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 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.

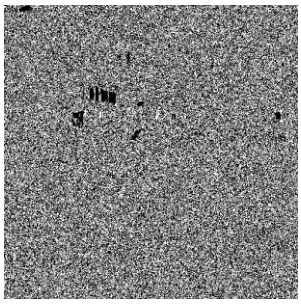
Main topic	Sub topics	Main points
Images	Effects of differing **numbers** of points and of number **range** for those points.	*Grayscale*, *colour*, *resolution*, dynamic range, storage.
Fourier transform theory	What is meant by the **frequency domain**, how it applies to **discrete** (sampled) images, 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. *Inverse Fourier transform*. Importance of *magnitude* and *phase*.
Consequences of transform approach	Basic **properties** of Fourier transforms, **other transforms**, frequency domain **operations**.	*Translation* (shift), *rotation* and *scaling*. *Principle of Superposition* and linearity. *Walsh*, *Hartley*, *Discrete Cosine* and *Wavelet transforms*. *Filtering* and other operations.

Table 2.1 Overview of Chapter 2

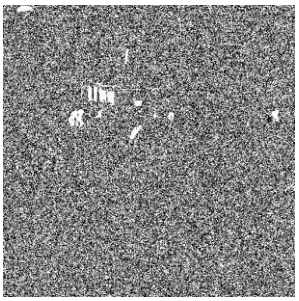
2.2 Image Formation

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

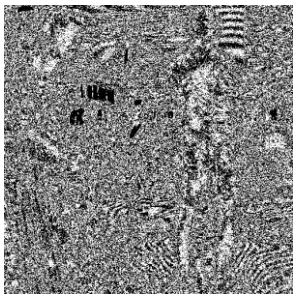
(a) original image



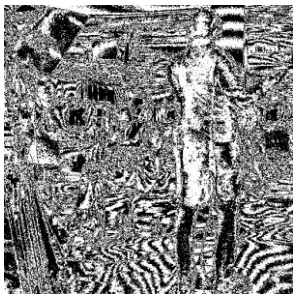
(b) bit 0 (LSB)



(c) bit 1

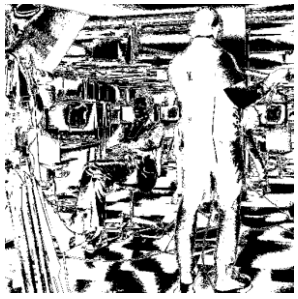


(d) bit 2



(e) bit 3

(f) bit 4



(g) bit 5

(h) bit 6

(i) bit 7 (MSB)

Figure 2.1 Decomposing an Image Into its Bits

The ideal value of *m* is actually related to the signal to noise ratio (dynamic range) of the camera. This is stated as approximately 45 dB for an analogue camera and since there are 6 dB per bit, then 8 bits will cover the available range. Choosing 8-bit pixels has further advantages in that it is very convenient to store pixel 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. These are the 8-bit numbers encountered in Section 1.5.2, stored as unsigned 8-bit bytes (`uint8`). The relative influence of the eight bits is shown in the image of the subjects in Figure 2.1. Here, the least significant bit, bit 0 (Figure 2.1(b)), carries the least information (it changes most rapidly) and that image is largely noise. 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 are people in the original image can be recognised much better from the high order bits, much more reliably than it can from the other bits (also notice the odd effects which would appear to come from lighting in the middle order bits). The variation in lighting is hardy perceived by human vision.

Colour images (also mentioned in Section 1.5.2) 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 color 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 eight bits for each of the three RGB components. These images are known as **24-bit** true colour and they can contain 16777216 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 gray level images only since they contain enough information to perform feature extraction and image analysis; greater depth on colour analysis/parameterisation is to be found in Chapter 13, Appendix 4 on Colour Models. Should the image be originally in colour, we will consider processing its luminance only, often computed in a standard way. In any case, the amount of memory used is always related to the image size.

