CHAPTER

Introduction

1

CHAPTER OUTLINE HEAD

1.1 Overview

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

Table 1.1 Overview of Chapter 1

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

1.2 Human and computer vision

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

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

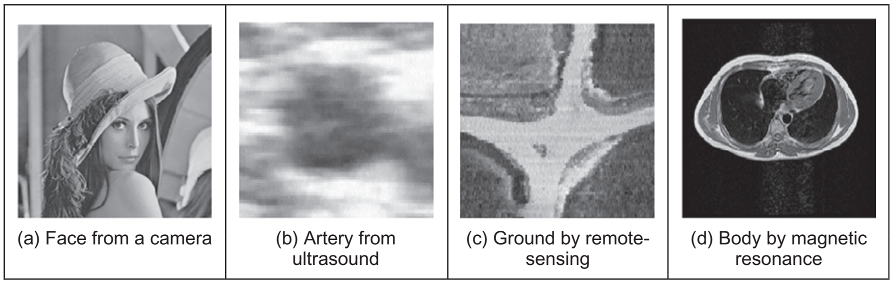

(a) Face from a camera (b) Artery from ultrasound (c) Ground by remote-sensing (d) Body by magnetic resonance

FIGURE 1.1

Real images from different sources.

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

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

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

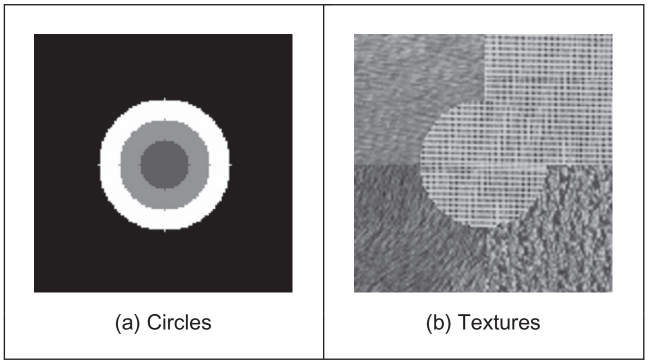


FIGURE 1.2

Examples of synthesized images.

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

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

1.3 The human vision system

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

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

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

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

1.3.1 The eye

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

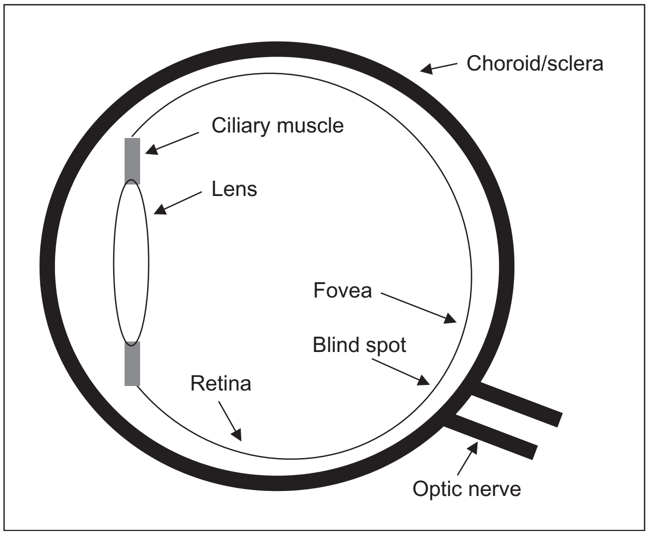


FIGURE 1.3

Human eye.

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

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

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

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

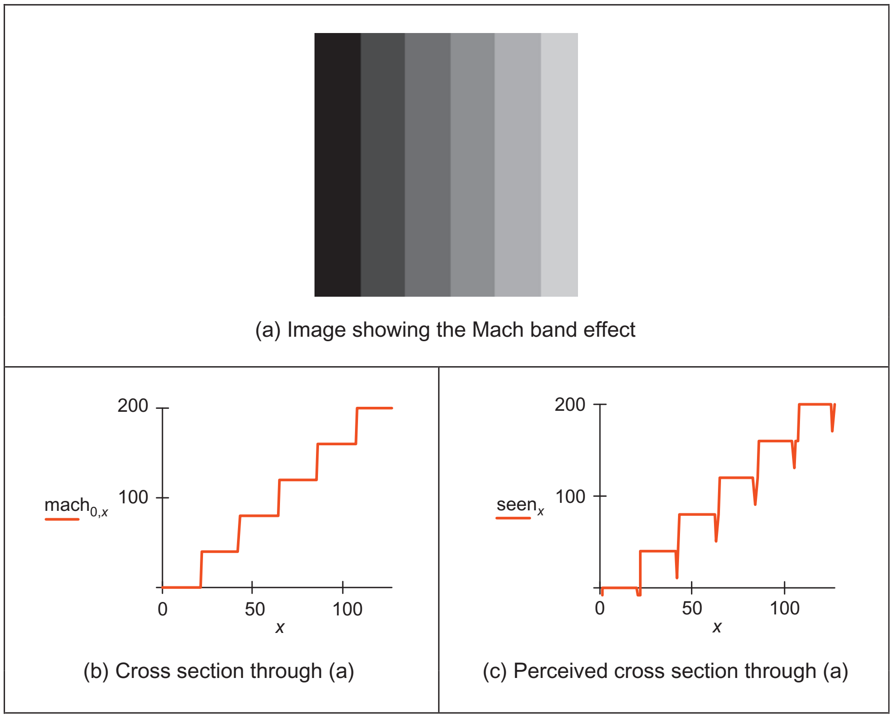


FIGURE 1.4

Illustrating the Mach band effect.

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

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

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

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

1.3.2 The neural system

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

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

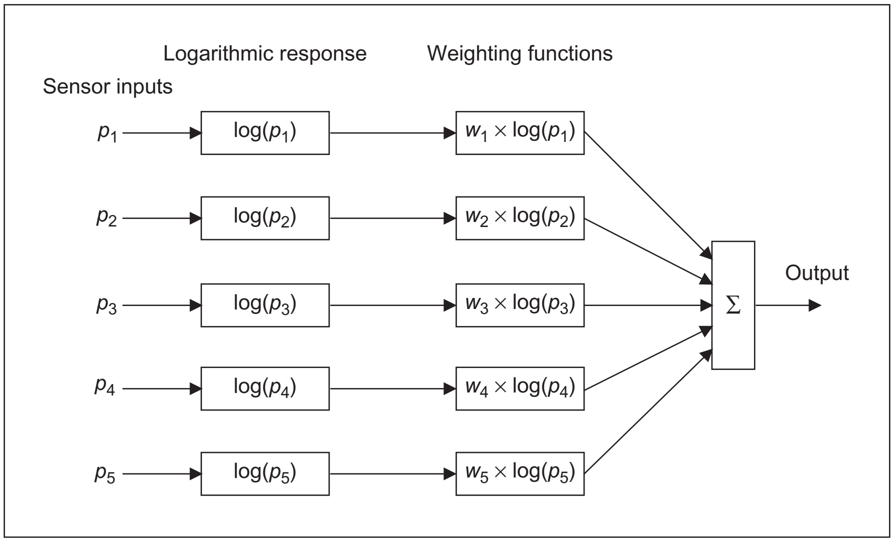


FIGURE 1.5

Neural processing.

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

1.3.3 Processing

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

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

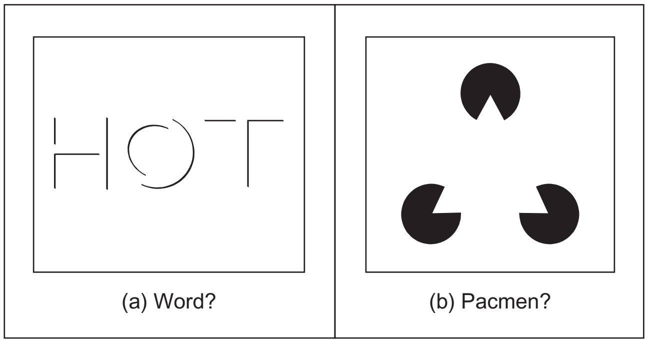


FIGURE 1.6

How human vision uses edges.

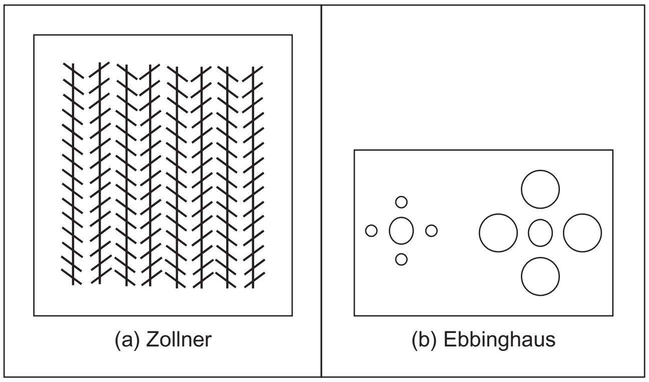


FIGURE 1.7

Static illusions.

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

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

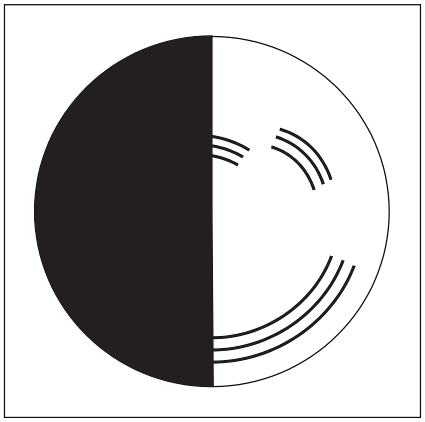


FIGURE 1.8

Benham's disk.

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

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

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

1.4 Computer vision systems

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

1.4.1 Cameras

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

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

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

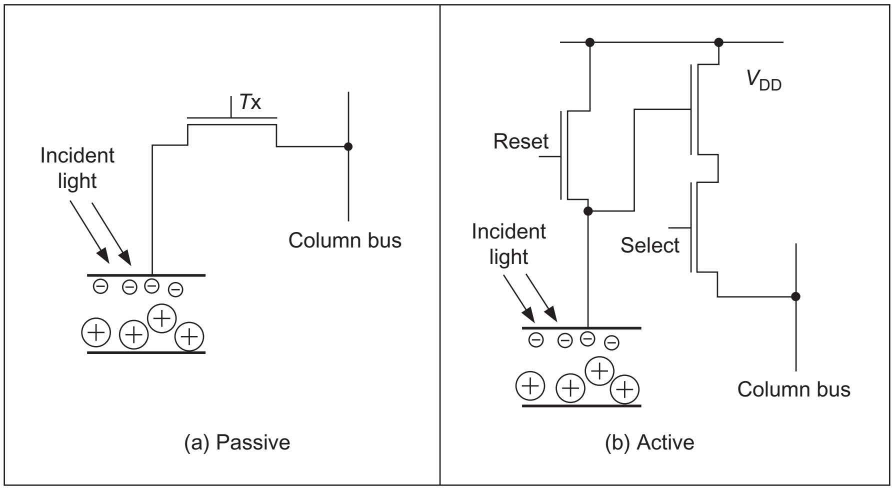


FIGURE 1.9

Pixel sensors.

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

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

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

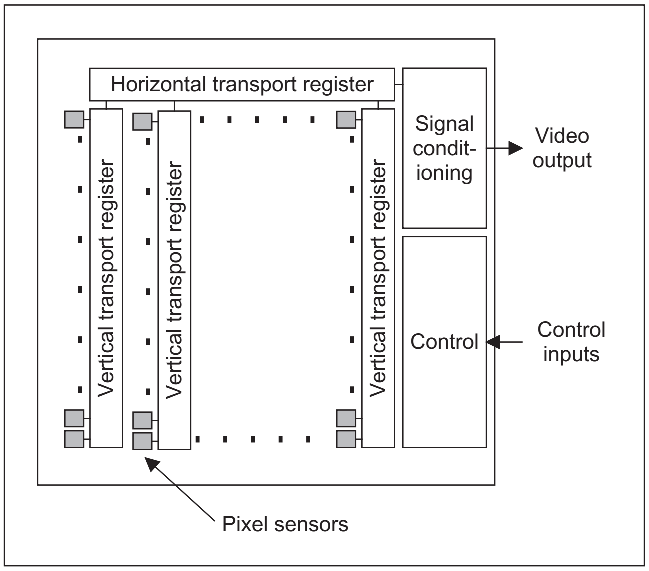

Horizontal transport register
Vertical transport register
Vertical transport register
Vertical transport register
Signal condit-ioning
Video output
Control
Control inputs
Pixel sensors

FIGURE 1.10

CCD sensing element.

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

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

1.4.2 Computer interfaces

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

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

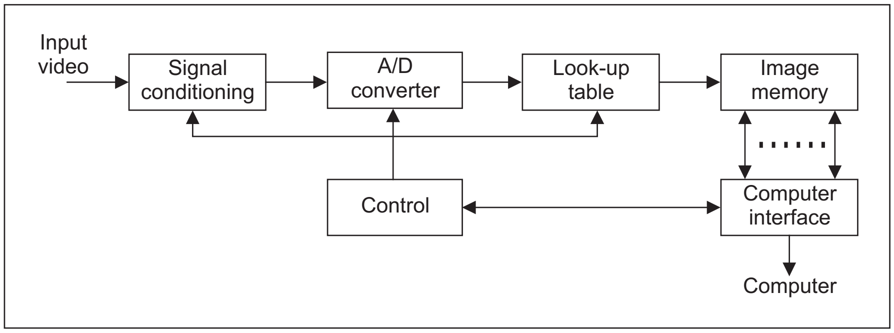


FIGURE 1.11

A computer interface—a framegrabber.

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

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

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

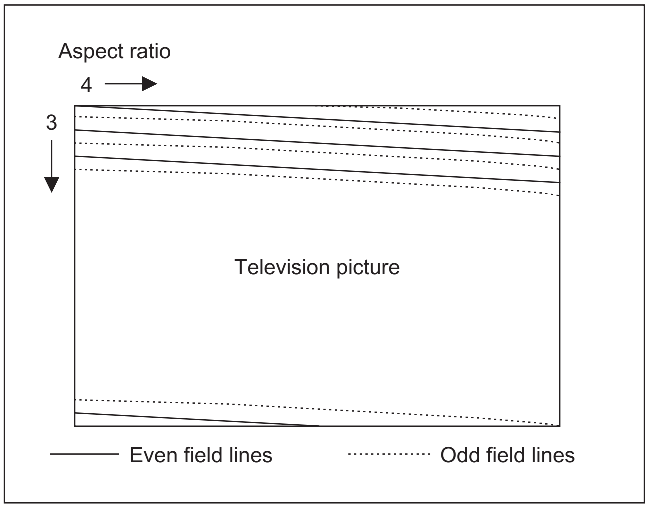


FIGURE 1.12

Interlacing in television pictures.

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

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

1.4.3 Processing an image

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

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

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

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

Table 1.2 Software Web Sites

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

1.5 Mathematical systems

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

1.5.1 Mathematical tools

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

Table 1.3 Mathematical Package Web Sites

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

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

1.5.2 Hello Matlab, hello images!

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

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

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

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

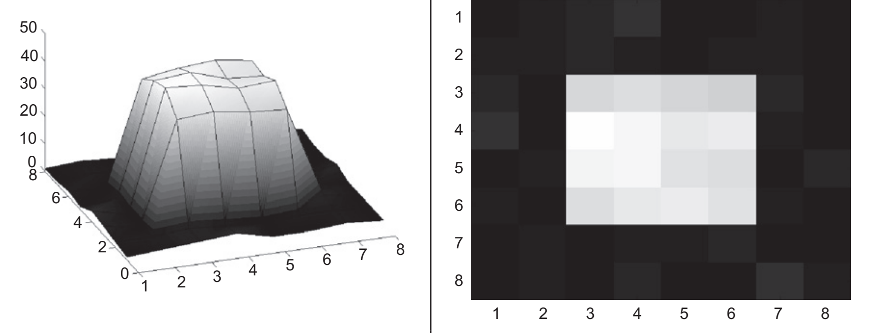
Matlab processes images, so we need to know what an image represents. Images are spatial data, data that is indexed by two spatial coordinates. The camera senses the *brightness* at a point with coordinates x,y. Usually, x and y refer to the horizontal and vertical axes, respectively. Throughout this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

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

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

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

(a) Set of pixel values



(b) Matlab surface plot

(c) Matlab image

FIGURE 1.13

Matlab image visualization.

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

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

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

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

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

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

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

CODE 1.1

Matlab script for Chapter 1.

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

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

CODE 1.1

(Continued)

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

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

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

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

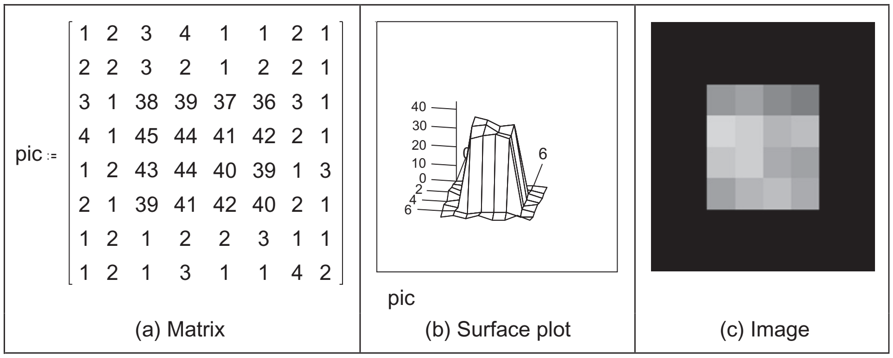
CODE 1.2

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

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

1.5.3 Hello Mathcad!

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



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

FIGURE 1.14

Synthesized image of a square.

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

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

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

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

CODE 1.3

Accessing an image in Mathcad.

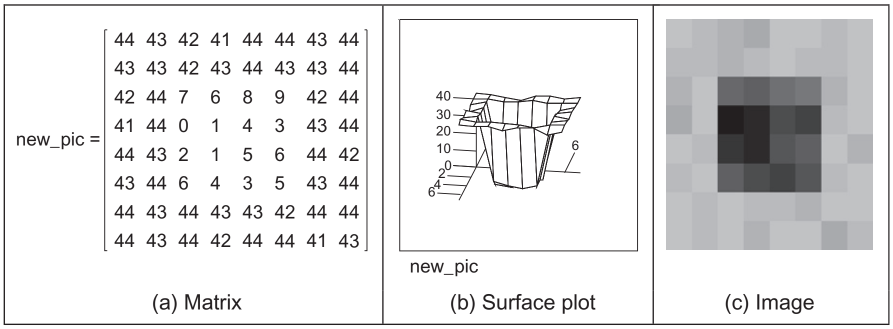
This synthetic image can be processed using the Mathcad programming language, which can be invoked by selecting the appropriate dialog box. This allows for conventional `for`, `while`, and `if` statements, and the earlier assignment operator, which is := in noncode sections, is replaced by ← in sections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

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

CODE 1.4

Processing image points in Mathcad.

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



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

FIGURE 1.15

Image of a square after division.

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

```
add_value(inpic,value):= for x∈0..cols(inpic)-1
                           for y∈0..rows(inpic)-1
                             newpicture_{y,x}←inpic_{y,x}+value
                         newpicture
```

CODE 1.5

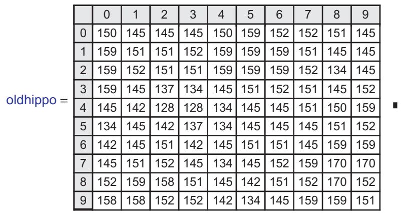
Function to add a value to an image in Mathcad.

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

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

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

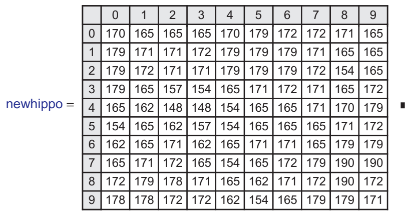
So if we are to make an image brighter (by addition) by the routine in Code 1.5 via Code 1.6 then we achieve the result shown in Figure 1.16. The matrix listings in Figure 1.16(a) and (b) shows that 20 has been added to each point (these show only the top left-hand section of the image where the bright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



oldhippo =

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

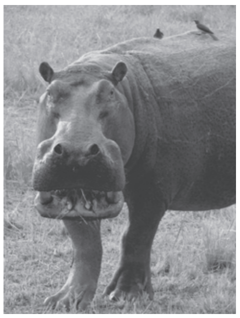
(a) Part of original image as a matrix



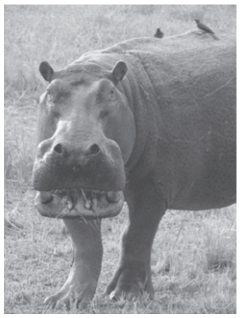
newhippo =

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

(b) Part of processed image as a matrix



(c) Bitmap of original image



(d) Bitmap of processed image

FIGURE 1.16

Processing an image.

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

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

CODE 1.6

Processing an image.

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

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

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

CODE 1.7

Creating the image of Figure 1.4(a).

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

1.6 Associated literature

1.6.1 Journals, magazines, and conferences

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

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

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

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

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

1.6.2 Textbooks

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

Table 1.4 Web Textbooks and Homepages

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

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

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

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

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

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

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

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

1.6.3 The Web

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

1.7 Conclusions

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

1.8 References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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