2
Formation

All digital images have to originate from somewhere. In this chapter we consider the issue of image formation both from a mathematical and an engineering perspective. The origin and characteristics of an image can have a large bearing on how we can effectively process it.

2.1 How is an image formed?

The image formation process can be summarized as a small number of key elements. In general, a digital image s can be formalized as a mathematical model comprising a functional representation of the scene (the object function o) and that of the capture process (the point-spread function (PSF) p). Additionally, the image will contain additive noise n. These are essentially combined as follows to form an image:

$$\begin{aligned} \text{Image} &= \text{PSF} * \text{object function} + \text{noise} \\ s &= p * o + n \end{aligned} \tag{2.1}$$

In this process we have several key elements:

- *PSF* this describes the way information on the object function is spread as a result of recording the data. It is a characteristic of the imaging instrument (i.e. camera) and is a deterministic function (that operates in the presence of noise).

- *Object function* This describes the object (or scene) that is being imaged (its surface or internal structure, for example) and the way light is reflected from that structure to the imaging instrument.

- *Noise* This is a nondeterministic function which can, at best, only be described in terms of some statistical noise distribution (e.g. Gaussian). Noise is a stochastic function which is a consequence of all the unwanted external disturbances that occur during the recording of the image data.

- *Convolution operator* $*$ A mathematical operation which 'smears' (i.e. convolves) one function with another.

Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab Chris Solomon and Toby Breckon
 ISBN: 978-0-470-84472-4

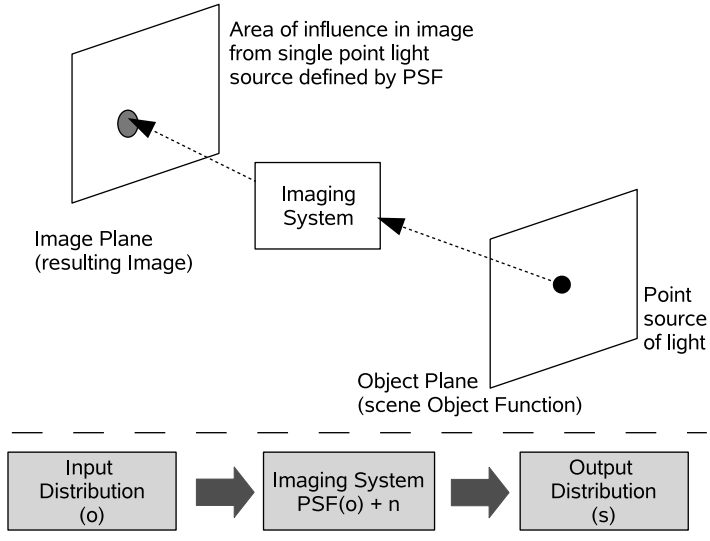


Figure 2.1 An overview of the image formation 'system' and the effect of the PSF

Here, the function of the light reflected from the object/scene (*object function*) is transformed into the image data representation by convolution with the *PSF*. This function characterizes the image formation (or capture) process. The process is affected by *noise*.

The PSF is a characteristic of the imaging instrument (i.e. camera). It represents the response of the system to a point source in the object plane, as shown in Figure 2.1, where we can also consider an imaging system as an input distribution (scene light) to output distribution (image pixels) mapping function consisting both of the PSF itself and additive noise (Figure 2.1 (lower)).

From this overview we will consider both the mathematical representation of the image formation process (Section 2.2), useful as a basis for our later consideration of advanced image-processing representations (Chapter 5), and from an engineering perspective in terms of physical camera imaging (Section 2.3).

2.2 The mathematics of image formation

The formation process of an image can be represented mathematically. In our later consideration of various processing approaches (see Chapters 3–6) this allows us to reason mathematically using knowledge of the conditions under which the image originates.

2.2.1 Introduction

In a general mathematical sense, we may view image formation as a process which transforms an *input distribution* into an *output distribution*. Thus, a simple lens may be viewed as a 'system' that transforms a spatial distribution of light in one domain (the object

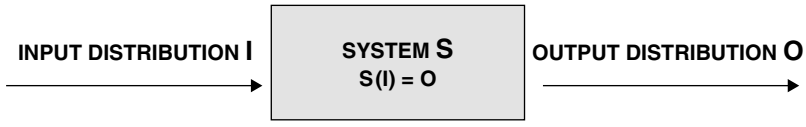


Figure 2.2 Systems approach to imaging. The imaging process is viewed as an operator *S* which acts on the input distribution *I* to produce the output *O*

plane) to a distribution in another (the image plane). Similarly, a medical ultrasound imaging system transforms a set of spatially distributed acoustic reflection values into a corresponding set of intensity signals which are visually displayed as a grey-scale intensity image. Whatever the specific physical nature of the input and output distributions, the concept of a *mapping* between an input distribution (the thing you want to investigate, see or visualize) and an output distribution (what you actually measure or produce with your system) is valid. The systems theory approach to imaging is a simple and convenient way of conceptualizing the imaging process. Any imaging device is a system, or a 'black box', whose properties are defined by the way in which an input distribution is mapped to an output distribution. Figure 2.2 summarizes this concept.

The process by which an imaging system transforms the input into an output can be viewed from an alternative perspective, namely that of the Fourier or *frequency domain*. From this perspective, images consist of a superposition of harmonic functions of different frequencies. Imaging systems then act upon the spatial frequency content of the input to produce an output with a modified spatial frequency content. Frequency-domain methods are powerful and important in image processing, and we will offer a discussion of such methods later in Chapter 5. First however, we are going to devote some time to understanding the basic mathematics of image formation.

2.2.2 Linear imaging systems

Linear systems and operations are extremely important in image processing because the majority of real-world imaging systems may be well approximated as *linear systems*. Moreover, there is a thorough and well-understood body of theory relating to linear systems. Nonlinear theory is still much less developed and understood, and deviations from strict linearity are often best handled in practice by approximation techniques which exploit the better understood linear methods.

An imaging system described by operator *S* is *linear* if for any two input distributions *X* and *Y* and any two scalars *a* and *b* we have

$$S\{aX + bY\} = aS\{X\} + bS\{Y\} \tag{2.2}$$

In other words, applying the linear operator to a weighted sum of two inputs yields the same result as first applying the operator to the inputs independently and then combining the weighted outputs. To make this concept concrete, consider the two simple input

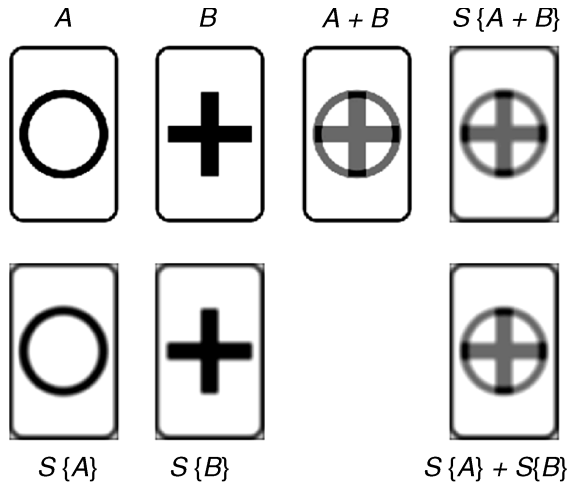


Figure 2.3 Demonstrating the action of a linear operator *S*. Applying the operator (a blur in this case) to the inputs and then linearly combining the results produces the same result as linearly combining them and then applying the operator

distributions depicted in Figure 2.3 consisting of a cross and a circle. These radiate some arbitrary flux (optical photons, X-rays, ultrasonic waves or whatever) which are imperfectly imaged by our linear system. The first row of Figure 2.3 shows the two input distributions, their sum and then the result of applying the linear operator (a blur) to the sum. The second row shows the result of first applying the operator to the individual distributions and then the result of summing them. In each case the final result is the same. The operator applied in Figure 2.3 is a *convolution* with Gaussian blur, a topic we will expand upon shortly.

2.2.3 Linear superposition integral

Consider Figure 2.4, in which we have some general 2-D input function $f(x', y')$ in an input domain (x', y') and the 2-D response $g(x, y)$ of our imaging system to this input in the output domain (x, y). In the most general case, we should allow for the possibility that each and every point in the input domain may contribute in some way to the output. If the system is linear, however, the contributions to the final output must combine linearly. For this reason, basic linear image formation is described by an integral operator which is called the *linear superposition integral*:

$$g(x, y) = \int\int f(x', y')h(x, y; x', y')\,\mathrm{d}x'\,\mathrm{d}y' \tag{2.3}$$

This integral actually expresses something quite simple. To understand this, consider some arbitrary but specific point in the output domain with coordinates (x, y). We wish to know

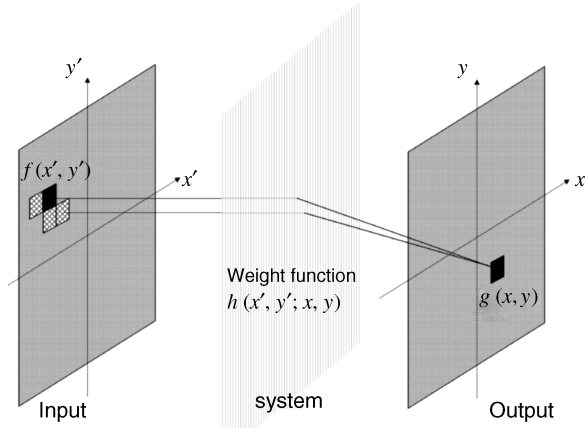


Figure 2.4 The linear superposition principle. Each point in the output is given by a weighted sum (integral) of all input points. The weight function is called the point-spread function and specifies the contribution of each and every input point to each and every output point

$g(x, y)$, the output response at that point. The linear superposition integral can be understood by breaking it down into three steps:

(1) Take the value of the input function f at some point in the input domain (x', y') and multiply it by some weight h, with h determining the amount by which the input flux at this particular point contributes to the output point.

(2) Repeat this for each and every valid point in the input domain multiplying by the appropriate weight each time.

(3) Sum (i.e. integrate) all such contributions to give the response $g(x, y)$.

Clearly, it is the weighting function h which determines the basic behaviour of the imaging system. This function tells us the specific contribution made by each infinitesimal point in the input domain to each infinitesimal point in the resulting output domain. In the most general case, it is a function of *four* variables, since we must allow for the contribution of a given point (x', y') in the input space to a particular point (x, y) in the output space to depend on the exact location of *both points*. In the context of imaging, the weighting function h is referred to as the *point-spread function* (PSF). The reason for this name will be demonstrated shortly. First, however, we introduce an important concept in imaging: the (Dirac) delta or impulse function.

2.2.4 The Dirac delta or impulse function

In image processing, the delta or impulse function is used to represent mathematically a bright intensity source which occupies a very small (infinitesimal) region in space. It can be

modelled in a number of ways,[1] but arguably the simplest is to consider it as the limiting form of a scaled rectangle function as the width of the rectangle tends to zero. The 1-D rectangle function is defined as

$$\begin{aligned} \text{rect}\left(\frac{x}{a}\right) &= 1 \quad |x| < a/2 \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{2.4}$$

Accordingly, the 1-D and 2-D delta function can be defined as

$$\begin{aligned} \delta(x) &= \lim_{a \to 0} \frac{1}{a} \text{rect}\left(\frac{x}{a}\right) && \text{in 1-D} \\ \delta(x,y) &= \lim_{a \to 0} \frac{1}{a^2} \text{rect}\left(\frac{x}{a}\right) \text{rect}\left(\frac{y}{a}\right) && \text{in 2-D} \end{aligned} \tag{2.5}$$

In Figure 2.5 we show the behaviour of the scaled rectangle function as $a \to 0$. We see that:

As $a \to 0$ the support (the nonzero region) of the function tends to a vanishingly small region either side of $x = 0$.

As $a \to 0$ the height of the function tends to infinity but the *total area under the function* remains equal to one.

$\delta(x)$ is thus a function which is zero everywhere except at $x = 0$ precisely. At this point, the function tends to a value of infinity but retains a finite (unit area) under the function.

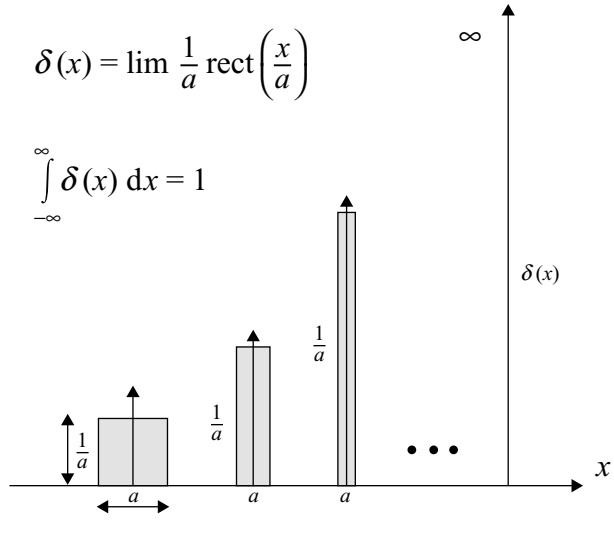


Figure 2.5 The Dirac delta or impulse function can be modelled as the limiting form of a scaled rectangle function as its width tends to zero. Note that the area under the delta function is equal to unity

[1] An alternative is the limiting form of a Gaussian function as the standard deviation $\sigma \to 0$.

Thus:

$$\begin{aligned} \delta(x) &= \infty \quad x = 0 \\ &= 0 \quad x \neq 0 \end{aligned} \tag{2.6}$$

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1 \tag{2.7}$$

and it follows that a shifted delta function corresponding to an infinitesimal point located at $x = x_0$ is defined in exactly the same way, as

$$\begin{aligned} \delta(x{-}x_0) &= \infty \quad x = x_0 \\ &= 0 \quad x \neq x_0 \end{aligned} \tag{2.8}$$

These results extend naturally to two dimensions and more:

$$\begin{aligned} \delta(x,y) &= \infty \quad x = 0, y = 0 \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{2.9}$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(x,y)\,\mathrm{d}x\,\mathrm{d}y = 1 \tag{2.10}$$

However, the most important property of the delta function is *defined by* its action under an integral sign, i.e. its so-called *sifting property*. The *sifting theorem* states that

$$\begin{aligned} &\int_{-\infty}^{\infty} f(x)\delta(x{-}x_0)\,\mathrm{d}x = f(x_0) && \text{1-D case} \\ &\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(x{-}x_0, y{-}y_0)\,\mathrm{d}x\,\mathrm{d}y = f(x_0,y_0) && \text{2-D case} \end{aligned} \tag{2.11}$$

This means that, whenever the delta function appears inside an integral, the result is equal to the remaining part of the integrand evaluated at those precise coordinates for which the delta function is nonzero.

In summary, the delta function is formally defined by its three properties of *singularity* (Equation (2.6)), *unit area* (Equation (2.10)) and the *sifting property* (Equation (2.11)).

Delta functions are widely used in optics and image processing as idealized representations of point and line sources (or apertures):

$$\begin{aligned} &f(x,y) = \delta(x{-}x_0, y{-}y_0) && \text{(point source located at } x_0, y_0\text{)} \\ &f(x,y) = \delta(x{-}x_0) && \text{(vertical line source located on the line } x = x_0\text{)} \\ &f(x,y) = \delta(y{-}y_0) && \text{(horizontal line source located on the line } y = y_0\text{)} \\ &f(x,y) = \delta(ax + by + c) && \text{(source located on the straight line } ax + by + c\text{)} \end{aligned} \tag{2.12}$$

2.2.5 The point-spread function

The point-spread function of a system is defined as the response of the system to an input distribution consisting of a very small intensity point. In the limit that the point becomes infinitesimal in size, we can represent the input function as a Dirac delta function $f(x', y') = \delta(x'-x_0, y'-y_0)$. For a linear system, we can substitute this into the linear superposition integral, Equation (2.3), to give

$$g(x, y) = \int\int \delta(x'-x_0, y'-y_0) h(x, y; x', y')\, \mathrm{d}x'\, \mathrm{d}y' \tag{2.13}$$

By applying the sifting theorem of Equation (2.11) this gives a response $g(x, y)$ to the point-like input as

$$g(x, y) = h(x, y; x_0, y_0) \tag{2.14}$$

In other words, the response to the point-like input (sometimes termed the *impulse* response) is *precisely equal* to h. However, in the context of imaging, the weighting function in the superposition integral $h(x, y; x', y')$ is termed the point spread function or PSF because it measures the spread in output intensity response due to an infinitesimal (i.e. infinitely sharp) input point.

Why, then, is the PSF an important measure? Consider for a moment that any input distribution may be considered to consist of a very large ($\rightarrow \infty$) collection of very small ($\rightarrow$ infinitesimal) points of varying intensity. The PSF tells us what each of these points will look like in the output; so, through the linearity of the system, the output is given by the sum of the PSF responses. It is thus apparent that the *PSF of such a linear imaging system (in the absence of noise) completely describes its imaging properties* and, therefore, is of primary importance in specifying the behaviour of the system.

It also follows from what we have said so far that an ideal imaging system would possess a PSF that exactly equalled the Dirac delta function for all combinations of the input and output coordinates (i.e. all valid combinations of the coordinates (x, y, x', y')). Such a system would map each infinitesimal point in the input distribution to a corresponding infinitesimal point in the output distribution. There would be no blurring or loss of detail and the image would be 'perfect'. Actually, the fundamental physics of electromagnetic diffraction dictates that such an ideal system can never be achieved in practice, but it nonetheless remains a useful theoretical concept. By extension then, a good or 'sharp' imaging system will generally possess a *narrow* PSF, whereas a poor imaging system will have a broad PSF, which has the effect of considerably *overlapping* the output responses to neighbouring points in the input. This idea is illustrated in Figure 2.6, in which a group of points is successively imaged by systems with increasingly poor (i.e. broader) PSFs.

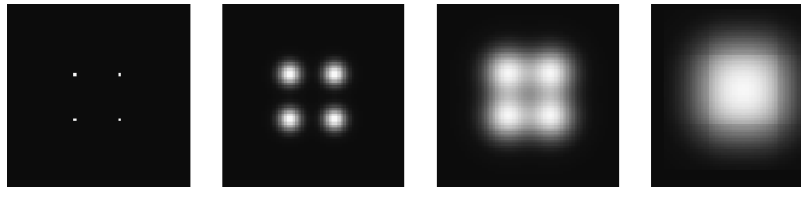


Figure 2.6 The effect of the system PSF. As the PSF becomes increasingly broad, points in the original input distribution become broader and overlap

2.2.6 Linear shift-invariant systems and the convolution integral

So far in our presentation, we have assumed that the system PSF may depend on the *absolute* locations of the two points in the input and output domains. In other words, by writing the PSF as a function of four variables in Equation (2.3), we have explicitly allowed for the possibility that the response at, let us say, point x_1, y_1 in the output domain due to a point x_1', y_1' in the input domain *may be quite different* from that at some other output point x_2, y_2. If this were the case in practice for all pairs of input and output points, then specification of the system PSF would be a very cumbersome business. As an example, consider discretizing both the input and output domains into even a modest number of pixels such as 128×128. Since every pixel in the input can in principle contribute to every pixel in the output in a different way, we would then require $128^4 \approx 268.4$ million values to fully specify the system behaviour.

Fortunately, most imaging systems (at least to a very good approximation) possess a property called *shift invariance* or *isoplanatism*. A shift-invariant system is characterized by a PSF that depends *only on the difference* between the coordinates in the input and output domains, not on their absolute values. Thus, a shift-invariant PSF in a 2-D imaging system has a functional form which makes it dependent not on four variables, but on two variables only:

$$h(x, y; x', y') = h(x'', y'') = h(x - x', y - y') \tag{2.15}$$

Shift invariance has a simple physical interpretation. Consider a source point in the input domain at coordinates x', y'. If we move this source point to different locations in the input domain, changing the values of the coordinates x', y', the corresponding response (i.e. its shape/functional form) in the output domain will remain the same but will simply be correspondingly translated in the output domain. In other words, *shift the input and the corresponding output just shifts too*. Figure 2.7[2] is a simulation of a photon-limited system in which photons in the input domain are randomly emitted and imaged by the linear shift-invariant system. This shows clearly how the image is built as the weighted superposition of shift-invariant responses to an increasing number of point-like inputs.

When we have a shift-invariant system, the linear superposition integral Equation (2.3) reduces to a simpler and very important form – a convolution integral:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x - x', y - y') \, \mathrm{d}x' \, \mathrm{d}y' \tag{2.16}$$

Note that the infinite limits are used to indicate that the integration is to take place over all values for which the product of the functions is nonzero. The output $g(x, y)$ is said to be given by the convolution of the input $f(x, y)$ with the PSF $h(x, y)$. Note that the corresponding form for convolution of 1-D functions follows naturally:

$$g(x) = \int_{-\infty}^{\infty} f(x') h(x - x') \, \mathrm{d}x' \tag{2.17}$$

[2] The Matlab code which produced Figure 2.7 is available at http://www.fundipbook.com/materials/.

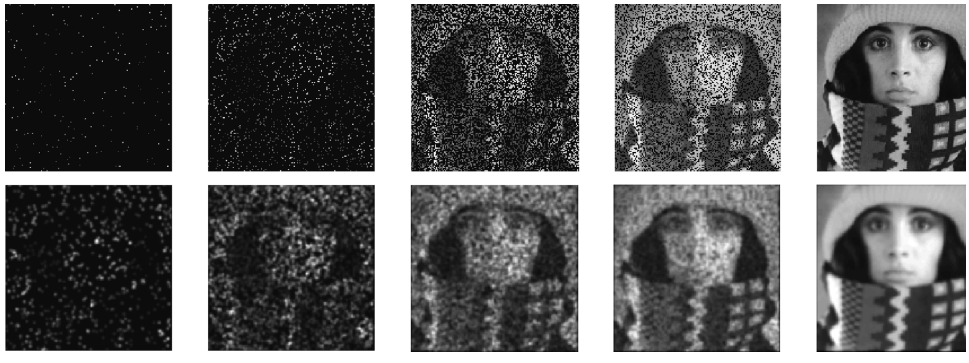


Figure 2.7 Each image in the top row shows arbitrary points in the input domain (increasing in number from left to right) whilst the images in the bottom row show the response of a LSI system to the corresponding input. The final image at the bottom right consists of a weighted superposition of responses to the input points, each of which has a similar mathematical shape or form

Convolution integrals are so important and common that they are often written in an abbreviated form:

$$\begin{aligned} g(x,y) &= f(x,y) ** h(x,y) \quad \text{(2-D)} \\ g(x) &= f(x) * h(x) \quad \text{(1-D)} \end{aligned} \tag{2.18}$$

where the asterisks denote the operation of convolving the input function f with the system PSF h. In general, the function h in the convolution integrals above is called the kernel.

2.2.7 Convolution: its importance and meaning

It would be hard to overstate the importance of convolution in imaging. There are two main reasons:

(1) A very large number of image formation processes (and, in fact, measurement procedures in general) are well described by the process of convolution. In fact, if a system is both linear and shift invariant, then image formation is *necessarily* described by convolution.

(2) The *convolution theorem* (which we will discuss shortly) enables us to visualize and understand the convolution process in the spatial frequency domain. This equivalent frequency-domain formulation provides a very powerful mathematical framework to deepen understanding of both image formation and processing. This framework will be developed in Chapter 5 and also forms a major part of Chapter 6.

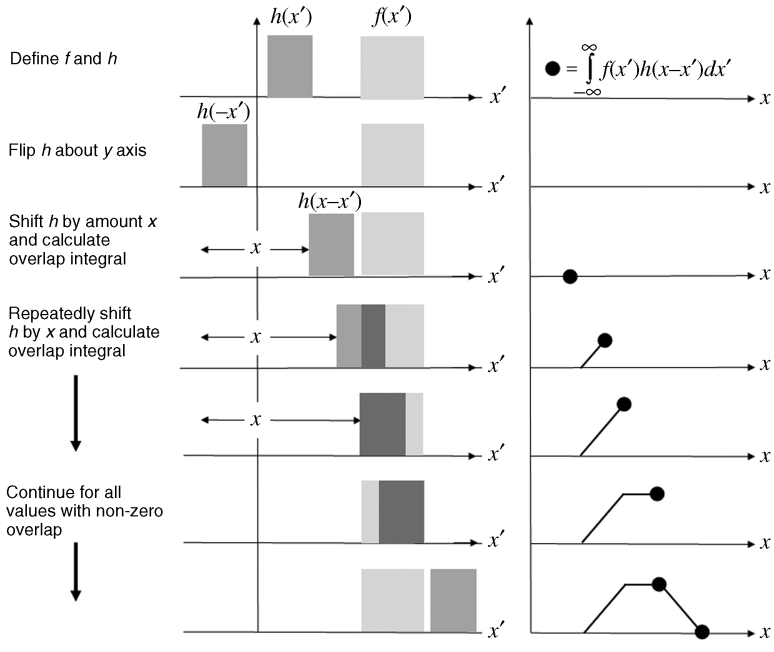
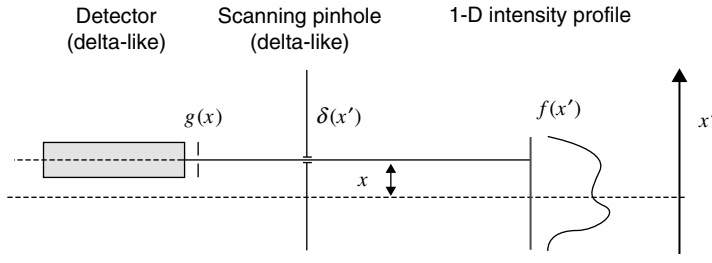


Figure 2.8 The calculation of a 1-D convolution integral

First, let us try to understand the mechanism of the convolution process. A convolution integral is a type of 'overlap' integral, and Figure 2.8 gives a graphical representation of how convolutions occur and are calculated. Basically, a flipped version of one of the functions is systematically displaced and the integral of their product (the overlap area) is calculated at each shift position. All shift coordinates for which the overlap area is nonzero define the range of the convolution.

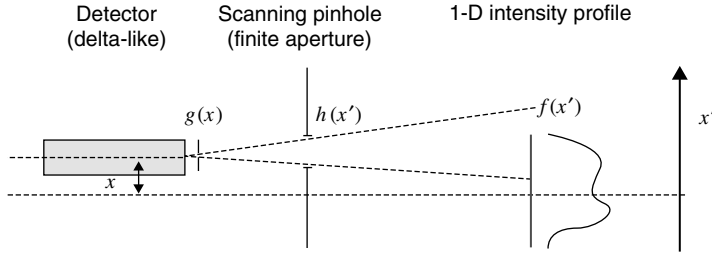
We might well still ask, but why then is the physical process of forming an image so often described by convolution? This is perhaps best understood through a simple example (see Figure 2.9a–c). Gamma rays are too energetic to be focused, so that the very earliest gamma ray detectors used in diagnostic radionuclide imaging consisted of a gamma-ray-sensitive scintillation crystal encapsulated by a thick lead collimator with a *finite-size aperture* to allow normally incident gamma photons to strike and be absorbed in the crystal. The scan was acquired by raster-scanning with uniform speed over the patient; in this way, the derived signal is thus proportional to the gamma activity emanating from that region of the body lying beneath the aperture. Note that a finite-size aperture is absolutely necessary because reducing the aperture size to an infinitesimal size would permit a vanishingly small fraction of emitted photons to actually reach the detector, which is clearly useless for diagnostic purposes.

Let us first remind ourselves that, in general, the linear superposition integral allows for the possibility that *every point* in the input domain $f(x')$ may in principle contribute to the detector response at a chosen point x and does this through the PSF $h(x, x')$; thus, we have quite generally that $g(x) = \int f(x')h(x, x')\,\mathrm{d}x'$. Consider then Figure 2.9a, where for



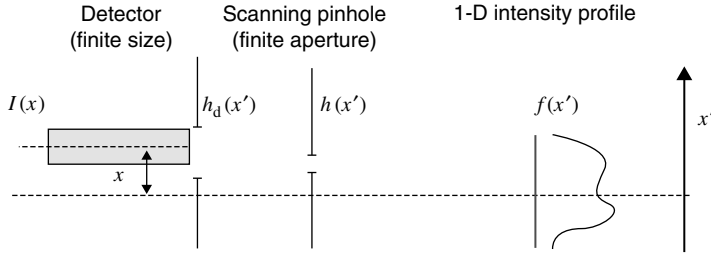
$$g(x) = \int f(x')\, h(x,x')\,\mathrm{d}x' = \int f(x')\, \delta(x - x')\,\mathrm{d}x' = f(x)$$

(a)



$$g(x) = \int f(x')\, h(x,x')\,\mathrm{d}x' = \int f(x')\, h(x - x')\,\mathrm{d}x'$$

(b)



$$I(x) = \int g(x')\, h_{\mathrm{d}}(x,x')\,\mathrm{d}x' \text{ where } g(x) = \int f(x')\, \delta(x - x')\,\mathrm{d}x'$$

$$I(x) = f(x') * h(x,) * h_{\mathrm{d}}(x)$$

(c)

Figure 2.9 (a) A delta like detector views a source intensity profile through a delta-like scanning pinhole. The resulting image is a convolution of the source with the pinhole PSF. In principle, this replicates the source profile so that $g(x) = f(x)$ but this idealized response can never describe a real system. (b) Here, a delta like detector views a source intensity profile through a finite-size scanning pinhole. The resulting image is a convolution of the source with the pinhole PSF. All real systems exhibit such finite aperture effects. (c) A finite-size detector viewing a source intensity profile through a finite scanning pinhole. The resulting image is a convolution of the source intensity with both the pinhole and detector PSFs. In general, if *N* LSI elements are involved in the image formation process, the resulting image is described by convolving the source with each corresponding PSF

simplicity, we depict a 1-D detector and a 1-D intensity source. The idealized pinhole aperture may be approximated by a delta function $\delta(x')$. It is readily apparent that, if the detector were kept stationary but the pinhole aperture scanned along the x' axis, we would only get a nonzero detector response $g(x)$ at the detector when the shift coordinate $x = x'$. This situation is thus described by the linear superposition $g(x) = \int f(x')\delta(x-x')\,dx'$.[3]

If we now consider a *finite* pinhole aperture, as in Figure 2.9b, it is clear that many points in the input domain will now contribute to the detector response $g(x)$ but some will not. In fact, it is clear that only those points in the input domain for which the difference $x-x'$ lies below a certain threshold can contribute. Note that the PSF of this scanning aperture is determined by pure geometry and the detector response is thus independent of the absolute position x' of the aperture – the dependence on x' comes only as a result of how the input $f(x')$ changes.

Finally, consider Figure 2.9c. Here, we depict a finite detector aperture as well as a finite scanning aperture, and this results in a more complicated situation. However, this situation is simply described by an additional further convolution such that the actual recorded intensity at the finite detector is now given by $I(x) = \int g(x')h_{\mathrm{d}}(x-x')\,\mathrm{d}x'$, where $g(x')$ is the intensity at some point on the detector and $h_{\mathrm{d}}(x')$ is the detector response (PSF). The way to understand this situation is depicted in Figure 2.10. Essentially, we break it down into two parts, first calculating the resulting intensity

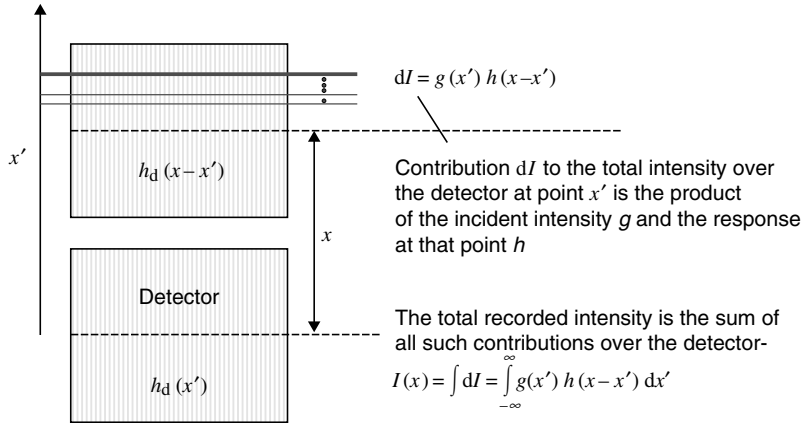


Figure 2.10 The response of a finite detector with response function $h_{\mathrm{d}}(x)$ to an incident intensity field $g(x)$ is given by their convolution. We consider the contribution to the total intensity recorded by the detector at some ordinate x' when the detector is displaced by some amount x. This is $\mathrm{d}I = g(x')h_{\mathrm{d}}(x-x')$. By integrating all such contributions over the detector surface we obtain the convolution integral

[3] Strictly, this supposes that the effective sensitive area of the detector is infinitesimal and thus also well approximated by a delta response – this will occur either because it is infinitesimally small or because the scanning aperture lies very close to it.

at a fixed point on the detector and then considering how all such contributions make up the final recorded intensity. Referring to Figure 2.10, the detector has a response $h_d(x')$ when referred to the origin. We are interested in the total intensity recorded by this detector $I(x)$ when it is displaced some distance x from the origin. The contribution to the recorded intensity dI at some precise point with ordinate x' is given by the product of the incident intensity at that point $g(x')$ times the shifted response $h_d(x-x')$ – thus $dI = g(x')h_d(x-x')$. The total recorded intensity at the detector is thus given by all such contributions dI for which the ordinate x' lies within the dimensions of the detector dimension (i.e. for which $dI = g(x')h_d(x-x') \neq 0$. Thus, we obtain $I(x) = \int dI = \int_{-\infty}^{\infty} g(x')h_d(x-x')\,dx'$.

2.2.8 Multiple convolution: N imaging elements in a linear shift-invariant system

In Figure 2.9c we effectively have an imaging system in which two aperture functions describe the imaging properties: $h(x)$ and $h_d(x)$. Under these circumstances, we have seen that the recorded intensity $I(x)$ is given by successive convolutions of the input $f(x)$ with the PSF of the scanning aperture $h(x)$ and the PSF of the detector $h_d(x)$. Symbolically, we denote this by $I(x) = f(x) * h(x) * h_d(x)$. Provided the assumption of linearity and shift invariance remains valid, this result extends naturally to an arbitrary system of N imaging elements (e.g. a sequence of lenses in an optical imaging system). Thus, in general, *any processing sequence* in which N linear and shift-invariant system elements act upon the input is described by a sequence of N convolutions of the input with the respective PSFs of the elements. This is summarized in Figure 2.11.

2.2.9 Digital convolution

In digital image processing, signals are discrete not continuous. Under these conditions, convolution of two functions is achieved by discretizing the convolution integral. In the simple 1-D case, this becomes

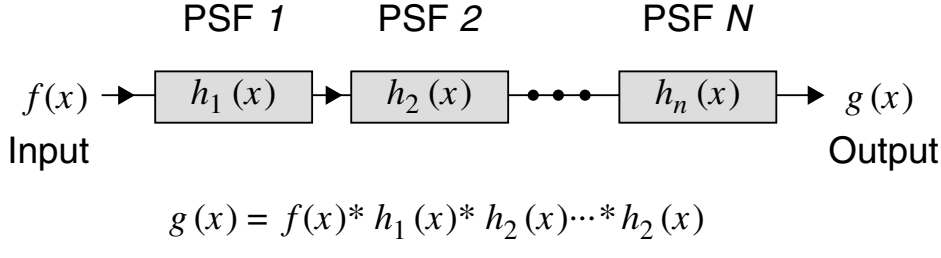


Figure 2.11 The output of an LSI system characterized by N components each having a PSF $h_i(x)$ is the repeated convolution of the input with each PSF

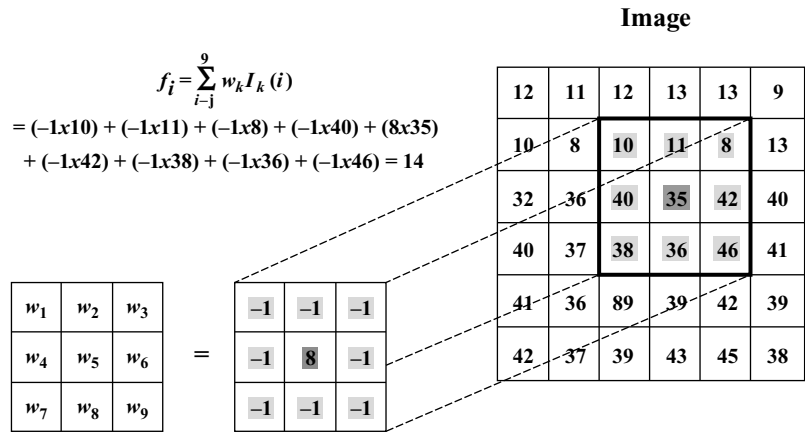


Figure 2.12 Discrete convolution. The centre pixel of the kernel and the target pixel in the image are indicated by the dark grey shading. The kernel is 'placed' on the image so that the centre and target pixels match. The filtered value of the target pixel is then given by a linear combination of the neighbourhood pixels, the specific weights being determined by the kernel values. In this specific case the target pixel of original value 35 has a filtered value of 14

$$g_j = \sum_i f_i h_{j-i} \tag{2.19}$$

where the indices j and i correspond to discrete values of x and x' respectively.

In two dimensions, the discrete convolution integral may be written as

$$g_{kl} = \sum_j \sum_i f_{ij} h_{k-i,l-j} \tag{2.20}$$

where the indices k and l correspond to x and y and i and j correspond to x' and y'. This equation somewhat belies the simplicity of the mechanism involved. The filter kernel h is only nonzero when both the shift coordinates $k - i$ and $l - j$ are small enough to lie within the spatial extent of h (i.e. are nonzero). For this reason, although the image f_{ij} has a large spatial extent, the *filter kernel* $h_{k-i,l-j}$, when it corresponds to a PSF, is typically of much smaller spatial extent. The convolved or filtered function is thus given by a weighted combination of all those pixels that lie beneath the filter kernel (see Figure 2.12). A simple example of 1-D discrete convolution in Matlab® is given in Example 2.1 and some simple examples of 2-D convolution using Matlab are given in Example 2.2. Note that 2-D convolution of two functions that are both of large spatial extent is much more computationally efficient when carried out in the Fourier domain. We will discuss this when we introduce the convolution theorem later in Chapter 5. Figures 2.13 and 2.14 result from the code in Examples 2.1 and 2.2.

Example 2.1

Matlab code

```
f=ones(64,1); f=f./sum(f);
g=conv(f,f); g=g./sum(g);
h=conv(g,g); h=h./sum(h);
j=conv(h,h); j=j./sum(j);

subplot(2,2,1),plot(f,'k-'); axis square;
    axis off;
subplot(2,2,2),plot(g,'k-'); axis square;
    axis off;
subplot(2,2,3),plot(h,'k-'); axis square;
    axis off;
subplot(2,2,4),plot(j,'k-'); axis square;
    axis off;
```

What is happening?

```
%Define rectangle signal f and normalize
%Convolve f with itself to give g and normalize
%Convolve g with itself to give h and normalize
%Convolve h with itself to give j and normalize
```

Comments

Matlab functions: ***conv.***

This example illustrates the repeated convolution of a 1-D uniform signal with itself. The resulting signal quickly approaches the form of the normal distribution, illustrating the central limit theorem of statistics.

Example 2.2

Matlab code

```
A=imread('trui.png');
PSF= fspecial('gaussian',[5 5],2);
h=fspecial('motion',10,45);
B=conv2(PSF,A);
C=imfilter(A,h,'replicate');
D=conv2(A,A);
subplot(2,2,1),imshow(A);
subplot(2,2,2),imshow(B,[]);
subplot(2,2,3),imshow(C,[]);
subplot(2,2,4),imshow(D,[]);
```

What is happening?

```
%Read in image
%Define Gaussian convolution kernel
%Define motion filter
%Convolve image with convolution kernel
%Convolve motion PSF using alternative function
%Self-convolution – motion blurred with original
%Display original image
%Display filtered image
%Display filtered image
%Display convolution image with
  itself (autocorrln)
```

Comments

Matlab functions: ***conv2***, ***imfilter***.

This example illustrates 2-D convolution. The first two examples show convolution of the image with a Gaussian kernel using two related Matlab functions. The final image shows the *autocorrelation* of the image given by the convolution of the function with itself.

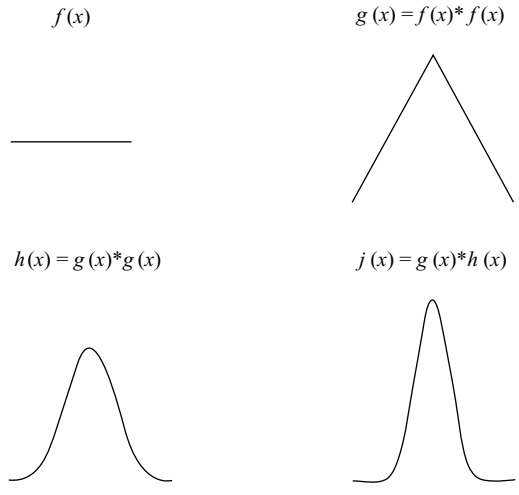


Figure 2.13 Repeated convolution of a 1-D rectangle signal. The output rapidly approaches the form of the normal distribution, illustrating the central limit theorem of statistics

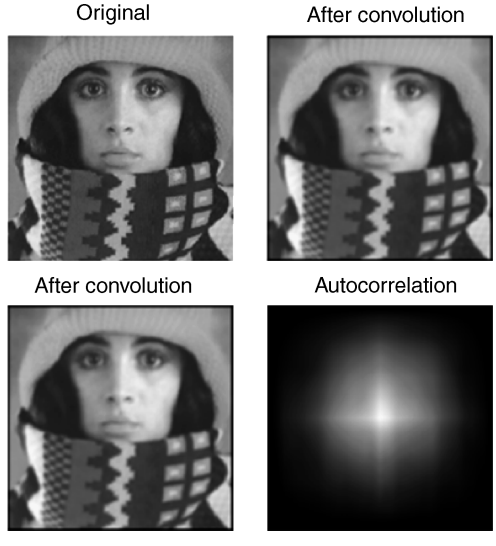


Figure 2.14 Top left: original image; top right: after convolution with Gaussian kernel; bottom left: after convolution with Gaussian kernel; bottom right: convolution of the image with itself

2.3 The engineering of image formation

From our mathematical consideration of the image formation, we now briefly consider the engineering aspects of this process. It is not possible to consider every eventuality in this instance, but we instead limit ourselves to the most common and, hence, *practical* imaging scenarios.

2.3.1 The camera

In general, let us consider our imaging device to be a camera. A camera image of a conventional scene is essentially a projection of the 3-D world (i.e. the scene) to a 2-D representation (i.e. the image). The manner in which this 3-D to 2-D projection occurs is central to our ability to perform effective analysis on digital images.

If we assume an object/scene is illuminated by a light source (possibly multiple sources), then some of the light will be reflected towards the camera and captured as a digital image. Assuming we have a conventional camera, the light reflected into the lens is imaged onto the camera image plane based on the camera projection model. The camera projection model transforms 3-D world coordinates (X, Y, Z) to 2-D image coordinates (x, y) on the image plane. The spatial quantization of the image plane projection into a discretized grid of pixels in turn transforms the 2-D image coordinate on the image plane to a pixel position (c, r). The majority of scene images we deal with will be captured using a perspective camera projection.

Perspective projection: In general the perspective projection (Figure 2.15, right) can be stated as follows:

$$x = f\frac{X}{Z} \qquad y = f\frac{Y}{Z} \tag{2.21}$$

A position (X, Y, Z) in the scene (depth Z) is imaged at position (x, y) on the image plane determined by the focal length of the camera f (lens to image plane distance). The perspective projection has the following properties (illustrated in Figure 2.16):

- *Foreshortening* The size of objects imaged is dependent on their distance from the viewer. Thus, objects farther away from the viewer appear smaller in the image.

- *Convergence* Lines that are parallel in the scene appear to converge in the resulting image. This is known as perspective distortion, with the point(s) at which parallel lines meet termed the vanishing point(s) of the scene.

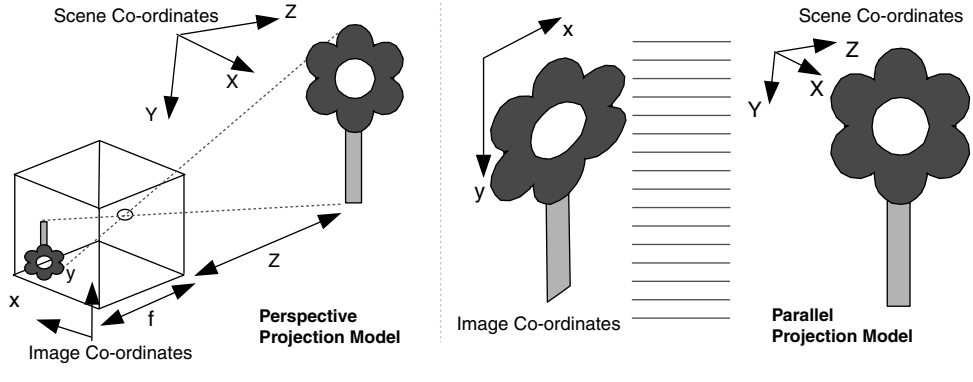


Figure 2.15 Camera projection models: perspective (left) and parallel/affine (right)

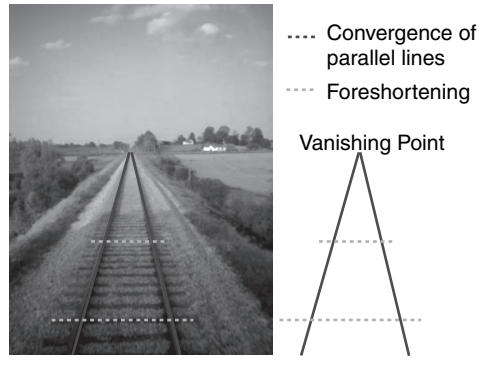


Figure 2.16 Effects of the perspective projection model on scene geometry in the image

To be more precise this is the pin-hole perspective projection model, as we assume all the light rays pass through a single point and, hence, the scene is inverted (horizontally and vertically) on the image plane (Figure 2.15, left). Realistically, we deploy a lens, which further complicates matters by introducing lens distortion (affecting f), but this optics-related topic is beyond the scope of this text. With a lens, the scene is still imaged upside down (and back to front), but this is generally dealt with by the camera and should be ignored for all applied purposes. It should be noted that the perspective projection is not easily invertible from an image alone – i.e. given 2-D image position (x, y) and camera focal length f it is not possible to recover (X, Y, Z). However, given knowledge of (X, Y) in the scene (measured relative to the camera position), the recovery of position depth Z may become possible (subject to noise and distortion). Accurate (X, Y) positions, relative to the exact camera position, are, however, difficult to determine and rarely recorded at time of image capture.

Orthographic projection: An alternative camera projection model to the perspective projection is the orthographic (or parallel) projection. This is used by some specialist imaging instruments; for instance, a flat-bed scanner produces an orthographic projection of a scanned document, a medical scanner produces an orthographic projection of the human body. The orthographic projection is simply denoted as

$$x = X \qquad y = Y$$

The orthographic projection is an affine transformation such that relative geometric relationships are maintained, as the scene to image plane projection is parallel. As such, the image features are not reversed (Figure 2.15, right), but notably *all* information relating to scene depth is lost. In an orthographic projection, size is independent of the distance from the viewer and parallel lines are preserved. The projection may have a scaling factor m such that $x = mX$ and $y = mY$, but this relates to regular image zooming or reduction and maintains these properties. Images captured over a close range (i.e. short focal length, macro-scale photography) or of a planar target can often be assumed to have an orthographic camera projection because of the negligible effects of the perspective projection over this range.

In addition to the projection model, the camera has a number of other characteristics that determine the final image formed from the scene. Notably any camera is of fixed spatial

resolution (number of pixels making up the image) and has a fixed depth per pixel (number of bits used to represent each pixel) (see Section 2.1). Both of these are discussed in the next section regarding the image digitization process. In addition, every camera has unique lens distortion characteristics related to differences in manufacture (referred to as the intrinsic camera parameters) that together with similar subtle differences in the image sensor itself make it impossible to capture two identical digital images from a given camera source – *thus no two images are the same.*

2.3.2 The digitization process

From our discussion of how our image is formed by the camera as a 3-D to 2-D projective transform (Section 2.3.1), we now move on to consider the discretization of the image into a finite-resolution image of individual pixels. The key concept in digitization is that of quantization: the mapping of the continuous signal from the scene to a discrete number of spatially organized points (pixels) each with a finite representational capacity (pixel depth).

2.3.2.1 Quantization

Quantization in digital imaging happens in two ways: spatial quantization and colour quantization.

Spatial quantization corresponds to sampling the brightness of the image at a number of points. Usually a $C \times R$ rectangular grid is used but variations from rectangular do exist in specialist sensors. The quantization gives rise to a matrix of numbers which form an approximation to the analogue signal entering the camera. Each element of this matrix is referred to as a pixel – an individual picture element. The fundamental question is: How well does this quantized matrix approximate the original analogue signal ? We note that

(1) If n^2 samples are taken at regular intervals (uniform sampling, $n \times n$ image) within a bounding square, then the approximation improves as n increases. As spatial resolution increases so does image quality in terms of the reduction in approximation error between the original and the digitized image.

(2) In general, as long as sufficient samples are taken, a spatially quantized image is as good as the original image (for the purposes required, i.e. at a given level of detail).

The precise answer to our question is provided by the *sampling theorem* (also referred to as the *Nyquist sampling theorem* or *Shannon's sampling theorem*). The sampling theorem, in relation to imaging, states: an analogue image can be reconstructed exactly from its digital form as long as the sampling frequency (i.e. number of samples per linear measure) is at least twice the highest frequency (i.e. variation per measure) present in the image. Thus we require:

$$\text{sampling interval} \leq \frac{1}{\text{Nyquist frequency}} \tag{2.22}$$

where

$$\text{Nyquist frequency} = 2 \times (\text{Maximum frequency in image})$$

The sampling theorem is concerned with the number of samples needed to recreate the original image, not with the adequacy of the digitization for any particular type of processing or presentation. As far as the implementation in digitization hardware is concerned, the sampling interval is effectively set by:

- Δt (time between samples) in a scanning-based image capture process where a sensor is being swept over a region or scene (e.g. a desktop flat-bed scanner);

- the spatial distribution of the charge-coupled device (CCD) or complementary metal–oxide–semiconductor (CMOS) elements on the capture device itself (see Section 2.3.2.2).

Spatial quantization of the image causes aliasing to occur at edges and features within the image (Figure 2.17, left) and differing spatial quantization can affect the level of detail apparent within the image (Figure 2.17, right).

In addition to spatial sampling, we also have the colour or intensity sampling to consider – effectively *intensity quantization*. For each sample point (pixel) captured at a given spatial resolution we obtain a voltage reading on the image sensor (i.e. the CCD/CMOS or analogue-to-digital (A/D) converter) that relates to the amount and wavelength of light being projected through the camera lens to that point on the image plane. Depending on the sensitivity of the sensor to particular levels and wavelengths of light, these analogue voltage signals can be divided (i.e. discretized) into a number of bins each representing a specific level of intensity – pixel values. For an 8-bit grey-scale sensor this continuous voltage is divided into 2^8 bins, giving 256 possible pixel values and thus representing a grey scale ranging from 0 (black) to 255 (white).

Different levels of intensity quantization give different levels of image colour quality, as shown in Figure 2.18. Notably, a significant reduction in the level of quantization can be performed ($256 \rightarrow 16$) before any real effects become noticeable. For colour sensors each of the red, green and blue channels are similarly each quantized into an N-bit representation (typically 8-bits per channel, giving a 24-bit colour image). In total, 24-bit colour gives 16.7 million possible colour combinations. Although these representations may seem limited, given current computing abilities, it is worth noting that the human visual system can at most

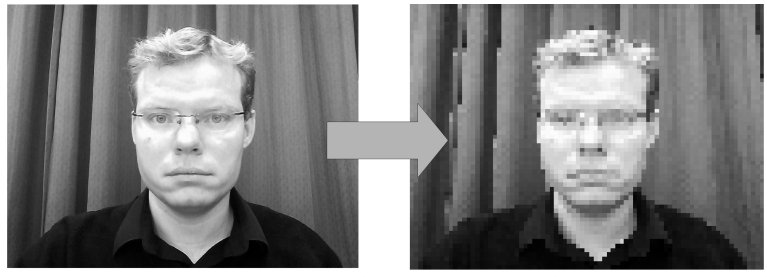


Figure 2.17 The effects of spatial quantization in digital images

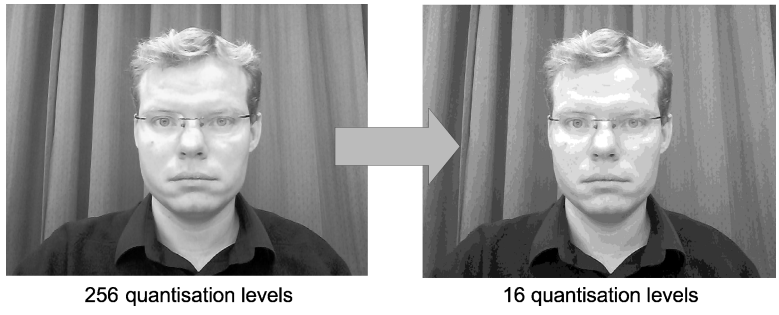


Figure 2.18 The effects of intensity quantization in digital images

determine between ~40 different levels of grey and it is estimated to see around 7–10 million distinct colours. This is illustrated in Figure 2.18, where a perceived difference in colour quality is only visible after a significant 256 → 16 level change. This limitation of human vision has implications for what 'we can get away with' in image compression algorithms (Section 1.3.2). Note that.. although the human visual system cannot determine the difference between these levels, the same is not always true for image processing. A shallow colour depth (i.e. low-intensity quantization) can lead to different objects/features appearing at the same quantization level despite being visually distinct in terms of colour. Again, aliasing in intensity quantization can occur (e.g. the 16 grey-levels example in Figure 2.18).

2.3.2.2 Digitization hardware

Both forms of quantization, spatial and intensity, are performed near simultaneously in the digitization hardware.

Traditionally, image capture was performed based on an analogue video-type camera connected to a computer system via an analogue to digital (A/D) converter. The converter required enough memory to store an image and a high-speed interface to the computer (commonly direct PCI or SCSI interface). When signalled by the computer, the frame grabber digitized a frame and stored the result in its internal memory. The data stored in the image memory (i.e. the digital image) can then be read into the computer over the communications interface. The schematic of such a system would typically be as per Figure 2.19, where we see an analogue link from the camera to the computer system itself and digitization is essentially internal to the hardware unit where the processing is performed.

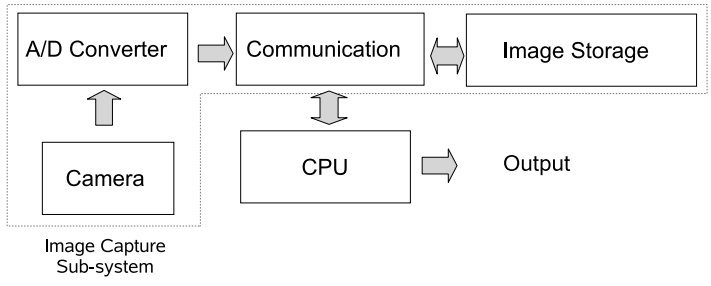


Figure 2.19 Schematic of capture-card-based system

Figure 2.20 Example of CCD sensor in a digital video camera

Such systems are now rarer and predominantly for high-speed industrial or specialist medical/engineering applications. By contrast, most modern frame-grabbing systems are directly integrated into the camera, with the A/D conversion being done at the sensor level in the electronics of the CCD or CMOS sensor inside the camera itself (i.e. digital capture, Figure 2.20). Here, analogue voltage readings occurring upon the sensor itself, in response to light being projected through the camera lens onto the sensor surface, are converted into a digitally sampled form. The digitization is carried out *in situ*, inside the sensor itself prior, to a digital transport link to the computer. Modern computer peripheral interfaces such as USB/FireWire (IEEE 1394)/GigE offer suitable high-speed digital interfaces to make digitization at sensor level and real-time transmission to the computer system a possibility. The image capture subsystem in Figure 2.19 is essentially reduced onto the capture device itself, either retaining only the most recent 'live' image to service requests from the computer (e.g. webcams) or, in the case of storage-based capture devices, retaining up to N digitized images in internal storage (e.g. digital camera/video). This is conceptually similar to what is believed to happen in the human eye, where basic 'processing' of the image is thought to occur in the retina before transmission onwards to the brain via the optic nerve. Increasingly, the advent of embedded processing is facilitating the onboard processing of images in so-called smart cameras. This trend in image capture and processing looks likely to continue in the near future.

2.3.2.3 Resolution versus performance

A final note on digitization relates to the spatial resolution of the image. A common misconception is that a greater resolution is somehow always a good thing. Image processing is by its very nature a very computationally demanding task. Just as in traditional computational analysis, where we discuss computational runtime in terms of $O(n)$ for a given algorithm operating on n items, in image processing we consider an $(n \times n)$ image (which generalizes to the differing $R \times C$ case).

With serial processing, any operation upon every image pixel (most common) is inherently quadratic in runtime, $O(n^2)$. Thus, the total number of operations increases

rapidly as image size increases. Here, we see the limitation of increasing image resolution to achieve better image-processing results- *speed*. As a result, efficiency is a major concern when designing image-processing algorithms and is especially important for systems requiring the processing of video data with high temporal resolution in real time. In order to counter this high computational demand and to address the increasing need for real-time image-processing systems, specialist parallel computing facilities have traditionally be frequently used in image-processing tasks. Specialist hardware for conventional PCs, such as image-processing boards, were also commonplace and essentially performed user-defined image-processing tasks using fast hardware implementations independently of the PC itself.

With the ongoing increase in consumer-level image-processing power (Moore's law), the adaptation of processors to specialist multimedia tasks such as image processing (e.g. Intel MMX technologies) and now the advent of desktop-level multi-processor systems (e.g. Intel Core-Duo), the use of such specialist hardware has reduced in the image-processing industry. Although still used in industrial applications, the advent of embedded PC technology and robust embedded image-processing-capable operating systems look set to change this over the coming years. Reconfigurable, programmable hardware such as field-programmable gate arrays (FPGA) and the use of graphics processing unit (GPU) processing (image processing on the PC graphics card processor) also offer some of the benefits of a hardware solution with the additional flexibility of software based development/testing.

Modern PCs can cope well with real-time image-processing tasks and have memory capacities that realistically exceed any single image. The same is not always true of video stream processing, and performance evaluation and optimization is still an area of research.

2.3.3 Noise

The main barrier to effective image processing and signal processing in general is noise. By noise we mean a variation of the signal from its true value by a small (random) amount due to external or internal factors in the image-processing pipeline. These factors cannot be (easily) controlled and thus introduce random elements into the processing pipeline. Noise is the key problem in 99 % of cases where image processing techniques either fail or further image processing is required to achieve the required result. As a result, a large part of the image processing domain, and any image processing system pipeline, is dedicated to noise reduction and removal. A robust image processing system must be able to cope with noise. At each stage of image processing, capture and sampling noise is introduced (Figure 2.21).

Noise in digital images can originate from a variety of sources :

- *Capture noise* can be the result of variations in lighting, sensor temperature, electrical sensor noise, sensor nonuniformity, dust in the environment, vibration, lens distortion, focus limitations, sensor saturation (too much light), underexposure (too little light).

- *Sampling noise* As discussed previously (Section 2.3.2), limitations in sampling and intensity quantization are a source of noise in the form of representational aliasing. The sampled digital image is not a true representation of the analogue image, but an alias of the original.

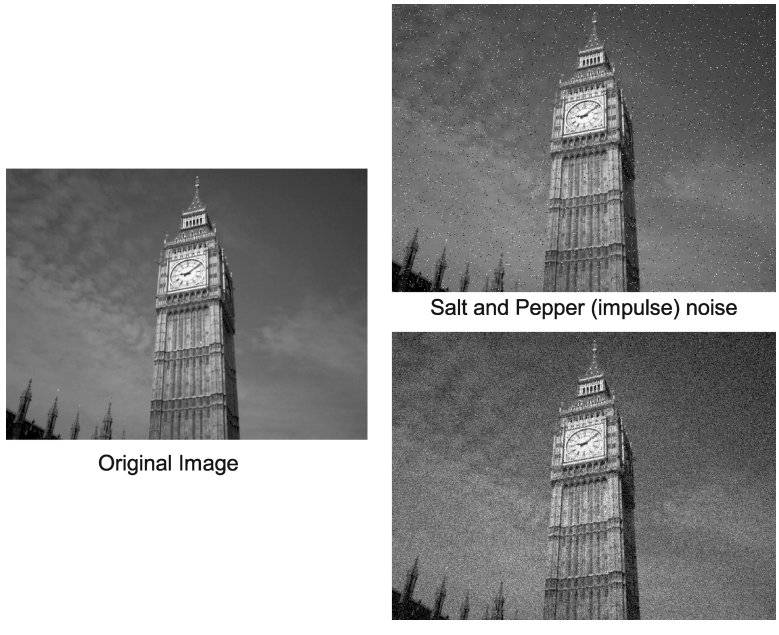


Figure 2.21 Examples of noise effects in digital imaging

- *Processing noise* Limitations in numerical precision (floating-point numbers), potential integer overflow and mathematical approximations (e.g. $\pi = 3.142\ldots$) are all potential sources of noise in the processing itself.

- *Image-encoding noise* Many modern image compression techniques (e.g. JPEG used intrinsically by modern digital cameras) are lossy compression techniques. By lossy we mean that they compress the image by removing visual information that represents detail not general perceivable to the human viewer. The problem is that this loss of information due to compression undermines image-processing techniques that rely on this information. This loss of detail is often referred to by the appearance of *compression artefacts* in the image. In general, loss = compression artefacts = noise. Modern image formats, such as PNG, offer lossless compression to counter this issue (see Section 1.3.2).

- *Scene occlusion* In the task of object recognition, objects are frequently obscured by other objects. This is known as occlusion. Occlusion poses a big problem for computer vision systems because *you don't know what you cannot see*. You may be able to infer recognition from a partial view of an object, but this is never as robust as full-view recognition. This limits available image information.

Ignoring systematic noise attributable to a specific cause (e.g. nonuniform lighting), noise in images can be characterized in a number of ways. The main two noise characterizations used in imaging are:

- *Salt and pepper noise* This is caused by the random introduction of pure white or black (high/low) pixels into the image (Figure 2.21). This is less common in modern image sensors, although can most commonly be seen in the form of camera sensor faults (hot pixels that are always at maximum intensity or dead pixels which are always black). This type of noise is also known as *impulse noise*.

- *Gaussian noise* In this case, the random variation of the image signal around its expected value follows the Gaussian or normal distribution (Figure 2.21). This is the most commonly used noise model in image processing and effectively describes most random noise encountered in the image-processing pipeline. This type of noise is also known as *additive noise*.

Ultimately, no digital image is a perfect representation of the original scene: it is limited in resolution by sampling and contains noise. One of the major goals of image processing is to limit the effects of these aspects in image visualization and analysis.

Exercises

The following exercises are designed to reinforce and develop the concepts and Matlab examples introduced in this chapter. Additional information on all of the Matlab functions represented in this chapter and throughout these exercises is available in Matlab from the function help browser (use *doc* <*function name*> at the Matlab command prompt, where <function name> is the function required)

Matlab functions: ***rand()***, ***fspecial()***, ***filter2()***, ***imresize()***, ***tic()***, ***toc()***, ***imapprox()***, ***rgb2ind()***, ***gray2ind()***, ***imnoise()***, ***imtool()***, ***hdrread()***, ***tonemap()***, ***getrangefromclass()***.

Exercise 2.1 Using Example 2.3 we can investigate the construction of a coherent image from a limited number of point samples via the concept of the PSF introduced in Section 2.1:

Example 2.3

Matlab code	**What is happening?**
A=imread('cameraman.tif');	%Read in an image
[rows dims]=size(A);	%Get image dimensions
Abuild=zeros(size(A));	%Construct zero image of equal size

```
%Randomly sample 1% of points only and convolve with Gaussian PSF

sub=rand(rows.*dims,1)<0.01;
Abuild(sub)=A(sub); h=fspecial('gaussian',[10 10],2);
B10=filter2(h,Abuild);
subplot(1,2,1), imagesc(Abuild); axis image; axis off;colormap(gray); title('Object points')
subplot(1,2,2), imagesc(B10); axis image; axis off;colormap(gray); title('Response of LSI system')
```

Here, we randomly select 1% (0.01) of the image pixels and convolve them using a Gaussian-derived PSF (see Section 4.4.4) applied as a 2-D image filter via the ***filter2()*** function. Experiment with this example by increasing the percentage of pixels randomly selected for PSF convolution to form the output image. At what point is a coherent image formed? What noise characteristics does this image show? Additionally, investigate the use of the ***fspecial()*** function and experiment with changing the parameters of the Gaussian filter used to approximate the PSF. What effect does this have on the convergence of the image towards a coherent image (i.e. shape outlines visible) as we increase the number of image pixels selected for convolution?

Exercise 2.2 The engineering of image formation poses limits on the spatial resolution (Section 1.2) of an image through the digitization process. At the limits of spatial resolution certain image details are lost. Using the Matlab example images 'football.jpg' and 'text.png', experiment with the use of the ***imresize()*** function both in its simplest form (using a single scale parameter) and using a specified image width and height (rows and columns). As you reduce image size, at what point do certain image details deteriorate beyond recognition/comprehension within the image? You may also wish to investigate the use of ***imresize()*** for enlarging images. Experiment with the effects of enlarging an image using all three of the available interpolation functions for the Matlab ***imresize()*** function. What differences do you notice? (Also try timing different resizing interpolation options using the Matlab ***tic()***/***toc()*** functions.)

Exercise 2.3 In addition to spatial resolution, the engineering of image formation also poses limits on the number of quantization (colour or grey-scale) levels in a given image. Using the Matlab function ***imapprox()*** we can approximate a given image represented in a given colour space (e.g. RGB or grey scale, Section 1.4.1.1) with fewer colours. This function operates on an indexed image (essentially an image with a reference look-up table for each of its values known as a colour map) – see 'indexed images' in Matlab Help. A conventional image (e.g. Matlab example 'peppers.png') can be converted to an indexed image and associated look-up table colour map using the Matlab ***rgb2ind()*** function (or ***gray2ind()*** for grey-scale images). Investigate the use of ***imapprox()*** and ***rgb2ind()***/***gray2ind()*** to reduce the number of colours in a colour image example and a greyscale image example. Experiment with different levels of colour reduction. How do your findings tally with the discussion of human colour perception in Section 2.3.2.1)?

Exercise 2.4 The Matlab ***imnoise()*** function can be used to add noise to images. Refer to Example 4.3 for details of how to use this function and experiment with its use as suggested in the associated Exercise 4.1.

Exercise 2.5 The Matlab function ***imtool()*** allows us to display an image and perform a number of different interactive operations upon it. Use this function to load the example image shown in Figure 2.16 (available as 'railway.png') and measure the length in pixels of the railway sleepers in the foreground (front) and background (rear) of the image. Based on the discussion of Section 2.3.1, what can we determine about the relative distance of the two sleepers you have measured to the camera? If we assume the standard length of a railway

sleeper is 2.5 m, what additional information would be need to determine the absolute distance of these items from the camera? How could this be applied to other areas, such as estimating the distance of people or cars from the camera?

Exercise 2.6 High dynamic range (HDR) imaging is a new methodology within image capture (formation) whereby the image is digitized with a much greater range of quantization levels (Section 2.3.2.1) between the minimum and maximum colour levels. Matlab supports HDR imaging through the use of the ***hdrread()*** function. Use this function to load the Matlab example HDR image 'office.hdr' and convert it to a conventional RGB colour representation using the Matlab ***tonemap()*** function for viewing (with ***imshow()***). Use the Matlab ***getrangefromclass()*** function and ***imtool()*** to explore the value ranges of these two versions of the HDR image.

For further examples and exercises see http://www.fundipbook.com