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

Figure 2.2 Effects of Differing Image Resolution

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 sitting subject's face, or anywhere else. Figure 2.2(b) is a 128×128 image, which is starting to show more of the detail, but it would be hard to determine the subject's identity. The original image, repeated in Figure 2.2(c), is a 256×256 image which shows a much greater level of detail, and the subject can be recognised from the 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. Common choices are for 512×512 or 1024×1024 8-bit images which require 256 KB and 1 MB of storage, respectively. If we take a sequence of, say, 20 images for motion analysis, we will need more than 5 MB to store twenty 512×512 images. Even though memory continues to become cheaper, this can still impose high cost. But it is not just cost which motivates an investigation of the appropriate image size, the appropriate value for *N*. The main question is: are there theoretical guidelines for choosing it? The short answer is 'yes'; the long answer is to look at digital signal processing theory.

The choice of sampling frequency is dictated by the *sampling criterion*. Presenting the sampling criterion requires understanding of how we interpret signals in the *frequency domain*. The way in is to look at the Fourier transform. This is a highly theoretical topic, but do not let that put you off (it leads to image coding, like the JPEG format, so it is very useful indeed). The Fourier transform has found many uses in image processing and understanding; it might appear to be a complex topic (that's actually a horrible pun!) but it is a very rewarding one to study. The particular concern is number of points per unit area or the appropriate sampling frequency of (essentially, the value for *N*), or the rate at which pixel values are taken from, a camera's video signal.

2.3 The Fourier Transform

The *Fourier transform* is a way of mapping a signal into its component frequencies. **Frequency** 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/seconds; s = 1/Hz).

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

So why do we do this? We have not changed the signal, only its representation. We can now 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, 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:

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

where: $Fp(\omega)$ is the Fourier transform, and $\Im$ denotes the Fourier transform process;

ω is the **angular** frequency, $\omega = 2\pi f$ measured in **radians/s** (where the frequency f is the reciprocal of time t, $f = 1/t$);

j is the complex variable $j = \sqrt{-1}$ (electronic engineers prefer j to i since they cannot confuse it with the symbol for current; perhaps they don't want to be mistaken for mathematicians who use $i = \sqrt{-1}$)

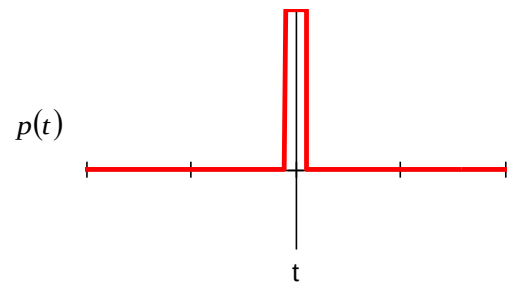
$p(t)$ is a **continuous** signal (varying continuously with time); and

$e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$ gives the frequency components in $p(t)$.

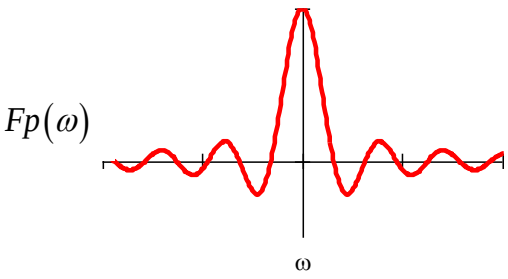
We can derive the Fourier transform by applying Equation 2.1 to the signal of interest. We can see how it works by constraining our analysis to simple signals. (We can then say that complicated signals are just made up by adding up lots of simple signals.) If we take a pulse which is of amplitude (size)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

$$p(t) = \begin{cases} A & \text{if} \quad -T/2 \le t \le T/2 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

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:



(a) pulse of amplitude A = 1



(b) Fourier transform

Figure 2.3 A Pulse and its Fourier Transform

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

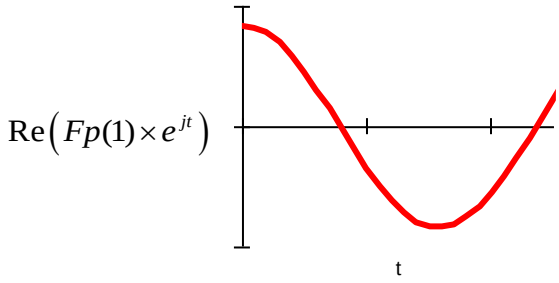
When we solve this we obtain an expression for $Fp(\omega)$:

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

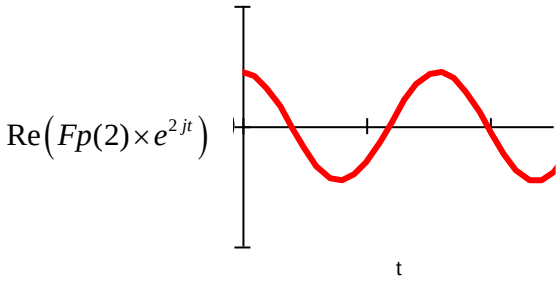
By simplification, using the relation $\sin(\theta) = (e^{j\theta} - e^{-j\theta})/2j$, then the Fourier transform of the pulse is:

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

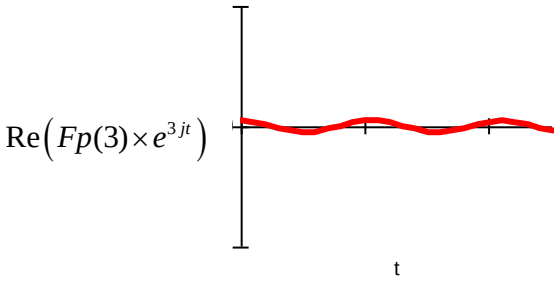
This is a version of the *sinc* function, sinc(x) = sin(x)/x. The original pulse, and its transform are illustrated in Figure 2.3. Equation 2.5 (as plotted in Figure 2.3(b)) suggests that a pulse is made up of a lot of low frequencies (the main body of the pulse) and a few higher frequencies (which give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



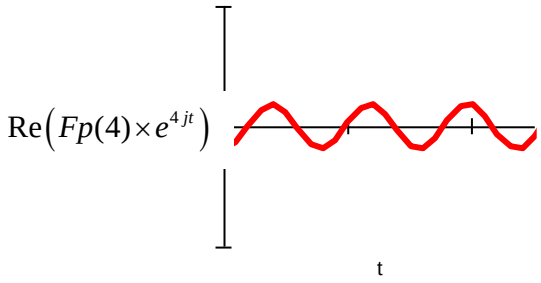
(**a**) contribution for ω = 1



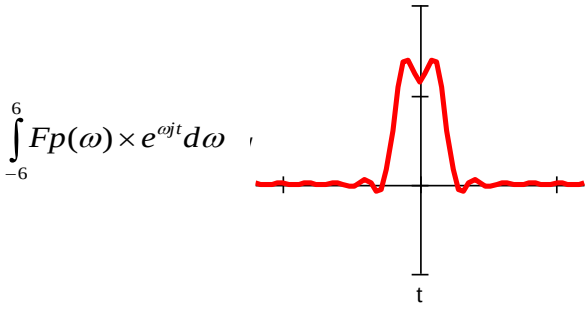
(**b**) contribution for ω = 2



(**c**) contribution for ω = 3



(**d**) contribution for ω = 4



(**e**) reconstruction by integration

Figure 2.4 Reconstructing a Signal from its 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, we should obtain the originally transformed signal. This process is illustrated in Figure 2.4 for the signal and transform illustrated in Figure 2.3. Note that since the Fourier transform is actually a **complex** number it has real and imaginary parts, and we only plot the **real** part here. A low frequency, that for $\omega = 1$, in Figure 2.4(a) contributes a large component of the original signal; a higher frequency, that for $\omega = 2$, contributes less as in Figure 2.4(b). This is because the transform coefficient is less for $\omega = 2$ than it is for $\omega = 1$. There is a very small contribution for

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

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

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

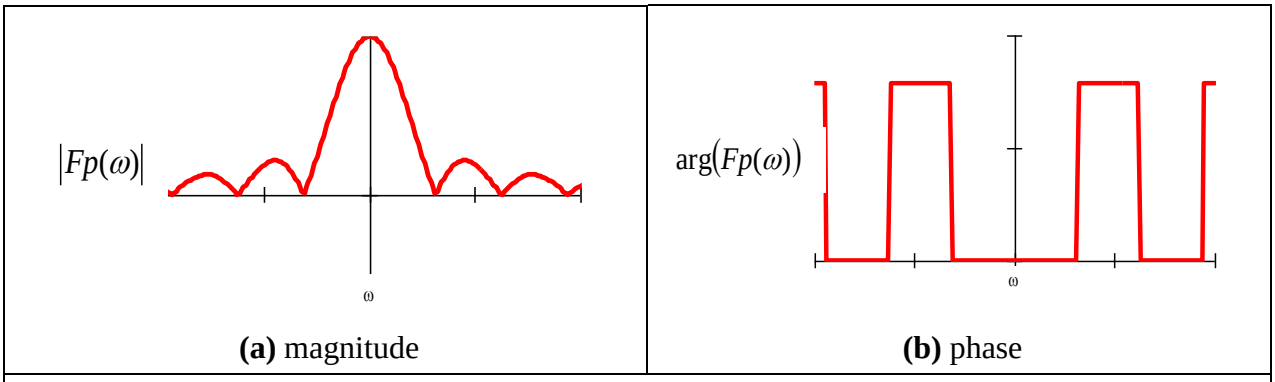
where $\mathrm{Re}(\ )$ and $\mathrm{Im}(\ )$ are the real and imaginary parts of the transform, respectively. The **magnitude** of the transform is then:

$$\left|Fp(\omega)\right| = \sqrt{\mathrm{Re}\left(Fp(\omega)\right)^2 + \mathrm{Im}\left(Fp(\omega)\right)^2} \tag{2.7}$$

and the **phase** is:

$$\arg\left(Fp(\omega)\right) = \tan^{-1}\left(\frac{\mathrm{Im}\left(Fp(\omega)\right)}{\mathrm{Re}\left(Fp(\omega)\right)}\right) \tag{2.8}$$

where the signs of the real and the imaginary components can be used to determine which quadrant the phase (argument) 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).



(a) magnitude **(b)** phase

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*. This was the process illustrated in Figure **2.4,** the process by which we reconstructed the pulse from its transform components. The inverse FT calculates $p(t)$ from $Fp(\omega)$ by the inverse transformation $\Im^{-1}$:

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

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, the Fourier transform of the convolution of two signals is

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

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

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

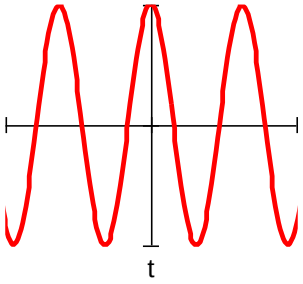
As such, the frequency domain dual of convolution is multiplication; the **convolution integral** can be performed by **inverse** Fourier transformation of the **product** of the transforms of the two signals. A frequency domain representation essentially presents signals in a different way but it also 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

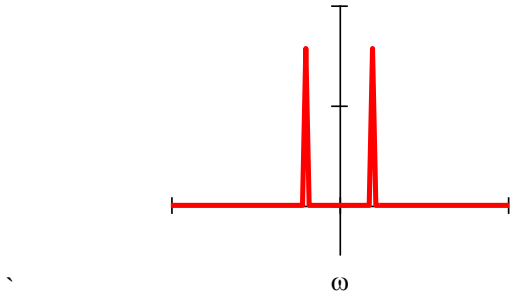
$$p_1(t) \otimes p_2(t) = \int_{-\infty}^{\infty} p_1(\tau)\, p_2(t+\tau)\, d\tau \tag{2.13}$$

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

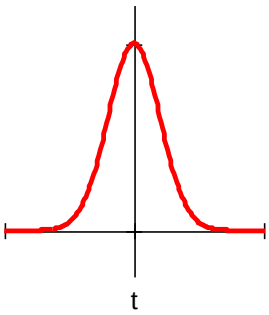
Time Domain Signals	Frequency Domain Spectra



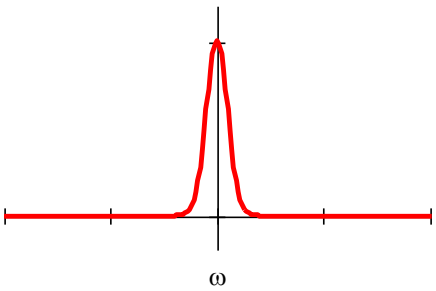
(a) cosine wave



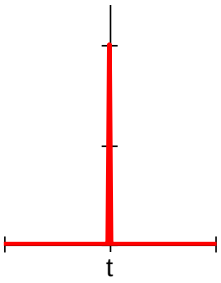
(b) Fourier transform of cosine wave



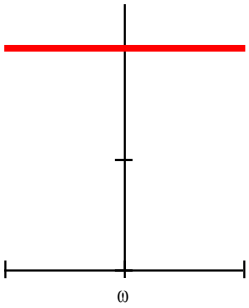
(c) Gaussian function



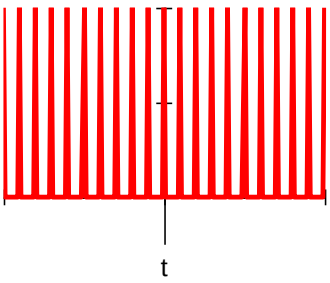
(d) spectrum of Gaussian function



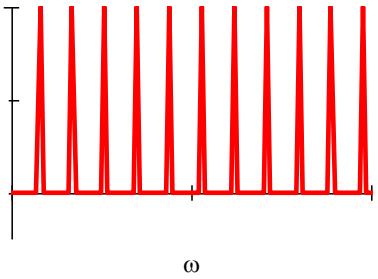
(e) delta function



(f) frequency content of delta function



(g) sampling function in time domain



(h) transform of sampling function

Figure 2.6 Fourier Transform Pairs

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

$$delta(t-\tau) = \begin{cases} 1 & \text{if} \quad t = \tau \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

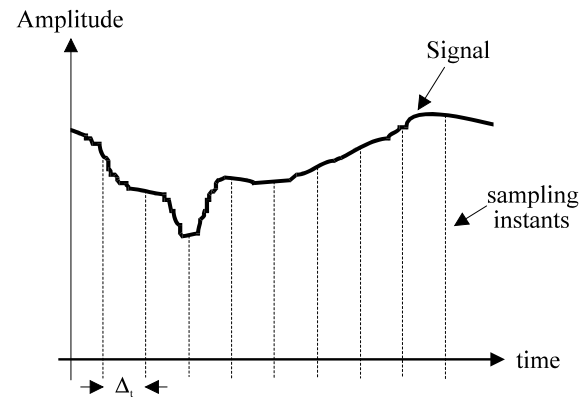
The relationship between a signal's time domain representation and its frequency domain version is also known as a *transform pair*: the transform of a pulse (in the time domain) is a sinc function in the frequency domain. Since the transform is symmetrical, the Fourier transform of a sinc function is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

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

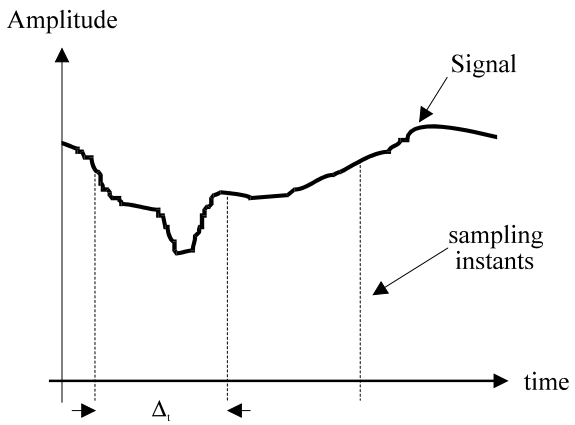
Finally, Figures 2.6(g) and (h) show that the transform of a set of uniformly-spaced delta functions is another set of uniformly-spaced delta functions, but with a different spacing. The spacing in the frequency domain is the reciprocal of the spacing in the time domain. By way of a (non-mathematical)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

2.4 The Sampling Criterion

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



(a) sampling at high frequency



(b) sampling at low frequency

Figure 2.7 Sampling at Different Frequencies

We can understand the process better in the frequency domain. Let us consider a time-variant signal which has a range of frequencies between $-f_{max}$ and f_{max} as illustrated in Figure 2.9(b). This range of frequencies is shown by the Fourier transform where the signal's **spectrum** exists only between these frequencies. This function is sampled every Δ_t s: this is a sampling function of spikes occurring every Δ_t s. The Fourier transform of the sampling function is a series of spikes separated by $f_{sample} = 1/\Delta_t$ Hz. The Fourier pair of this transform was illustrated earlier, Figures 2.6(g) and (h).

The sampled signal is the result of multiplying the time-variant signal by the sequence of spikes, this gives samples that occur every Δ_t s, and the sampled signal is shown in Figure 2.9(a). These are the outputs of the A/D converter at sampling instants. The frequency domain analogue of this sampling process is to **convolve** the spectrum of the time-variant signal with the spectrum of the sampling function. Convolving the signals, the convolution process, implies that we take the spectrum of one, **flip** it along the horizontal axis and then **slide** it across the other. Taking the spectrum of the time-variant signal and sliding it over the spectrum of the spikes, results in a spectrum where the spectrum of the original signal is **repeated** every $1/\Delta_t$ Hz, f_{sample} in Figure **2.9(b-d). If the spacing between samples is Δ_t, the repetitions of the time-variant signal's** spectrum are spaced at intervals of $1/\Delta_t$, as in Figure 2.9(b). If the sample spacing is large, then the time-variant signal's spectrum is replicated close together and the spectra **collide**, or interfere, as in Figure 2.9(d). The spectra just **touch** when the sampling frequency is **twice** the maximum frequency in the signal. If the frequency domain spacing, f_{sample}, is **more** than twice the maximum frequency, f_{max}, the spectra do **not** collide or interfere, as in Figure 2.9(c). If the sampling frequency exceeds twice the maximum frequency then the spectra cannot collide. This is the Nyquist sampling criterion:

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

If we do not obey Nyquist's sampling theorem the spectra collide. When we inspect the sampled signal, whose spectrum is within $-f_{max}$ to f_{max}, wherein the spectra collided, the corrupt spectrum

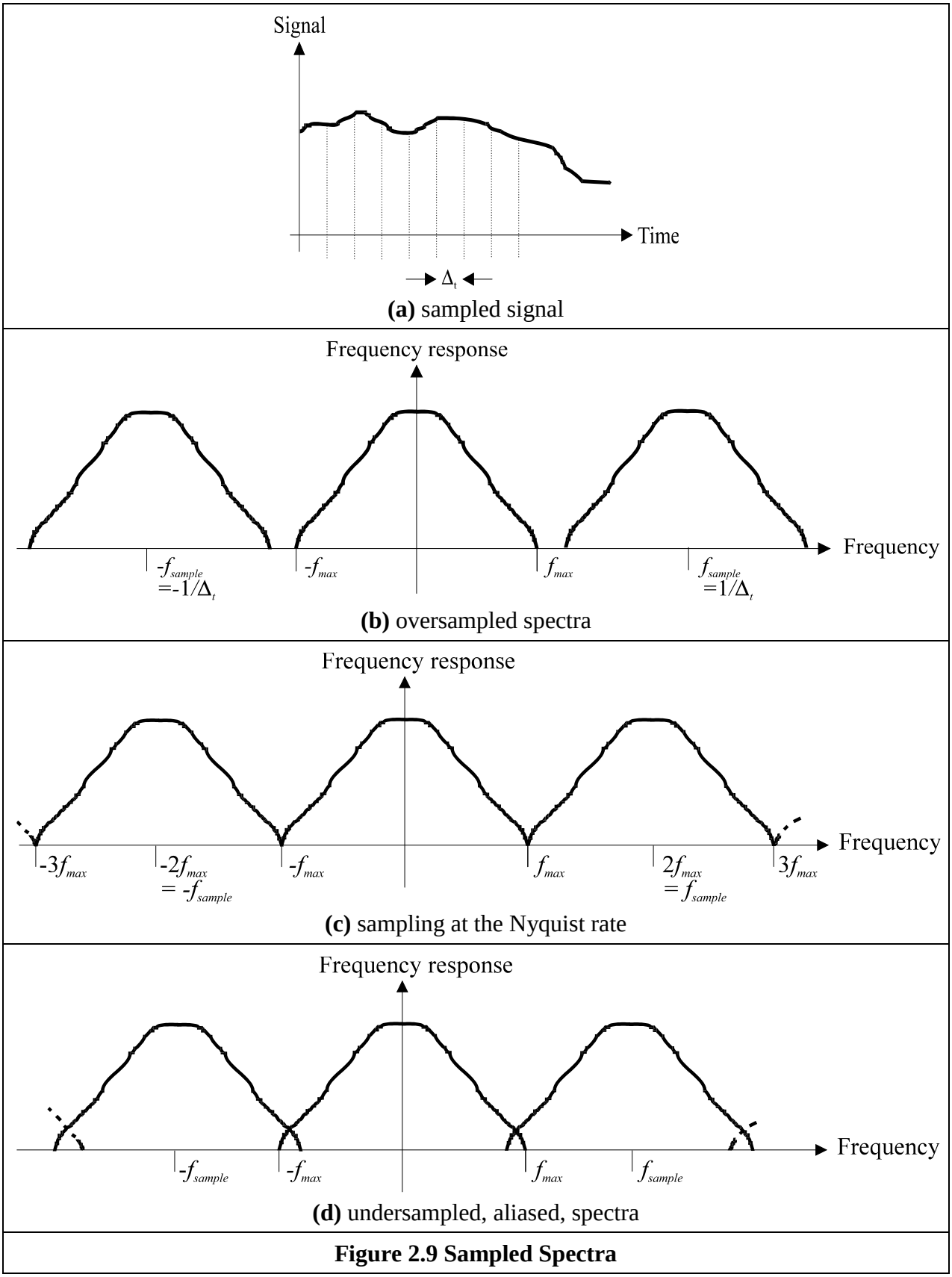
implies that by virtue of sampling, we have **ruined** some of the information. If we were to attempt to reconstruct a signal by inverse Fourier transformation of the sampled signal's spectrum, processing Figure 2.9(d) would lead to the wrong signal whereas inverse Fourier transformation of the frequencies between $-f_{max}$ and f_{max} in Figures 2.9(b) and (c) would lead back to the original signal. This can be seen in computer images as illustrated in Figure 2.8 which show an image of a group of people (the computer vision research team at Southampton) displayed at different spatial resolutions (the contrast has been increased to the same level in each sub-image, so that the effect we want to demonstrate should definitely show up in the print copy). Essentially, the people become less distinct in the lower resolution image, