://haidekker.org/cimage/.

In preparation for compiling the source code, all required libraries and their corresponding header files need to be installed. Your distribution should offer those (free) libraries in their repositories. If any of these libraries is unavailable in binary form, they need to be installed from source. Required libraries are:

1. libgtk2.0 and libgtk2.0-dev, the GTK library, version 1.2
2. imagemagick, an image conversion program (optional, but makes it possible to access important image formats)
3. libgif4 and libgif4-dev, the library for the free version of the GIF image format
4. libtiff4 and libtiff4-dev, the library for the TIFF image format
5. fftw3 and fftw3-dev, the FFTW library version 3.0

6. xserver-xorg-dev and libxmu-dev, X window development libraries needed
7. libgl1-mesa-dev and libglut-dev, libraries related to the OpenGL rendering system. These are optional and required only for the three-dimensional elevation landscape viewer
8. gtkglext-1.0 and gtkglext-1.0-dev, libraries that allow to render OppenGL scenes in GTK programs. These are optional and required only for the three-dimensional elevation landscape viewer

Once all libraries and the library header files (indicated by -dev) are installed, Crystal Image can be compiled. In the Crystal Image base directory, type

`./autogen.sh`	This configures the software for your system
`make`	This compiles the entire package

In the subdirectory `src`, you should now find an executable file `cimage`, and in the subdirectory `modules` should be several files that end with `dso`. Finally, the install shell script needs to be executed with root privileges:

`sudo ./install`	This installs the files in /usr/local/cimage

At this time, `cimage` can be started from a shell console, or a desktop shortcut can be created. It is recommended that Crystal image be configured by activating the `FILE->SETTINGS` menu. The following settings are available:

- *Show file import info*. If active, Crystal Image displays a text window with messages related to the import of TIFF and DICOM files. This option is used to debug import functions and is normally kept inactive.
- *Preferred interpolation method*. For any operation that needs interpolation, the preferred interpolation method can be selected. The default is bilinear; a better (and slower) choice is bicubic. The spline interpolation methods were developed by Thevenaz et al. (Thevenaz P, Blu T, Unser M. Interpolation revisited. IEEE Trans Med Imaging 2000; 19:739–758) and the implementation should be considered experimental. However, experiments have shown an extremely high quality of iterative interpolation operations.
- *Debug message level*. If Crystal Image is run from a shell console, debug messages can be printed. A higher number causes more detailed messages to be printed. A good value is 1.
- *Undo function*. Crystal Image offers two undo stacks, a one-level undo in memory and a 10-level undo on hard disk. For operations on large images, storing images onto the hard disk for undo may cause a noticeable delay.
- *Temp path for undo*. If the 10-level undo stack on hard disk is selected, a path to a temporary directory needs to be provided. `/tmp` or `/var/tmp` is a good

choice. This setting does not play a role for undo in memory or if undo is disabled.

- *Path to browser*. The help files are provided in html format, and the help function calls the browser entered in this field. For KDE users (kubuntu), `/usr/bin/konqueror` is a good choice.
- *Path to help files*. This is the location where the help files can be found. In the default installation, this is `/usr/local/cimage/doc`.
- *Path to dso modules*. This path points to user dso modules. Since Crystal Image always loads modules found in `/usr/local/cimage/dso`, this field should only be set for users who develop modules.
- *Path+name of 3D viewer*. This entry should point to the executable `cimage_sv`, which in the default installation is either `/usr/local/cimage/bin/cimage_sv` or `/usr/local/bin/cimage_sv`. This viewer is activated from the menu with `VIEW->3D surface plot`. If the executable is not found, or if OpenGL is not installed, Crystal Image uses a built-in fallback viewer.

Menu Functions

The `FILE` menu:

`OPEN`: Opens an image file in any of the supported formats. Built-in file formats are TIFF, DICOM, PNM, VSRT, and VFLT. Other file formats are accessible through libraries.

`OPEN MULTIPLE`: Opens multiple two-dimensional image files as a stack. All files need to have the same data type and the same image size. Standard UNIX wildcard characters (*, ?, []) can be used to select a useful file mask

`IMPORT`: Imports raw format and ASCII format files. The data type and image dimensions of the file need to be known and entered in a dialog window.

`3-CHANNEL-LOAD`: Loads three images and assigns them to the color channels R, G, and B. All images must have the same size. Images are rescaled to the value range 0 to 255.

`SAVE`: Saves the main image. Save options include TIFF, raw, raw ASCII, VSRT, and VFLT if applicable, PBM if applicable, and OpenDX native format. For stacks, the number of slices can be restricted.

`CLOSE`: Closes the main image window and discards the image.

`SETTINGS`: The settings are explained above.

`QUIT`: Exit the software.

The `EDIT` menu:

`UNDO`: reverts the last operation. This option needs to be activated form the `SETTINGS` menu. Undo can revert one single operation (undo in memory) or up to 10 operations (undo on disk). Undo files are stored in TIFF format.

TRIM IMAGE: Functions that affect the image values and metadata.

CROP: Makes it possible to crop the image. If a rectangular ROI was set, the ROI will serve as a default cropping rectangle. Values may exceed the image dimensions—in this case, the image canvas is enlarged.

ROTATE/ZOOM: Makes it possible to rotate the image by an arbitrary angle or to rescale the image. In addition, new slices can be inserted between slices of a stack.

IMAGE BINNING: Reduces the size of an image by a factor of 2, 3, or 4 by square binning. This function averages the values inside the bin and therefore reduces noise.

DATA TYPE: Makes it possible to change the image's data type between 8-bit (CHAR), signed 16-bit (SHORT), IEEE Float, and RGB. For color images, model conversions to HSV and YCbCr are possible. Color images can be separated into their color components (stack).

CLICK STACK: The image coordinates of up to five mouse clicks are stored and can be edited (e.g., to obtain an exactly horizontal or vertical profile).

IMAGE VALUES: Makes it possible to modify the image values inside or outside a region of interest. Image values can be set to zero, a specified value, or noise can be added.

The VIEW menu:

MAGNIFIER: Opens a magnifier window that displays a magnified view of a 32 × 32-pixel-wide area. The magnified region is determined by a mouse click.

3D SURFACE PLOT: Plots the current main image as a three-dimensional elevation landscape. If the external viewer, cimage_sv, is available (see installation), a number of OpenGL-related options such as lighting and perspective are available. Otherwise, a less sophisticated built-in fallback viewer will be used.

COLORMAPS: Opens the color map dialog to adjust brightness, contrast, and gamma, and to add false coloring and contour lines for visualization.

MIP: Maximum intensity projection. Projects an image stack by collecting pixels with maximum intensity parallel to the z-axis.

PROFILE: Draws an intensity profile. The function connects the last two clicked points with a straight line (clicking on different slices in a stack is possible; editing the points with EDIT>CLICKSTACK is also possible) and plots the intensity values over the position in a graph display window.

LINE AVERAGE (X): Takes a rectangular ROI and averages the image values along vertical scan lines, then plots the average values as a function of the relative x position similar to the profile function.

LINE AVERAGE (Y): Takes a rectangular ROI and averages the image values along horizontal scan lines, then plots the average values as a function of the relative y position similar to the profile function.

Z CUT: If the image is a stack, this function displays a two-dimensional slice where one dimension is the line connecting the two most recently clicked points and the other dimension is z (the slice)

HISTOGRAM: Displays the histogram of the image, stack, or current ROI.

The PROCESS menu:

INVERT: Creates a negative of the image by subtracting all image values from the highest image value.

BACKGROUND REMOVAL: Opens a dialog with background removal functions. Available functions are moving window (unsharp masking with a square box kernel of adjustable size), unsharp masking (unsharp masking with a Fourier-based Gaussian lowpass filter of adjustable σ), linear plane (least-squares fit of a planar background function), and parabolic plane (least-squares fit of a bi-parabolic background function). Background removal assumes that the background is represented by low image values.

CLUSTER LABELING: Identifies connected features with nonzero value that are separated by zero-valued background (requires prior image segmentation). Features may be labeled by size, aspect ratio, compactness, or irregularity. Optionally, tabular results may be displayed (Chapter 9.2).

THRESHOLD: Provides access to thresholding options (Chapter 2). Thresholding functions are manual single-level threshold with binary, soft, or hard threshold functions, interactive dual-level threshold selection with preview, and automatic thresholding with a hard threshold function using iterative threshold finding or Otsu's method.

BINARY: Operators that operate on binary images (Chapter 2). Nonbinary images are interpreted as zero/nonzero. This menu gives access to morphological operators (erode, median, dilate, close, open; with multiple iterations and alternating 4/8 neighborhood), skeletonization (modified Zhang–Suen algorithm), hole filling (making all zero-valued regions that are not connected with the background nonzero), and feature centering (moving all features so that the joint centroid is in the image center).

FFT: Makes it possible to perform forward and inverse FFT and DHT of a single slice (Chapter 3). FFT creates a two-slice stack with real and imaginary parts.

FILTER: Provides access to spatial-domain filters, adaptive spatial-domain filters (Chapter 5), and frequency-domain filters (Chapter 3).

EQUALIZE HISTOGRAM: Performs global or local histogram equalization (Chapter 2). Note that local histogram equalization is not efficiently implemented and takes several minutes, depending on image size.

HOUGH TRANSFORM: Creates Hough transforms of the current image slice (Chapter 7). Currently, forward and inverse line and circle transforms are implemented.

IMAGE MATH: Allows pixel-by-pixel math operations on a single image or on pairs of images. For operations on pairs of images, the first image needs to be loaded and pushed on the stack, followed by loading the second image, before the operation can be performed. Image pairs need to have the same x and y dimensions. Stacks need to have the same z dimension unless one image is not a stack, in which case the operation is repeated on a slice-by-slice basis.

The SEGMENTATION menu:

SNAKES: Several implementations of two-dimensional active contours (Chapter 6). This function creates a ROI rather than actually segmenting the image. The snake implementations need to be considered experimental.

SPIDER OP: Segmentation of convex objects by polygonal approximation (Haidekker MA, Andresen R, Evertsz CJ, Banzer D, Peitgen HO, Brit J Radiol 1997; 70:586–93). Samples image intensity along radial probing rays and uses the maximum intensity or maximum gradient as nodes (i.e., intersections with the object). Contains experimental specialized functions for vertebral density analysis.

HYSTERESIS THRESHOLD: Segmentation by hysteresis thresholding (Chapter 2).

WATERSHED: Watershed segmentation of overlapping convex objects. Requires a binary segmented image. This function can also provide the Euclidean distance map and the ultimate eroded points (Chapter 2). In presegmented gray-scale images, the gray-scale information may be included for segmentation.

REGION GROWING: Two- and three-dimensional region-growing functions that use the click stack as seed points (Chapter 2).

GROW MERGE: This is a modification of the region-growing algorithm where multiple regions are grown from individual seed points. They merge if their average value falls within a threshold range. This function has never been fully tested and should be considered incomplete.

K-MEANS CLUSTERING: Implementation of the k-means clustering algorithm.

The MEASURE menu:

STATISTICS: Measures gray-value statistics inside the image or a ROI slice by slice. In pre-thresholded images, this function optionally excludes zero-valued pixels as background.

BOUNDARY LENGTH: Requires a presegmented image. Counts the number of pixels that are 8-connected with the background.

CLUSTER MAXIMUM: Determines a threshold value which, when used to segment the image, creates the maximum number of connected features.

FRACTAL DIMENSION: Implementation of various estimators of the fractal dimension (Chapter 10).

ROSE PLOT: Directional frequency decay (Chapters 8 and 10). This function takes a gray-scale image and computes the FFT. In radial directions, a linear function is fit into log magnitude over log frequency, and the slope (power law of decay) is plotted as a function of the radial angle. The frequency range can be limited. This function reveals self-similar properties in the frequency domain if they exist in the image.

RUN LENGTHS: Computes a run-length histogram of a binary or gray-scale image. Binning of gray levels and runs is possible. Produces a four-slice stack with runs in the 0°, 45°, 90°, and 135° directions.

The SPECIAL menu:

CT VERTEBRAL BONE: Functionally similar to the SPIDER OP.

CT VASCULAR GRAFTS: Functionally similar to the SPIDER OP.

WAVELETS: Provides the wavelet transform and wavelet-based noise filtering (Chapter 4). This function is provided by a module and appears only if the dso was loaded properly.

COOCCURRENCE MATRIX: Computes the co-occurrence matrix of an image (Chapter 9). The displacement can be selected. This function is provided by a module and appears only if the dso was loaded properly.

AUTOCORRELATION MATRIX: Computes the autocorrelation matrix of an image in the spatial domain. This function is provided by a module and appears only if the dso was loaded properly.

GENERATE: Offers different functions to create synthetic images and test images. This function is provided by a module and appears only if the dso was loaded properly.

The MACROS menu:

LOAD MACRO: Loads a text file with macro definitions, parses the macros, and adds the macros to the MACROS menu. Loaded macros can now be activated using this menu.

DELETE MACROS: Deletes all presently loaded macros.

IMMEDIATE EXECUTION: Opens a console for the immediate execution of ImageBasic commands (Chapter 14).

The HELP menu:

CIMAGE HELP: Starts a browser (see the installation and setup instructions) and displays the help index file.

ABOUT: Displays the software version.

`IMAGE INFO`: Displays information about the presently loaded image (dimensions, image values, image type, ROI).

`SHOW PID`: Adds the PID (process ID number) of the present Crystal Image instance to display window titles. Useful to assign display and result windows to each other when multiple instances of Crystal Image are running.

`LICENSE`: Displays license and warranty information.

Keyboard and Mouse Shortcuts in the Image Window

Once an image is open (the main image window), a number of mouse functions and keyboard shortcuts are available. Note that the keyboard shortcuts are case-sensitive. For example, *a* is the lowercase key, and *A* is the uppercase key (shift + *a*). Most keyboard shortcuts are sensitive to where the crosshairs hover when the key is pressed.

Left-click	Display the image value under the cross-hairs in the statusbar.
Left-drag	Drag a rectangular ROI.
Middle-drag	Move the ROI.
Right-click	Remove ROI (or close polygonal ROI).
Shift-click	Set a vertex for a polygonal ROI.
Arrow keys	Move ROI by one pixel.
PgUP key	Next slice in stack.
PgDN key	Previous slice in stack.
a key	"Select All" ROI over the entire image.
c and *C* keys	Create a circular ROI and shrink/enlarge it.
e key	Shortcut for `Edit->Values`.
F key	If pressed above a cluster, returns Fourier shape descriptors of the cluster.
f key	Flood-fill an area. If used above the background, fills with the maximum image value; if used above the foreground, fills with zeros.
r and *R* keys	Add/remove a snake repulsor (works only when a snake is active).
s and *S* keys	Add/remove a snake attractor (works only when a snake is active).

Macros

The basic structure of a Crystal Image macro was introduced in Section 14.4 and an example macro was given in Algorithm 14.1. Macros are written as plain ASCII text files with a suitable text editor. For the popular `kate` text editor, part of the KDE desktop environment, a definition for syntax highlighting (file `cimage.xml`) exists, and its use is highly recommended because it gives instant feedback about the correct syntax. Macros are case-insensitive. Spaces, tabs, and linefeeds are ignored in macro files. Therefore, tabs and empty lines should be used for structuring and to improve readability. Any text file may contain one or more macro definitions and zero or more procedure definitions. A macro definition is enclosed by the keywords `macro` and

`endmacro`, and a procedure definition is enclosed by the keywords `procedure` and `endproc`. A mismatch between the keywords is detected in the parsing stage and the macro file is rejected. A macro is entered into the MACRO menu, whereas a procedure is not accessible outside the macro interpreter. A procedure acts like a subroutine in BASIC. A procedure can be called from a macro, and control is returned to the calling function when an `endproc` statement is reached. With the exception of menu entry, a macro and a procedure behave in an identical manner.

Variables Variable names must start with a letter and may contain letters, digits, and the underscore. Variable names must not contain keywords because the parser extracts the keywords. For example, a variable `bandnumber` is not allowed because it contains the keyword `AND`. Variables are typed and can contain integer values, real values, text strings, or images. Variables do not need to be predefined, as they are created upon first use. The type is assigned when a variable is created and cannot be changed. If the result of an expression does not match the variable type, a type mismatch error occurs. However, variables can be disposed of and created with a different type. All variables are global and thus accessible in all macros and procedures. Variables are deleted (and their allocated memory freed) after the macro run finishes. In immediate-execution mode, variables are deleted after the immediate-execution window is closed. The variable `mainimage` is predefined and contains the image that is currently open and displayed. The variable `mainimage` must not be used on the left-hand side of an expression.

Basic Operations The parser knows the most common arithmetic expressions, +, −, *, and / for addition, subtraction, multiplication, and division; = is the assignment operator when it follows a variable name at the beginning of an expression and the equal operator inside an expression. Other Boolean operators are <> and != (not equal), <= (less than or equal), >= (greater than or equal), < (less), > (greater than), and the logical operators are `AND` (logical AND), `OR` (logical OR), and `XOR` (logical exclusive-or). The corresponding operators must have matching types, although exceptions exist. The following table provides an overview of type-mixing results.

	int	real	image	string
int	int	real[a]	image[b]	
real	real[a]	real[a]	image[b]	
image	image[b]	image[b]	image[c]	
string	—	—	—	string[a]

[a] Comparison operators are an exception since they always return an integer scalar. Logical operators are not allowed.

[b] Boolean and logical operators cannot be combined when one operand has a floating-point type. Any operation between an image and a scalar operand results in a floating-point type image.

[c] Image size must match. Boolean and logical operators cannot be used with floating-point type images. Comparison operators cannot be used if both operators are images.

The − (minus) sign may be used as a unary operator only with numbers (e.g., `a = -5` is an allowed expression) but is invalid with images. To change the sign of the values of an image, it needs to be explicitly subtracted from zero: for example, `inv = 0-img`. The + (plus) sign can never be used as a unary operator.

Arrays Crystal Image's ImageBASIC knows one-, two-, and three-dimensional arrays. Array indices are enclosed in brackets, [·]. Images can readily be addressed as arrays, provided that the number of indices matches the dimension. It is not possible, for example, to extract a slice from a stack by addressing the stack (three-dimensional image) as a two-dimensional array. Any expression that includes array indices results in a floating-point scalar. For example, the assignment `e = img[120,5]`; returns the image element at coordinates $x = 120$ and $y = 5$ of the two-dimensional image `img` and assigns it to the scalar floating-point variable e. If the image is a color image, the luminance value is returned. If the image is three-dimensional, an error occurs in this example. A one-dimensional array named `user` is predefined. Several operations return multiple values in the `user` array. All other array variables must be defined in advance with the `dim` function.

Flow Control The available structures for flow control are the `for` loop, `foreach` loop, `while` loop, `repeat/until` loop, and `if` statement.

```
for (var=expr1, expr2 [,expr3]); statements; endfor;
```

executes a for-loop. The variable `var` may be integer- or real-valued. `expr1` is the expression that determines its initial value and `expr2` its final value. The final value is included; that is, the loop executes once when `var` reaches the final value. The `for` loop executes at least once. `expr3` determines the increment, and its use is optional. The default increment is 1. Each `for` must have a matching `endfor`.

```
foreach (strvar, strexpr1 [,strexpr2]); statements; endfor;
```

executes a foreach loop. `strexpr1` is a string expression that is used as a file mask. `strexpr2` specifies an optional working directory. The foreach loop executes once for each file in the current working directory that matches the expression `strexpr1`, and the matching file name is assigned to `strvar`. For example, a `foreach` loop can be executed for each TIFF file in the current working directory: `foreach (fn, "*.tiff")`; Each `foreach` must have a matching `endfor`.

```
while (expr); statements; endwhile;
```

executes a while loop. The statements enclosed between `while` and `endwhile` are executed as long as `expr` evaluates as nonzero. If `expr` is evaluates as zero at the beginning of the loop, the while loop does not execute the enclosed statements at all. Each `while` must have a matching `endwhile`.

```
repeat; statements; until (expr);
```

executes a repeat loop. The loop executes at least once and exits when `expr` evaluates as nonzero. Each `repeat` must have a matching `until`.

```
if (expr); statements1; [else; statements2; ] endif;
```

allows the conditional execution of a macro program. If `expr` evaluates as true (nonzero), the first block of statements is executed. If `expr` evaluates as false (zero), the second block of statements is executed. Each `if` must have a matching `endif`, but the `else` directive is optional.

Procedures

call: Executes a different macro or procedure as a subroutine.

Syntax: `call (strexpr);`
Parameter: Strexpr evaluates to the name of a macro or procedure as given in its definition.
Example: `call ("loadfile");`

convert_image: Converts an image to a different type.

Syntax: `convert_image (imgvar, strexpr, expr);`
Parameters: imgvar contains the image to be converted; strexpr is the target type and may be any of "float," "char," "short," or "rgb," expr is a Boolean expression that directs whether to expand the value range to match the target type.
Example: `convert_image (myimg, "float", 0);`

dim: Dimensions (declares) array variables with floating-point elements.

Syntax: `dim (varname[subscript] ,varname[subscript]...);`
Parameters: One or more variables with the array size in brackets.
Example: `dim (ary1[100], twodimary[64,64]);`

display: Places an image into the program's main image buffer for display.

Syntax: `display (imgexpr [, strexpr]);`
Parameter: Image expression for image to be displayed. The optional strexpr is the window title.

dispose: Deletes a variable from the variable list and frees its memory if it is an image or array.

Syntax: `dispose (var [,var...]);`
Parameter: Variable to dispose of.
Example: `dispose (a,b,c);`

fillroi: Fills a region of interest with a specified value.

Syntax: `fillroi (imgvar, expr1 [,expr2] );`
Parameters: imgvar is the image variable on which to operate, and expr1 is the value to fill the ROI with. If the image is a stack, all slices are filled unless expr2 is given; in that case, only the slice numbered by integer expr2 is filled.
Example: `fillroi (img, 0);`

graph: Plots *x,y* data stored in arrays.

Syntax: `graph (intexpr, numarray1 [, numarray2] );`
Parameters: intexpr specifies the number of data points. The arrays contain the data to be plotted. If two arrays are given, both need to be one-dimensional; the first contains the *x* data and the second contains the *y* data. If one array is given, it must be two-dimensional. In that case, the first subscript indexes the data points, and the second subscript determines the data set; 0 = *x*-data; 1 = first *y*-data, etc. Thus, with a $m \times n$ array it is possible to plot $n - 1$ curves with up to m data points.
Example: `graph (100, xydata);`
Notes: Intexpr must be less than m. All numarrays must be variable references. Expressions are not allowed.

print: Prints data into the output window.

Syntax: `print (expr, expr,... [,]);`
Parameters: A list of string or numerical expressions that get evaluated and printed in sequence. If the list ends with a comma, the final line feed is suppressed.
Example: `print ("Thresholding by", "auto_threshold(img,0));`

save: Saves an image to a file.

Syntax 1: `save (imgexpr, "interactive");`
Syntax 2: `save (imgexpr, strexpr1, strexpr2 [, expr1 [,expr2 [,expr3]]]);`
Parameters: Image expression or image variable that contains the image to be saved. Strexpr1 is a string expression for the file name, strexpr2 is a string expression for the file type (may be any of "tiff," "vsrt," "raw," "pnm," "opendx"), expr1 is the start slice, expr2 is the end slice, and expr3 is the tiff compression.
Examples:
```
save (thisimage, "output.tif," "tiff," 0,39,0);
save (myimage, "interactive");
save (astack, "output.dx", "opendx");
```
Notes: If the keyword "interactive" is used, the file save dialog is opened by the macro, and the user must enter the file name interactively. If imgexpr is an expression, the resulting image will always be a 32-bpp floating-point image. Only if a variable name is provided will the original image type be retained.

setroi: Defines a region of interest.

Syntax: `setroi ([expr,... ]);`

Parameters: If the parameter list is empty, the ROI will be deleted. If the parameter list contains exactly three values, they define a circular ROI with *x*, *y*, and radius. If the parameter list contains four values, they define a rectangular ROI. If the parameter list contains $2n$ values with $2n \geq 6$, the values define a polygonal ROI.

Note: The macro `Display` command may reset the ROI.

Functions Contrary to procedures, functions return a value and must be used in the right-hand side of an assignment.

log, log10, exp, int, round, power: General-purpose math functions.

Syntax: `log (expr)` (example)
`power (expr1, expr2)`

Description: These functions return the natural logarithm, the base 10 logarithm, the exponential, the next smallest integer, and the nearest integer of the argument. Log, log10, and exp operate on images on a pixel-by-pixel basis. Int and round require a floating-point argument and return an integer. Power raises expr1 to the power of expr2. If expr1 is an image and expr2 is a scalar, each pixel of the image will be raised to the power of expr2. The result is an image. If expr1 is a scalar and expr2 an image, the result is also an image, and each pixel of the result will contain expr1 raised to the power of the corresponding pixel in expr2. This is the function that needs to be used as opposite of `log10` where expr1 is 10.

strlen, strnum, strpos, substring: String-related functions.

Description: Strlen returns the length of a string argument. Strnum converts a numerical value into a string; its first argument is a string expression, and the second (optional) argument is either the number of digits or a C-style format string. strpos returns the position of the first string in the second or a negative value if the first string does not occur in the second. Substring requires three arguments, the first a string and the next two integer values that indicate the start and end of the substring.

auto_threshold, otsu_threshold: Determine an optimal threshold value.

Syntax: `auto_threshold (imgexpr, expr)`

Return value: Floating-point number

Parameters: Image expression to determine ideal threshold for. Expr is the image slice to perform the function on.

Example:
```
bin = threshold (thisimage,
  auto_threshold(thisimage,0), "hard");
```

Notes: Auto_threshold computes the iterative threshold for an image; otsu_threshold uses Otsu's method. This function can be used as a parameter in the threshold function.

center: Center features so that their centroid coincides with the image center.

Syntax: `center (imgexpr, mode [, expr2])`
Return value: Image
Parameters: Imgexpr is the image that contains the features. Expr1 is a flag: if expr1 is zero, the image is considered binary (zero/nonzero pixels); if expr1 is nonzero, the centroid is weighted by the image gray values; expr2 is the slice number. If it is omitted, the center operation will be performed on all slices of an image stack.
Example:
```
c = center (centerme, 1);
```

cluster2d, cluster3d: Perform cluster labeling.

Syntax:
```
cluster2d (imgexpr, expr1, expr2, ..., expr6)
cluster3d (imgexpr, expr2,..., expr6)
```
Return value: Image
Parameters: imgexpr is the image to label. Expr1 is the slice number (only for two-dimensional operation); expr2 is the threshold value; expr3 is the minimum cluster size (all clusters below that size are considered noise and deleted); expr4 limits the number of clusters to detect; expr5 is the neighborhood connectivity (4 or 8); expr6 is the labeling order (0: size, 1: compactness, 2: aspect, 3: irregularity).
Example:
```
lbl = cluster2d (img, 0, 1, 3, 32767, 4, 0);
```
Note: The three-dimensional function cannot yet handle different labeling orders, and expr6 is ignored.

convolve: Spatial-domain filters (the name "convolve" comes from a development stage when this function provided only convolution-based filters).

Syntax:
```
convolve (imgexpr, strexpr [, expr...] )
```
Return value: Image
Parameters: Image expression to perform the filtering on. Strexpr defines the filter function, which is one of "smooth," "sharpen," "sharpen more," "laplace" for standard 3×3 kernels, "sobel" and "compass" for the Sobel and compass edge detectors, "gauss" for generalized Gaussian smoothing (expr is neighboorhood size 1...9); "LoG" for Laplacian of Gauss second derivative (expr is neighboorhood size 1...9); "locvar" for local variance (expr is neighboorhood size 1...9); "locmax" to find local maxima (expr is neighboorhood size 1...9), "AMMSE" is the adaptive lowpass or minimum mean-squared-error filter (expr is neighboorhood size 1...9); "TSM" is the adaptive three-state median filter (expr is a real-valued threshold), "ADLF" is the anisotropic diffusion lowpass filter [expr1 is a real-valued threshold, expr2 is the number of iterations (int)], "KUWA" is the Kuwahara filter (expr is neighboorhood size 1...9).
Example:
```
lapl = convolve (thisimage, "laplace");
```

crop: Crops an image to the absolute coordinates (x1,y1) to (x2,y2).

Syntax: `crop (imgexpr, expr1, expr2, expr3, expr4)`
Return value: Image
Parameters: Image expression for the image to crop. Expr1 through expr4 are x1, y1, x2, and y2, respectively.
Examples: `smallimg = crop (myimg, 10,10,250,250);`

dimension: Estimates a fractal dimension.

Syntax: `dimension (imgexpr, strexpr, expr1, expr2, expr3, expr4, expr5)`
Return value: Real-valued number
Parameters: Image expression of a 2D image to obtain the dimension from. Strexpr determines the estimator notion and may be any of "boxcount," "minkowsky," "massdim," "mahattan," "dispersion." Expr1 is the slice number in a stack; expr2 and expr3 are the lower and upper size range (their significance depends on the notion); expr4 and expr5 are the lower and upper threshold, and expr5 is a flag that determines whether zero-valued pixels should be considered.
Example: `D = dimension (img, "minkowsky", 1, 7, 1, 255, 0);`
Notes: Expr2 and expr3 are responsible for restricting the scale. With box counting, expr2 is the minimum box size and expr3 is the maximum box size: for example: 2 to 64. The Minkowsky and Mass dimensions are different. For Minkowsky, this is the dilation radius to analyze (typical values, 1 and 5), and for the Mass dimension this is the number of scale reductions. For example, 1 to 4 would cover box sizes (squared) 4, 16, 64, and 256. Thresholds are ignored in the mass dimension notion. Expr5 is ignored except in the mass dimension computation.

fft, invfft: Compute the complex-valued FFT and inverse FFT of one slice of an image.

Syntax: `fft (imgexpr, expr1)`
`invfft (imgexpr)`
Return value: Image
Parameters: Image expression for the image to transform. Expr1 is the slice in an image stack or zero for a two-dimensional image.

fourierfilt: Applies a frequency-domain filter

Syntax: `fourierfilt (imgexpr, strexpr, expr1,expr2,expr3,expr4, expr5)`
Return value: Image
Parameters: Image expression to perform the filter operation on. Strexpr is the filter function and can be any of "hhp" (homomorphic HP), "bhp" (Butterworth highpass), "blp" (Butterworth lowpass), "glp" (Gaussian lowpass), or "deriv" (first derivative).
Expr1 through expr4 are the filter parameters (real). Expr5 is a flag whether to logarithmize data before filtering.
Example:
```
newimg = fourierfilt (img, "hhp",
  10.0,0.3,0.2,0,0);
```

freichen: Applies the Frei–Chen edge/line detector algorithm.

