2 Images, sampling and frequency domain processing

2.1 Overview

In this chapter, we shall look at the basic theory which underlies image formation and processing. We shall start by investigating what makes up a picture and then look at the consequences of having a different number of points in the image. We shall also look at images in a different representation, known as the frequency domain. In this, as the name implies, we consider an image as a collection of frequency components. We can actually operate on images in the frequency domain and we shall also consider different transformation processes. These allow us different insights into images and image processing which will be used in later chapters not only as a means to develop techniques, but also to give faster (computer) processing.

Table 2.1 Overview of Chapter 2

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

2.2 Image formation

A computer image is a matrix (a two-dimensional array) of *pixels*. The value of each pixel is proportional to the *brightness* of the corresponding point in the scene; its value is often derived from the output of an A/D converter. The matrix of pixels, the image, is usually square and we shall describe an image as $N \times N$ m-bit pixels where N is the number of points along the axes and m controls the number of brightness values. Using m bits gives a range of 2^m values, ranging from 0 to $2^m - 1$. If m is 8 this gives brightness levels ranging between 0 and 255, which are usually displayed as black and white, respectively, with shades of grey in between, as they are for the *greyscale image* of a walking man in Figure **2.1**(a). Smaller values of m give fewer available levels reducing the available contrast in an image.

The ideal value of m is actually related to the signal to noise ratio (bandwidth) of the camera. This is stated as approximately 45 dB and since there are 6 dB per bit, then 8 bits will cover the available range. Choosing 8-bit pixels has further advantages in that it is very convenient to store pixel values as *bytes*, and 8-bit A/D converters are cheaper than those with a higher resolution. For these reasons images are nearly always stored as 8-bit bytes, though some applications use a different range. The relative influence of the 8 bits is shown in the image of the walking subject in Figure **2.1**. Here, the least significant bit, bit 0 (Figure **2.1**(b)), carries the least information (it changes most rapidly). As the order of the bits increases, they change less rapidly and carry more information. The most information is carried by the most significant bit, bit 7 (Figure **2.1**(i)). Clearly, the fact that there is a walker in the original image can be recognised much better from the high order bits, much more reliably than it can from the other bits (notice too the odd effects in the bits which would appear to come from lighting at the top left corner).

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

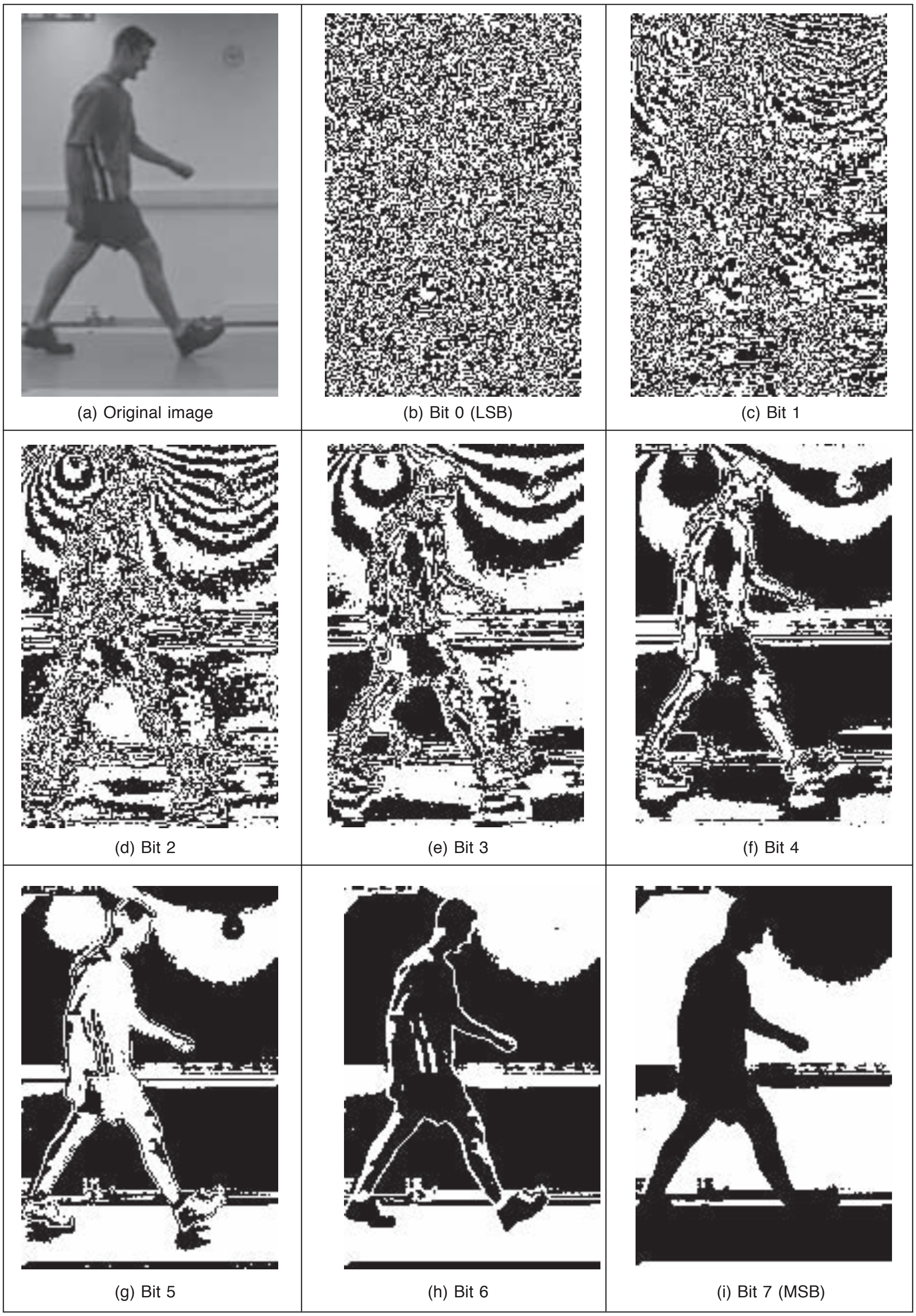
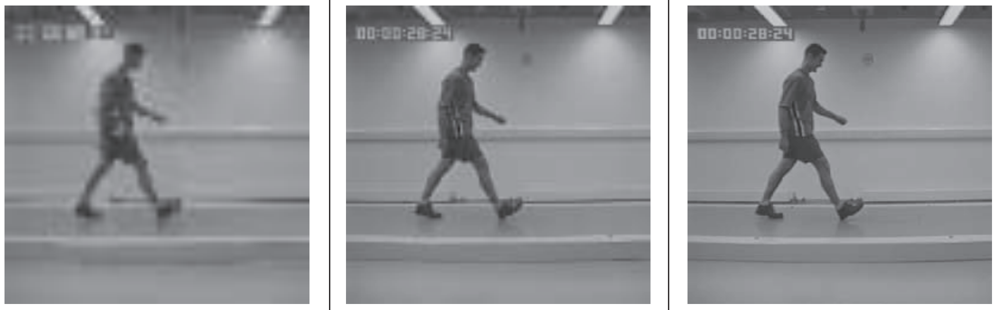


Figure 2.1 Decomposing an image into its bits

consider processing its luminance only, often computed in a standard way. In any case, the amount of memory used is always related to the image size.

Choosing an appropriate value for the image size, *N*, is far more complicated. We want *N* to be sufficiently large to resolve the required level of spatial detail in the image. If *N* is too *small*, the image will be coarsely quantised: lines will appear to be very ‘blocky’ and some of the detail will be *lost*. Larger values of *N* give more *detail*, but need more storage space and the images will take longer to process, since there are more pixels. For example, with reference to the image of the walking subject in Figure **2.1**(a), Figure **2.2** shows the effect of taking the image at different resolutions. Figure **2.2**(a) is a 64 × 64 image, that shows only the broad structure. It is impossible to see any detail in the subject’s face. Figure **2.2**(b) is a 128 × 128 image, which is starting to show more of the detail, but it would be hard to determine the subject’s identity. The original image, repeated in Figure **2.2**(c), is a 256 × 256 image which shows a much greater level of detail, and the subject can be recognised from the image. (These images actually come from a research programme aimed to use computer vision techniques to recognise people by their gait; face recognition would be of little potential for the low resolution image which is often the sort of image that security cameras provide.) If the image was a pure photographic image, some of the much finer detail like the hair would show up in much greater detail. This is because the grains in film are very much smaller than the pixels in a computer image. Note that the images in Figure **2.2** have been scaled to be the same size. As such, the pixels in Figure **2.2**(a) are much larger than in Figure **2.2**(c) which emphasises its blocky structure. The most common choices are for 256 × 256 or 512 × 512 images. These require 64 and 256 Kbytes of storage, respectively. If we take a sequence of, say, 20 images for motion analysis, we will need more than 1 Mbyte to store the 20 256 × 256 images, and more than 5 Mbytes if the images were 512 × 512. Even though memory continues to become cheaper, this can still impose high cost. But it is not just cost which motivates an investigation of the appropriate image size, the appropriate value for *N*. The main question is: are there theoretical guidelines for choosing it? The short answer is ‘yes’; the long answer is to look at digital signal processing theory.



(a) 64 × 64 (b) 128 × 128 (c) 256 × 256

Figure 2.2 Effects of differing image resolution

The choice of sampling frequency is dictated by the *sampling criterion*. Presenting the sampling criterion requires understanding how we interpret signals in the *frequency domain*.

The way in is to look at the Fourier transform. This is a highly theoretical topic, but do not let that put you off. The Fourier transform has found many uses in image processing and understanding; it might appear to be a complex topic (that's actually a horrible pun!) but it is a very rewarding one to study. The particular concern is the appropriate sampling frequency of (essentially, the value for N), or the rate at which pixel values are taken from, a camera's video signal.

2.3 The Fourier transform

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

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

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

$$Fp(\omega) = \int_{-\infty}^{\infty} p(t)e^{-j\omega t}dt \tag{2.1}$$

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

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

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

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

$$Fp(\omega) = \int_{-T/2}^{T/2} Ae^{-j\omega t}\,dt \tag{2.3}$$

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

$$Fp(\omega) = -\frac{Ae^{-j\omega T/2} - Ae^{j\omega T/2}}{j\omega} \tag{2.4}$$

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

$$Fp(\omega) = \left| \begin{array}{lll} \frac{2A}{\omega}\sin\left(\frac{\omega T}{2}\right) & \text{if} & \omega \neq 0 \\ AT & \text{if} & \omega = 0 \end{array} \right. \tag{2.5}$$

This is a version of the *sinc* function, sinc(*x*) = sin(*x*)/*x*. The original pulse and its transform are illustrated in Figure **2.3**. Equation 2.5 (as plotted in Figure **2.3**(a)) suggests that a pulse is made up of a lot of low frequencies (the main body of the pulse) and a few higher frequencies (which give us the edges of the pulse). (The range of frequencies is symmetrical around zero frequency; negative frequency is a necessary mathematical abstraction.) The plot of the Fourier transform is actually called the *spectrum* of the signal, which can be considered akin with the spectrum of light.

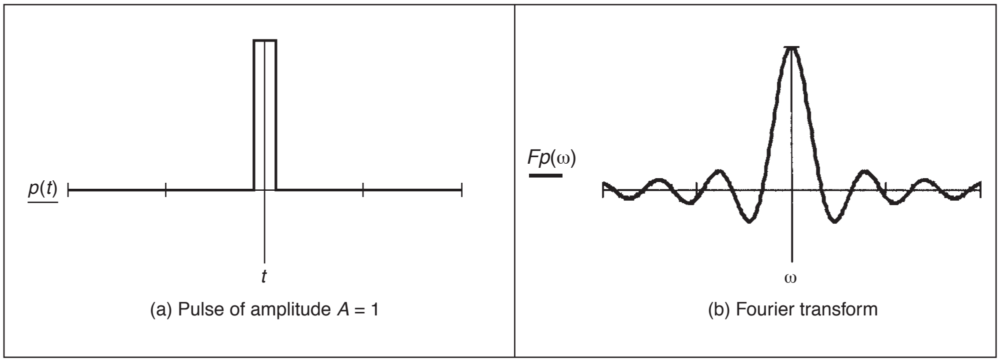
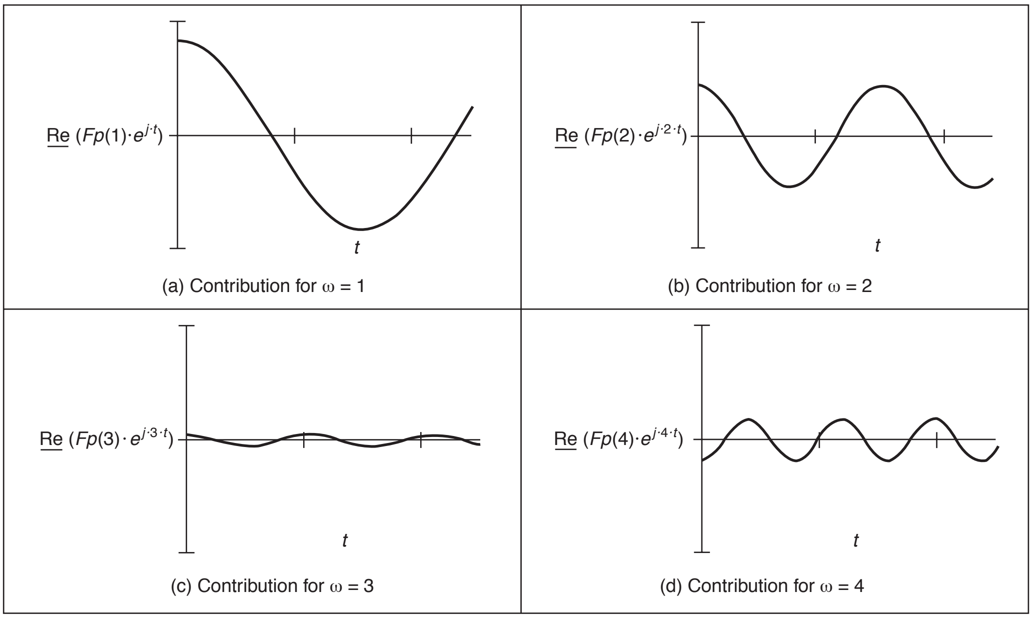


Figure 2.3 A pulse and its Fourier transform

So what actually is this Fourier transform? It tells us what frequencies make up a time domain signal. The magnitude of the transform at a particular frequency is the amount of that frequency in the original signal. If we collect together sinusoidal signals in amounts specified by the Fourier transform, then we should obtain the originally transformed signal. This process is illustrated in Figure **2.4** for the signal and transform illustrated in Figure **2.3**. Note that since the Fourier transform is actually a *complex* number it has real and imaginary parts, and we only plot the *real* part here. A low frequency, that for ω = 1, in Figure **2.4**(a) contributes a large component of the original signal; a higher frequency, that for ω = 2, contributes less as in Figure **2.4**(b). This is because the transform coefficient is less for ω = 2 than it is for ω = 1. There is a very small contribution for ω = 3, Figure **2.4**(c), though there is more for ω = 4, Figure **2.4**(d). This is because there are frequencies for which there is no contribution, where the transform is zero. When these signals are integrated, we achieve a signal that looks similar to our original pulse, Figure **2.4**(e). Here we have only considered frequencies from ω = −6 to ω = 6. If the frequency range in integration



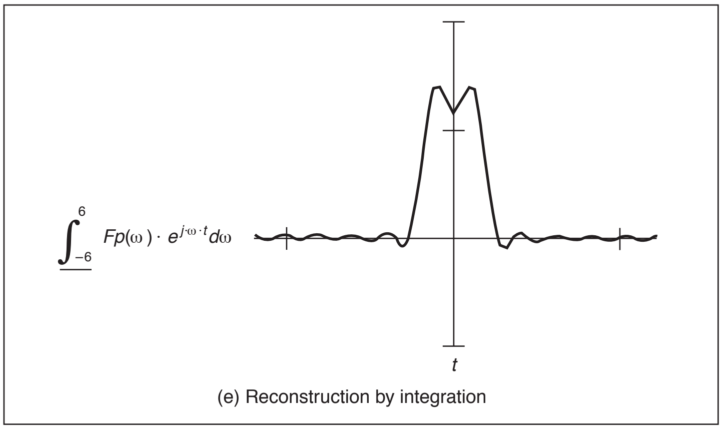


Figure 2.4 Reconstructing a signal from its transform

was larger, more high frequencies would be included, leading to a more faithful reconstruction of the original pulse.

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

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

where Re(ω) and Im(ω) are the real and imaginary parts of the transform, respectively. The *magnitude* of the transform is then:

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

and the *phase* is:

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

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

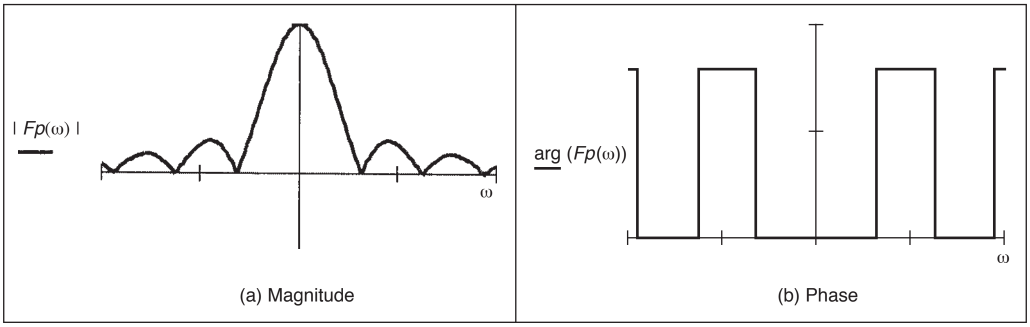


Figure 2.5 Magnitude and phase of Fourier transform of pulse

In order to return to the time domain signal, from the frequency domain signal, we require the *inverse Fourier transform*. Naturally, this is the process by which we reconstructed the pulse from its transform components. The inverse FT calculates *p*(*t*) from *Fp*(ω) according to:

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

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

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

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

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

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

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

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

Further, *correlation* is defined to be

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

where $\otimes$ denotes correlation ($\odot$ is another symbol which is used sometimes, but there is not much consensus on this symbol). Correlation gives a measure of the *match* between the two signals $p_2(\omega)$ and $p_1(\omega)$. When $p_2(\omega) = p_1(\omega)$ we are correlating a signal with itself and the process is known as *autocorrelation*. We shall be using correlation later, to *find* things in images.