Syntax: `freichen (imgexpr, strexpr)`
Return value: Image
Parameters: Image expression to perform the convolution on. Strexpr is either "edge" for the Frei–Chen edge space or "line" for the Frei–Chen line space.
Example:
```
fcedge = freichen (thisimage, "edge");
```

histeq: Performs local or global histogram equalization.

Syntax: `histeq (imgexpr, expr1, expr2)`
Return value: Image
Parameters: Performs histogram equalization of an image passed by imgexpr. A histogram is computed with the number of bins given by expr1. Expr2 is a flag that makes it possible to exclude zero-valued pixels from being processed.
Example:
```
eq = histeq (img, 256, 0);
```
Notes: This is a two-dimensional slice-by-slice operation. Local histogram equalization is not yet implemented.

histogram: Returns histogram descriptors of an image.

Syntax: `histogram (imgexpr, strexpr, expr1, expr2, expr3)`
Return value: Real-valued number
Parameters: Image expression to obtain histogram descriptors from. A string expression that determines which value to compute; may be any of "mean," "median," "mode," "variance," "skew," "kurtosis"; expr1 determines the slice if imgexpr is a stack; expr2 determines the number of bins to be used in histogram computation; expr3 is a flag that determines whether zero-valued pixels should be considered.
Example:
```
s= histogram (img, "mode", 0, 256, 0);
```

hysteresis_threshold: Performs hysteresis thresholding.

Syntax: `hysteresis_threshold (imgexpr, expr1, expr2)`
Return value: Image
Parameters: Image expression to perform the thresholding on. Expr1 is the lower threshold, expr2 is the upper threshold.
Example: `bin = hysteresis_threshold (thisimage, 45, 90);`
Notes: Hysteresis_threshold returns a binary image. On stacks, it performs a two-dimensional operation slice by slice.

imginfo: Retrieves global information about an image.

Syntax: `imginfo (imgvar, strexpr)`
Return value: Real or integer numerical value
Parameters: Image expression to get data from. Strexpr determines what information is returned. May be any of "xmax," "ymax," "zmax," "minval," "maxval," "meanval," "type."
Example: `xmax = imginfo (thisimage, "xmax");`
Notes: "meanval" computes the image mean value in a loop over all image pixels and may be a time-consuming operation. "type" returns a numerical value that needs to be translated into "char" (0), "short" (1), "float" (3), and "rgb" (2).

imgtorgb: Joins three images into a RGB image, rescaling each image.

Syntax: `imgtorgb (imgexpr1,imgexpr2,imgexpr3)`
Return value: Image
Parameters: Imgexpr1,2,3 are the images that get assigned to the red, green, and blue channels. An imgexpr that yields a color image will be converted to gray scale.
Example: `colrful = imgtorgb (redimg, greenimg, blueimg);`

invert: Inverts the image values (subtracts each pixel from the image maximum)

Syntax: `invert (imgexpr)`
Return value: Image
Example: `invimg = invert (img*mask);`

kltransform: Performs the Karhunen–Loève transform on a RGB image and returns the principal component.

Syntax: `kltransform (imgvar, expr1)`
Return value: Image
Parameters: imgvar must contain a RGB color image. Expr1 is a flag that makes it possible to exclude black pixels with $R = B = G = 0$.
Example: `pc = kltransform (rgbimg, 0);`
Note: imgvar may contain a RGB stack. In that case, all slices will be transformed and the result is also a stack. The result image is a floating-point gray-scale image.

load: Loads an image from a file.

Syntax: `load (strexpr)`

Parameter: String expression that results either in the keyword "interactive" (for interactive file selection) or a file name.

Return value: Image

measure: Measures the average image intensity value in a region of interest.

Syntax: `measure (imgexpr, expr1 [, expr2])`

Return value: Real-valued number

Parameters: imgexpr is the source image. Expr1 is a flag that determines whether zero-valued pixels should be considered; expr2 restricts the measurement to a single slice.

Example: `I = measure (img, 0);`

Notes: The measure function determines the average, standard deviation, and area of an image. If a ROI is specified, it will be honored. The function returns the pixel average value and stores the same value in `user[0]`. Standard deviation can be found in `user[1]`, and the number of counted pixels (i.e., the area) in `user[2]`. When the slice parameter expr2 is not given and the image is a stack, all pixels in the stack within the ROI are considered. The return values are different for RGB images, where the return value is 0 and `user[1]` is undefined. `user[2]` still contains the pixel count. `user[3]` through `user[5]` contain the average intensities for R, G, and B, respectively, and `user[6]` through `user[8]` contain the corresponding standard deviations. It is the macro programmer's responsibility to trap the RGB case.

morph: Performs morphological operators and rank filtering.

Syntax: `morph (imgexpr, strexpr, expr1 [, expr2])`

Return value: Image

Parameters: Image expression to perform the morphological operation on. Strexpr is the morphological operator and may be any of "erode," "dilate," "median," "cwmf," "patch," and "bound_erode." Expr1 defines the connectivity and must be either 4 or 8. If the fourth parameter expr2 is given, only the specified slice will be filtered (three-dimensional images only)

Example: `s = morph (thisimage, "median", 8);`

quantile: Computes the threshold value that divides the image histogram into q and $1 - q$.

Syntax: `quantile (imgexpr, expr1, expr2, expr3, expr4)`

Return value: Real-valued number

Parameters: Image expression to compute the quantile from. Expr1 is the slice in an image stack, expr2 is the quantile q given in percent, expr3 is the number of bins, and expr4 is a flag that determines whether zero-valued pixels should be considered.

Example: `t = quantile (myimg, 50, 1024, 1);`

Note: For an 8-bit image, expr3 should be no more than 256.

regiongrow: Performs region growing with multiple seed points and a dual threshold.

Syntax:	`regiongrow (imgexpr, expr1, expr2, vertexlist)`
Return value:	Image
Parameters:	imgexpr is the source image. Expr1 and expr2 are the lower and upper threshold. Vertexlist is a sequence of $3k$ integer expressions that define k seed points in the order x, y, z.
Example:	`w = regiongrow (oldimg, 100,150, 128,128,0);`

remove_bk: Background flattening filters.

Syntax:	`remove_bk (imgexpr, strexpr, expr1, expr2)`
Return value:	Image
Parameters:	Imgexpr is the image to remove background from. Strexpr determines the method and may be any of "movavg" (unsharp masking with a moving-average window), "unsharp masking" (uses FFT Gaussian blurring to create a background mask. The Gaussian sigma is 400/expr1 for compatibility with the radius, that is, a larger r means weaker filter action, "plane" (fits a linear plane where the tiles have a size of expr1), and "parabolic" (fits a two-dimensional parabolic function where the tiles have a size of expr1). Expr2 is a flag that determines whether zero-valued pixels should be considered.
Example:	`oneslice = remove_bk (myimg, "movavg," 40,1);`
Note:	This function assumes that background pixels have lower values than feature pixels.

resize: Resizes an image by interpolation or rebinning.

Syntax:	`resize (imgexpr, expr1 [, expr2])`
Return value:	Image
Parameters:	Image expression for image to resize. If only expr1 is given, the image is reduced by rebinning, and expr1 is the reduction factor (allowed values 2, 3, and 4). If both expr1 and expr2 are given, the image is interpolated to a new size of expr1 by expr2 pixels.
Example:	`new = resize (img, 1024, 1024);`

rgbtostack: Converts a RGB image to a three-slice stack with the order R–G–B.

Syntax:	`rgbtostack (imgexpr)`
Return value:	Image
Parameter:	Imgexpr must be a RGB image.
Example:	`stk = rgbtostack (myimg);`

roseplot: Fits a straight line into LOG(magnitude) over LOG(frequency) in the FFT of an image and returns the slope over different radial angles.

Syntax: `roseplot (imgexpr, expr1, expr2, expr3)`
Return value: Real-valued number
Parameters: Image expression to compute the rose plot on. Expr2 determines the number of points starting at the frequency expr1 into which to fit the slope; expr3 determines the number of radial points and must be less than 80.
Example: `havg = roseplot (fft(img,0), 10, 64);`
Notes: Imgexpr must contain a valid two-slice real/imaginary FFT image. The function returns the average Hurst exponent H, which is related to slope and dimension by $s = 2H + 1 = 2D$. In addition, the H is also returned in `user[0]` and the anisotropy of H in `user[1]`. The individual values of H for all radial directions specified in expr2 are returned in `user[2]` through `user[2+expr2-1]`.

rotate: Rotates an image clockwise.

Syntax: `rotate (imgexpr, expr)`
Parameters: Image expression for the image to rotate. Expr is the angle in degrees.

runlength: Computes the run-length histogram of a gray-scale image.

Syntax: `runlength (imgexpr, expr1, expr2)`
Return value: Image
Parameters: Image expression to compute the run-length histogram of. Expr1 is the number of bins for the gray-scale axis; expr2 is the number of bins for the run-length axis.
Example: `rlhist = runlength(img, 16, 16);`
Notes: A run is the number of consecutive pixels along a straight line that fall into the same gray bin. The histogram is two-dimensional (y-axis is the gray scale, x-axis is the run length), and the run count is the image value. The image returned is a stack with four slices, each corresponding to one run angle: 0°, 45°, 90°, 135°.

skeletonize: Computes the skeleton of a binary image.

Syntax: `imgvar = skeletonize (imgexpr, expr1 [,expr2] )`
Return value: Image
Parameter: Image expression to perform the skeletonization on. Expr1 is a threshold value that can be used to segment gray-scale images on-the-fly; expr2 is the slice number of a stack to skeletonize.
Example: `s = skeletonize (thisimage,1);`

substack: Returns one or more slices from a stack.

Syntax: `substack (imgexpr, expr1, expr2)`
Parameters: Imgexpr must be a stack. Substack extracts slices from expr1 to expr2 (inclusive).
Example: `oneslice = substack (myimg, 1,1);`

threshold: Performs thresholding on an image.

Syntax: `threshold (imgexpr, strexpr [, expr])`
Return value: Image
Parameters: Image expression to perform the thresholding on. Strexpr defines the method and is one of "binary," "soft," "hard," "otsu," "iterative," "softsym," or "hardsym." If the method is "otsu" or "iterative," expr must not be provided, and the image returned is a strictly binary image. For all other methods, expr is the threshold value, "binary" creates a strictly binary image, "soft" and "hard" apply soft and hard thresholding functions, respectively, and "softsym" and "hardsym" perform symmetrical thresholding with signed values.
Example:
```
bin = threshold (thisimage, "binary", 115);
```

threshold2: Performs two-level thresholding on an image.

Syntax: `threshold (imgexpr, expr1, expr2 [,expr3] )`
Return value: Image
Parameters: Image expression to perform the thresholding on. Expr1 specifies the lower threshold; expr2 specifies the upper threshold. If expr3 is provided, the function will be performed on only one slice.
Example:
```
bin = threshold2 (thisimage, 128, 131);
```
Notes: Threshold2 is a threshold "bandpass" which sets to 0 all image areas below expr1 or above expr2 and sets everything else to 1.

watershed: Performs a watershed segmentation an a binary image.

Syntax: `watershed (imgexpr, expr1, expr2 [, strexpr])`
Return value: Image
Parameters: Imgexpr is an image that contains a presegmented image. Expr1 is the threshold value; expr2 is the slice number for an image stack; strexpr is optional and may modify the returned image: if strexpr is "uep," the function returns the ultimate eroded points, and if strexpr is "edm," the function returns the Euclidean distance map.
Example:
```
w = watershed (myimg, 1, 0, "edm");
```
Notes: Performs the watershed transform on one slice of the image imgexpr. Note that a three-dimensional stack will be reduced to one single slice, so a substack operation in a loop is needed to do a full three-dimensional watershed transform. Before the operation, the image is thresholded with expr1.

APPENDIX B

SOFTWARE ON DVD

Apart from Crystal Image, the accompanying DVD comes with a number of software programs that may be useful for image processing, image analysis, or even general-purpose computing tasks. The most relevant programs are:

OpenDX	Data explorer and visualization software. Covered in Section 14.5.
ImageJ	Image analysis software maintained by the National Institutes of Health. Covered in Section 14.2.
Labplot	Universal data analysis, examination, and graphing tool.
CTsim	Computed tomography simulation tool, useful for exploring the principles of computed tomography image formation, and reconstruction.
Scilab	Matlab-style matrix algebra package with a number of add-on packages, such as a visual linear systems package. Covered in Section 14.1.2.
Octave	Open-source substitute for Matlab. Covered in Section 14.1.2.
GIMP	The GNU image manipulation program. This is a full-featured open-source substitute for the Photoshop software.
Inkscape	Vector-based graphics and illustration program.
OpenOffice	Suite of office applications (word processor, spreadsheet, presentation) that has a good degree of compatibility with Microsoft files.
AMIDE	Medical data explorer, covered in Section 14.3.
FMRIB Software Library	Functional MRI software library of Oxford University's FMRIB group.

Advanced Biomedical Image Analysis, By Mark A. Haidekker

INDEX

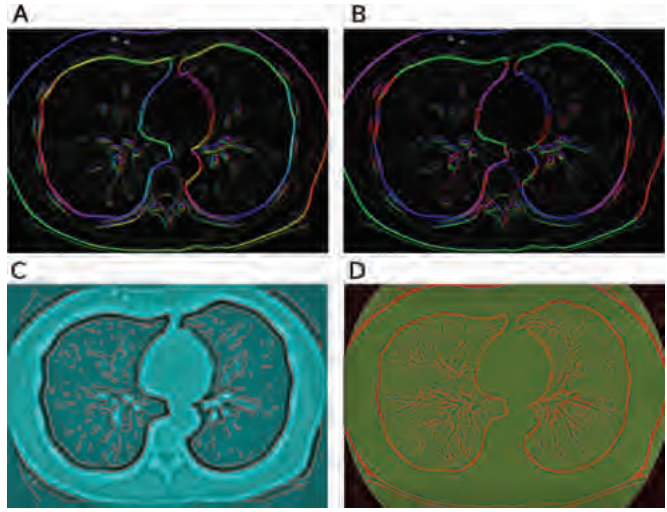


FIGURE 2.8 Demonstration of some edge detection operators. The original image is the CT slice in Figure 2.7A. Images A and B show the results of the Sobel and compass edge detectors with color-coded edge directions. Whereas the edge magnitude is very similar, the discrete nature of the edge directions in the compass operator becomes clearly visible. In image C the LoG operator with $\sigma = 3.0$ was applied, and zero crossings (edge locations) are marked in red. An apparent double edge appears at the steepest edges. Image D shows a color-coded version of the Frei–Chen operator. Red hues indicate high values (low projection angles) in edge space, with some similarity to the Sobel and compass operators. Blue–green hues indicate high values in line space, most prominent in the vascular structure and the thin traces of the ribs.

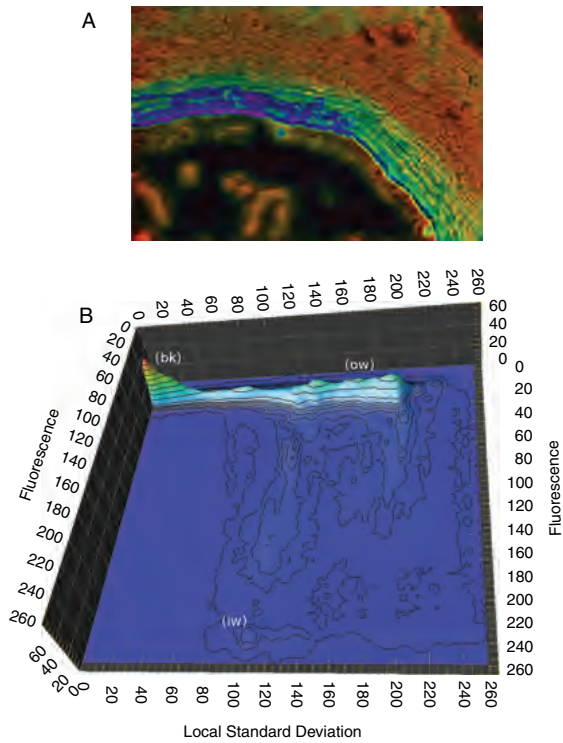


FIGURE 2.13 Multichannel thresholding. The upper image shows a composite created from Figure 2.12A (luminance), Figure 2.12B (green–blue hues), and the local range operator applied to Figure 2.12A (red–orange hues). A two-dimensional histogram of the last two images (B) shows peaks for the lumen (bk), the outer wall (ow) with high local irregularity but low fluorescence, and the inner wall (iw) with intermediate irregularity and high fluorescence.

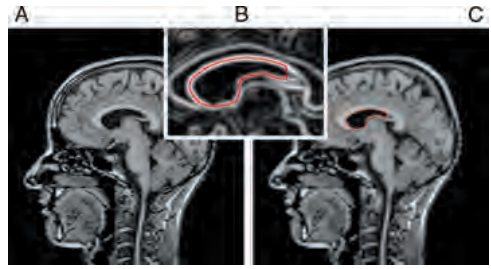


FIGURE 6.5 Application of a deformable contour for the segmentation of the ventricular space in an MR image slice of the brain (A). The snake tends primarily to follow the external energies, which are represented by the magnitude in the gradient image (inset B). Note that the snake uses a weak gradient ridge to take a "shortcut" on the right-hand side, where the bending energy would get very high. This behavior can be controlled with the parameter β. Image C shows the slice after the application of a smoothing filter with the snake superimposed.

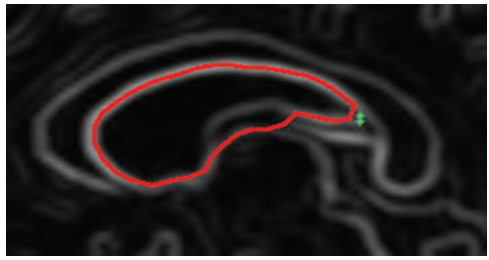


FIGURE 6.7 The snake is pulled into the high-bending zone by two attractors (green dots).

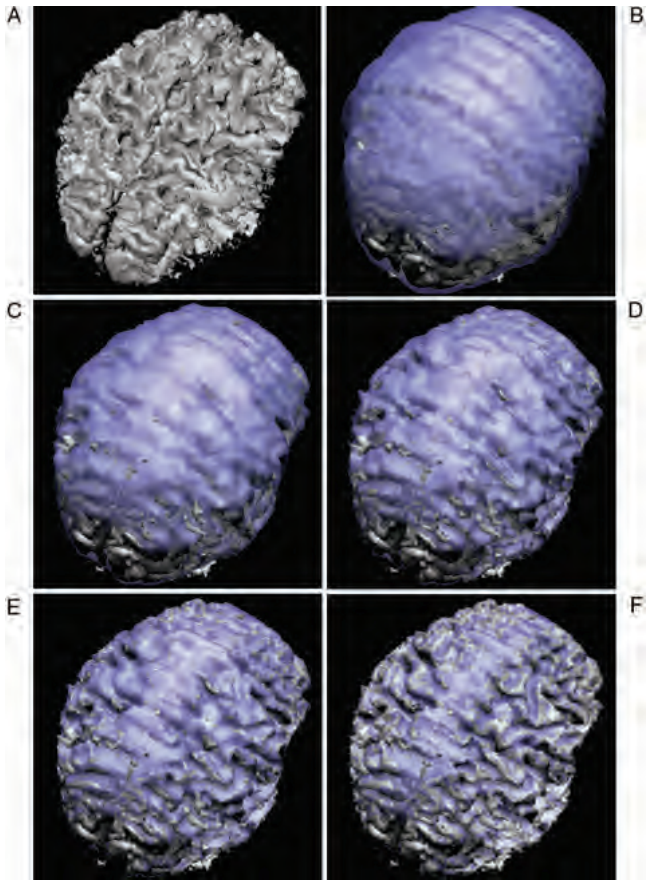


FIGURE 6.13 Surface rendering of a human brain from MR data (A) after segmentation of the brain and deepening of the sulci by thresholding. A balloon-line active surface is shown step by step to shrink around the segmented brain and follow the gradient forces into the sulci (B–F). Particularly in image F, the active surface can be seen to take some "shortcuts" to minimize its bending energy.

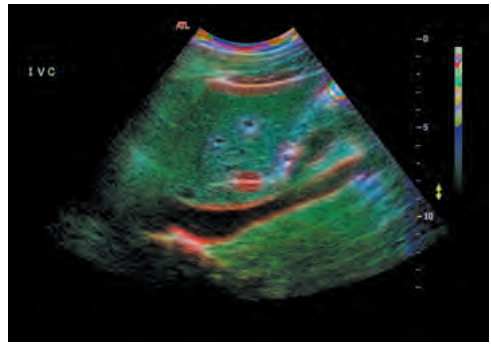


FIGURE 8.8 Composite image of the ultrasound image in Figure 8.2. The red color channel represents clustering, the green channel represents contrast, and the blue channel represents inertia.

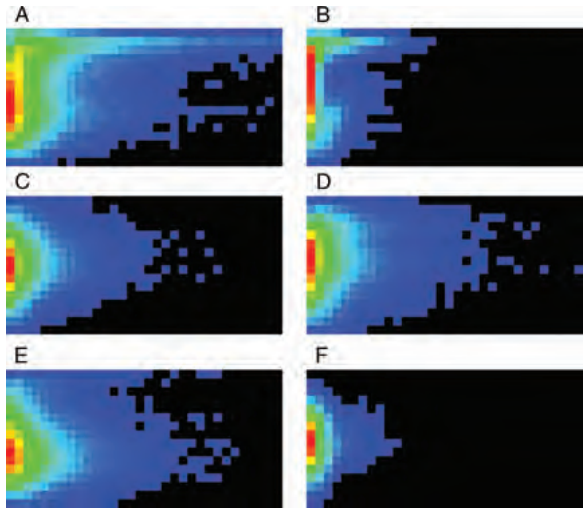


FIGURE 8.14 Run-length histograms of some of the textures in Figure 8.1: (A) corduroy at 0° scan angle, (B) corduroy at 90° scan angle, (C) glass, (D) carpet, (E) bark, and (F) knit, C–F at 0° scan angle. The histograms have been contrast-enhanced and false-colored for better visual representation. Black represents zero; blue, low values; green, intermediate values; and red, high values. Each histogram shows increasing gray values (16 bins) from top to bottom and increasing run lengths (32 bins) from left to right.

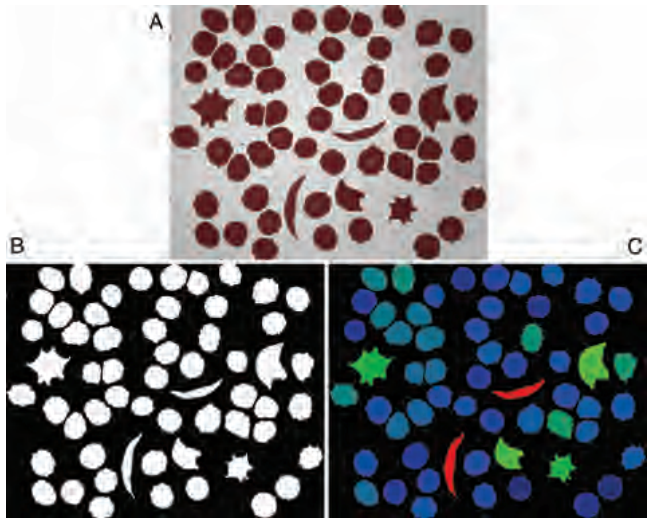


FIGURE 9.1 Example of shape analysis and classification. Image A shows various red blood cells (synthetically generated image) with three types of deformation. Each deformed cell occurs twice in the image, with a different size and rotation. Image B is the segmented (binarized) image, obtained through thresholding, and image C was obtained by quantifying the irregularity of the shape and using a false-color representation of the shape metric.

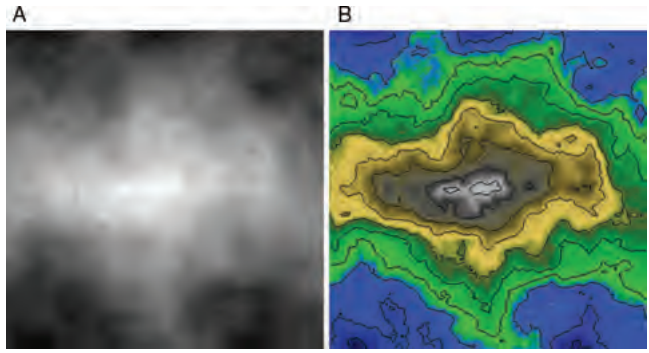


FIGURE 10.7 Landscape generation using the iterative midpoint displacement technique. A shows the random displacements as gray values (light gray areas have a higher elevation than dark gray areas). With suitable false coloring and added contour lines, a topographical map is created (B).

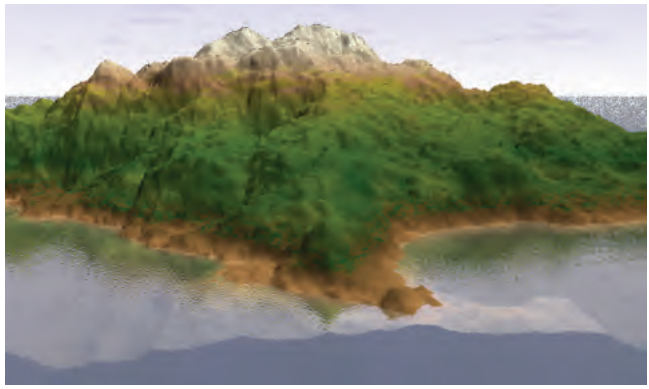


FIGURE 10.8 Three-dimensional rendering of the topographical map shown in Figure 10.7, with elevation-dependent colors, texture-mapped sea level, and texture-mapped sky added.

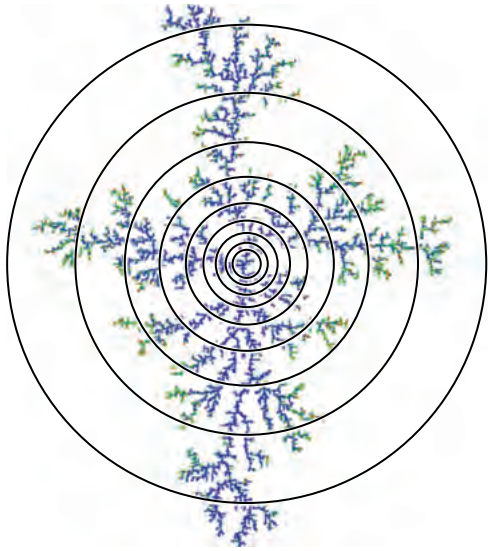


FIGURE 10.15 Determining the mass dimension of a simulated aggregation cluster. The mass (nonbackground pixels) of the aggregate inside the circles of increasing radius is counted, and the slope of the mass over the radius is determined on a double-logarithmic scale.

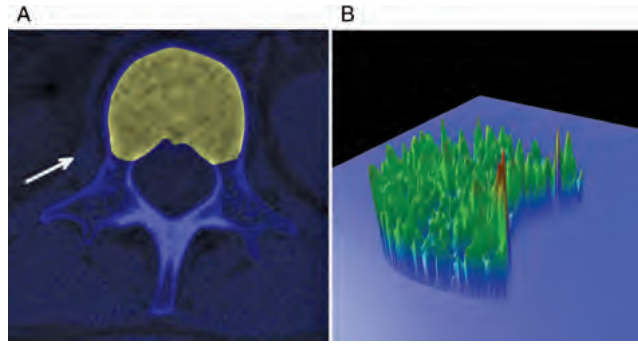


FIGURE 10.17 Elevation landscape representation of the segmented spongy area of a vertebral CT image. The left panel shows one cross-sectional CT slice through the vertebra with the spongy area highlighted. The right image is the corresponding elevation map. The white arrow indicates the view direction.

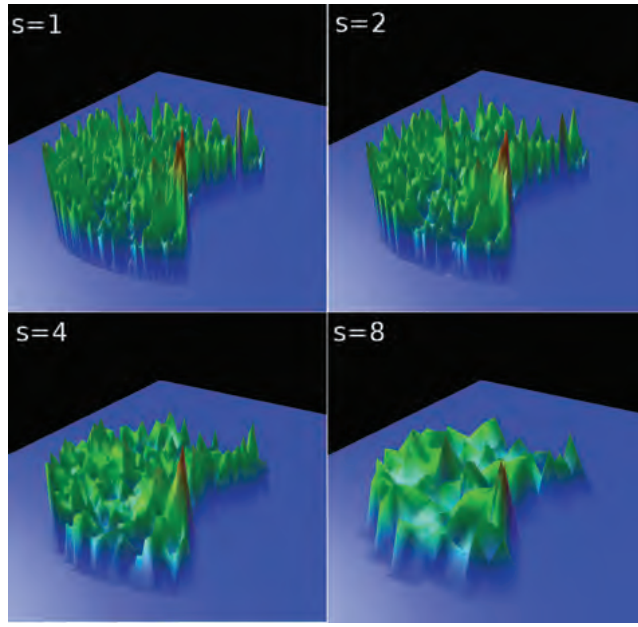


FIGURE 10.18 Four steps in the process of calculating the blanket dimension. Starting with the top left image, the landscape is drawn with one pixel per box ($s = 1$), whereas in subsequent iterations 4 pixels ($s = 2$), 16 pixels ($s = 4$), and 64 pixels ($s = 8$) are averaged. As a consequence, the elevation landscape becomes less jagged and the surface area decreases.

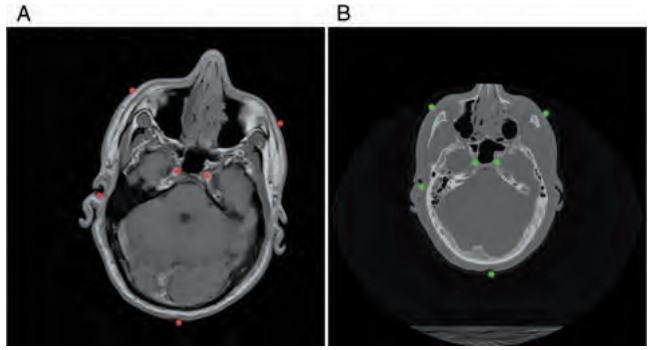


FIGURE 11.3 Matched but unregistered slices from the Visible Human data set. (A) shows a T_1>-weighted MR slice of the head, and (B) shows the corresponding CT slice. The images are mismatched in position, rotational orientation, and scale. Anatomy-based homologous fiducial markers (red in the MR image and green in the CT image) have been placed manually.

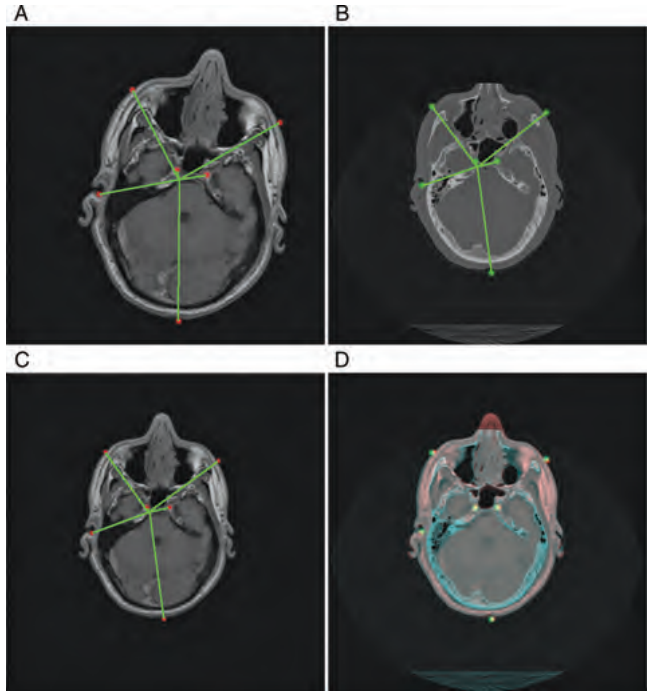


FIGURE 11.4 Registration process based on fiducial markers. First, the centroid of the markers is determined (green lines in A and B). Next, the centroids are brought in congruence with a translation operation. The angles of the green lines provide the necessary rotation angle, and the length ratio provides the scaling factor. After the transformation, the MR image (C) matches the CT image (B) in terms of position, rotation, and scale. It can be seen that the green lines, which connect the fiducial markers with the centroid, have the same length and orientation in (B) and (C). The images are then fused (D), with the MR image having a red tint and the CT image having a blue tint. A minor mismatch can still be seen; this mismatch cannot be corrected with rigid-body transformations.

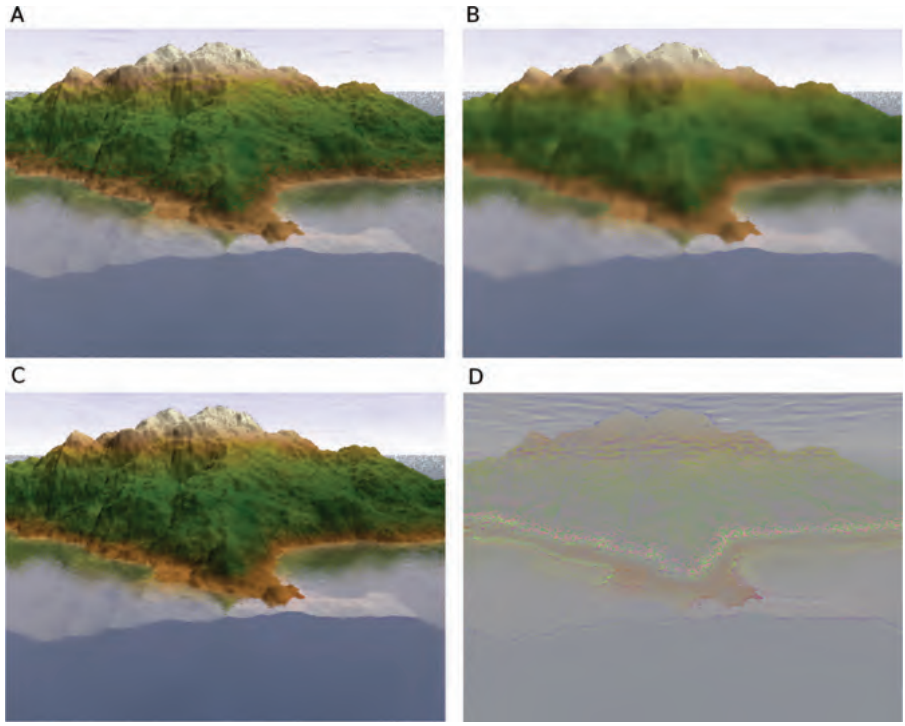


FIGURE 12.8 Demonstration of the effect of selective image information loss. Image A, subjected to an adaptive anisotropic diffusion lowpass filter, is strongly blurred (B). The loss of detail at high spatial frequencies is obvious. Conversely, when only the color (hue and saturation) information is blurred and the intensity information is left unaltered (C), almost no differences to the original image are detectable. The difference between images A and C is shown in image D.

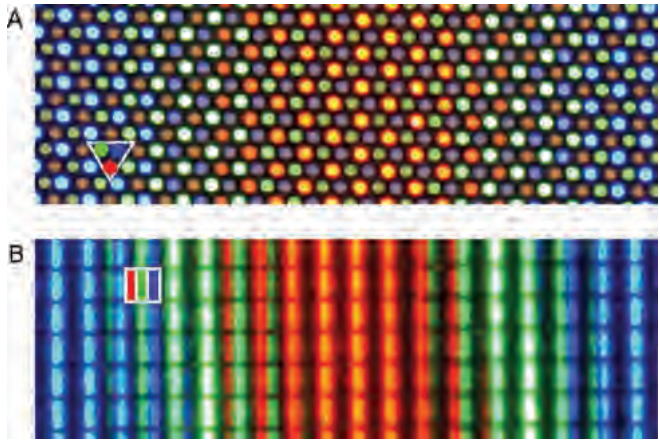


FIGURE 13.2 Demonstration of additive color generation. The images are magnified sections of photographs of a CRT screen (A) and a LCD screen (B), both displaying a color gradient blue–green–red–green–blue. In each image, one pixel is delineated (white triangle and square, respectively). Each pixel is approximately 0.25 mm in size. It can be seen that each pixel is composed of three components: red, green, and blue. Note that the camera that was used to take these photos has a high sensitivity for green, and the brightest green segments are overexposed.

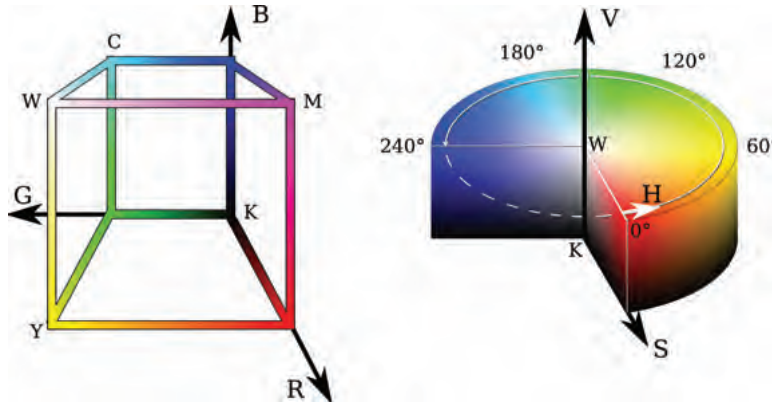


FIGURE 13.3 Graphical representations of the RGB and HSV color models. Each color represents one point inside the RGB cube, and its position is determined by the value of its R, G, and B components in Cartesian coordinates. Here, only the edges of the filled cube are shown. The edges of the cube give an example of the color gradients with black (k) and white (w) at diagonally opposite ends, and with the additive colors cyan (c), magenta (m), and yellow (y) at off-axis corners. The HSV model uses a cylinder coordinate system with the hue (H) as the angular component, the saturation (S) as the radial component, and the value (V, also known as intensity or brightness) as the axial component. The central axis of the cylinder covers the gray-value gradient from black to white. (HSV pie from http://en.wikipedia.org/wiki/HSL_and_HSV.)

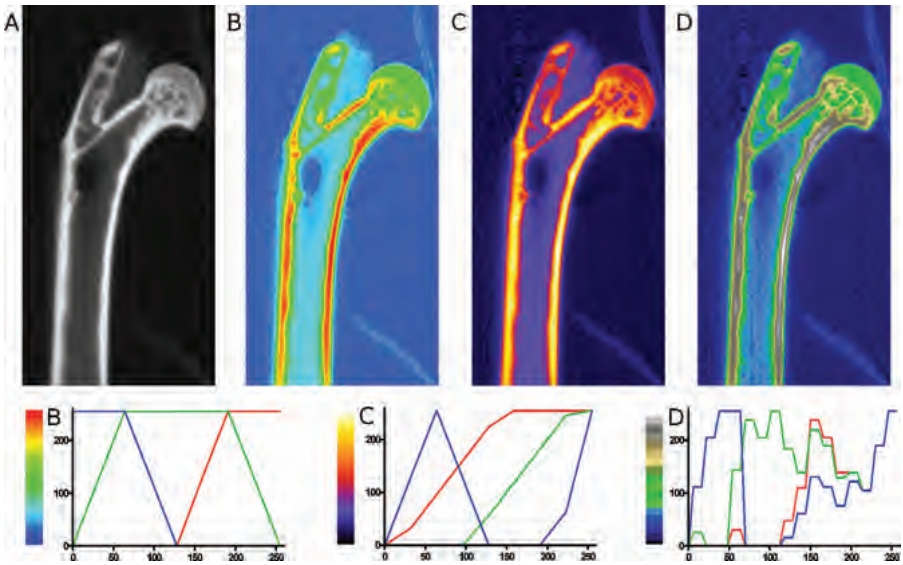


FIGURE 13.4 Gray-scale image of a micro-CT cross section of a mouse femur (A) and three false-color representations (B–D). In (B), the color scheme called Rainbow [Equation (13.5)] was applied, in (C) the color scheme called Fire, and in (D) a color scheme called Terra. The graphs underneath the color images are the corresponding lookup tables (the *x*-axis is the image gray value and the *y*-axis is the color component value). Whereas smooth transitions were desired in images B and C, the Terra color scheme in image D introduces abrupt transitions, which cause the appearance of contour bands.

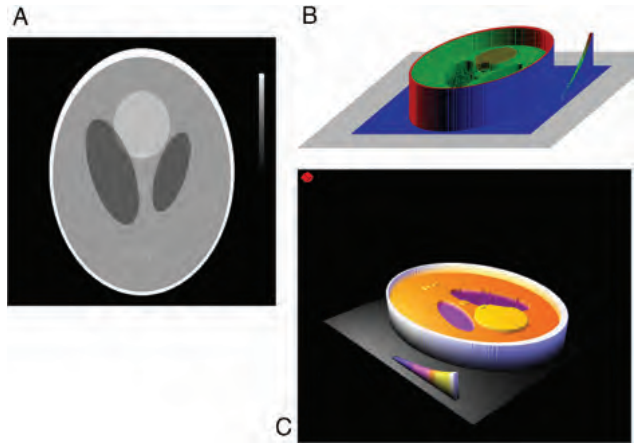


FIGURE 13.7 Examples of elevation landscape rendering. The source image (A) is the Shepp–Logan head phantom, with some additive noise in the left ventricle. A scale bar was placed to the right of the phantom. Rendering B was performed using the home-made isometric view that served as this chapter's example (Figure 13.6), and rendering C was performed with OpenGL. The red cube in the top left corner of image C is the light source. Elevation landscape rendering is particularly useful for recognizing shapes: From the original image, the intensity function in the scale bar at the right image edge cannot readily be determined. In the landscape renderings, however, the curved nature becomes obvious, and an anti-gamma function can be recognized.

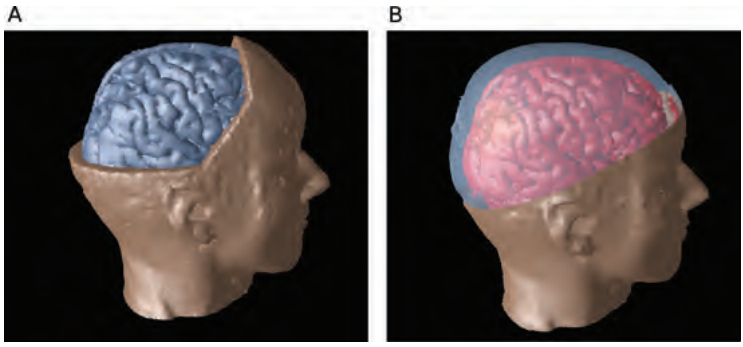


FIGURE 13.10 Visualization of the brain inside the head. The brain and the head were prepared as separate objects and placed in the scene for rendering. In image A, a section of the head is removed in the head image, whereby the brain is exposed in the removed section. In image B, a clip plane separates the upper section of the skull form the lower, and transparency was used in the upper section.

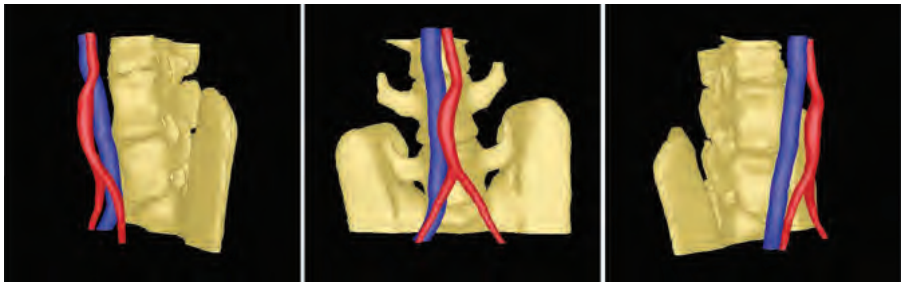


FIGURE 13.12 Three frames from the primate blood vessel animation. Aorta and vena cava are boldly colored (red and blue, respectively), while the spine was assigned a color with less saturation. Although some a priori knowledge was used for segmentation, the course of the blood vessels and the unusual shape of the aorta are clearly visible.

FIGURE 13.13 Training of medical students with a virtual patient. The setting is an ophthalmologist's office, and the patient can be asked to perform various tasks, such as telling the number of fingers indicated, or following the hand with his eyes. In this simulation, the hand is controlled by the medical student by means of a three-dimensional positioning device. Simple clicks make it possible to extend a specific number of fingers. (Courtesy of Dr. Kyle Johnsen.)

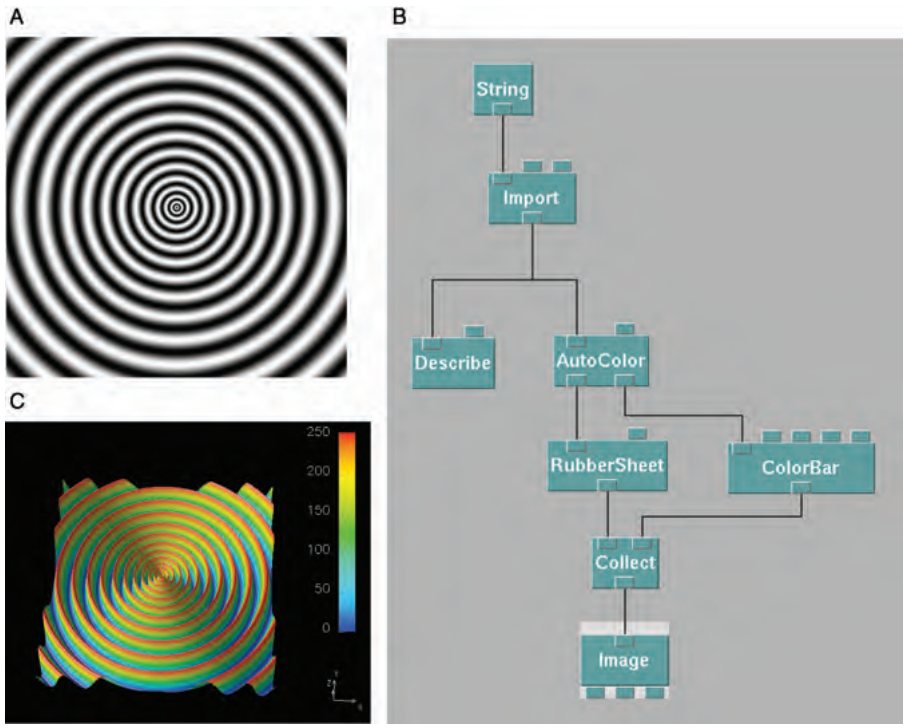


FIGURE 14.6 Example of an OpenDX visual program. The input image (A) exists in native DX format as created by Crystal Image. In the visual program (B), the `String` specifies the file name, and the `Import` operator converts the image into a field. With `AutoColor`, color is being added to the nodes of the field. `Rubbersheet` creates the elevation map. Finally, the `Image` operator renders the object (C). The `ColorBar` operator is responsible for adding the color scale bar to image C, and the `Describe` operator describes the data object at the point where it is connected.

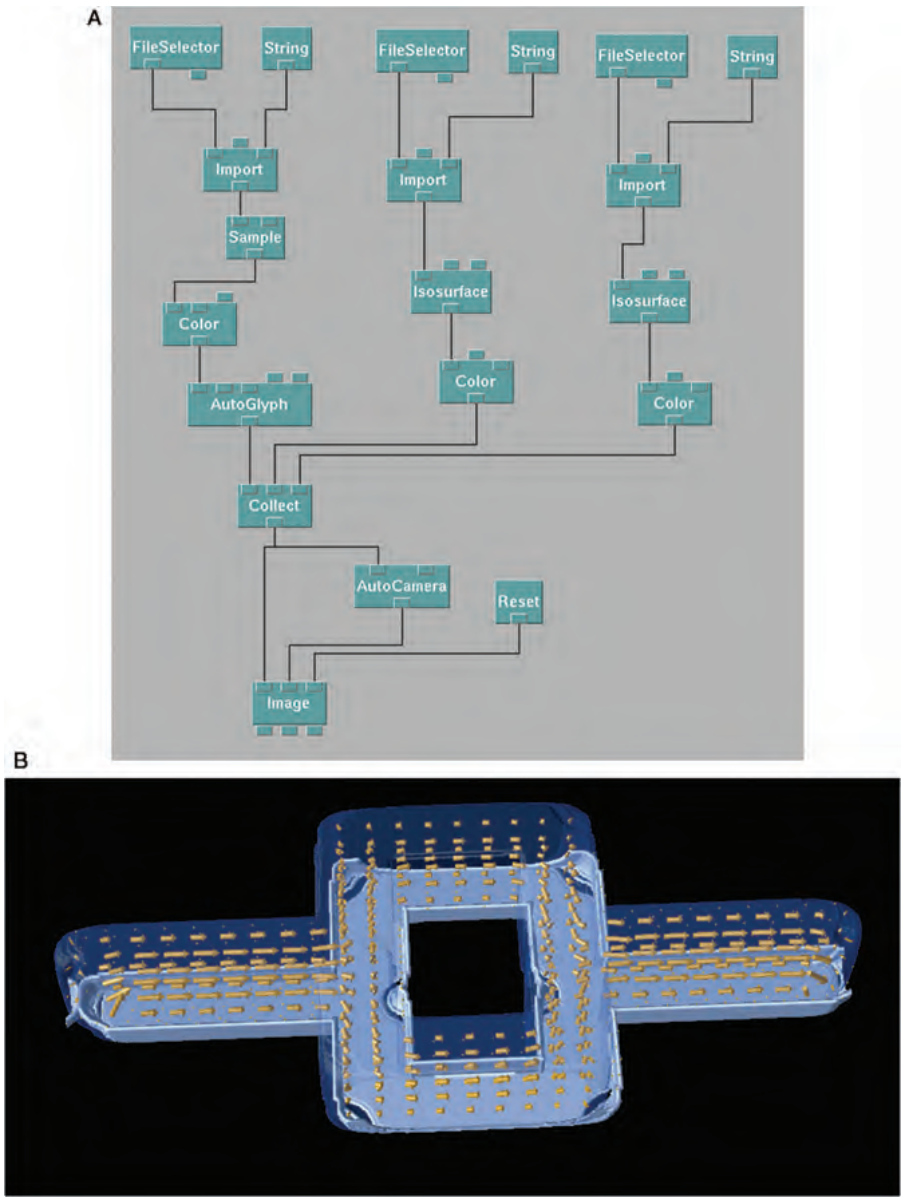


FIGURE 14.8 Visualization of a vector field. The visual program (A) is relatively complex because it merges three components: the vector field itself, visualized with the `AutoGlyphs` operator, and two scalar fields that contain the outline of the flow chamber, visualized as isosurfaces. The lower isosurface (about one-third of the chamber) was made nontransparent, whereas the upper part of the flow chamber was made semitransparent.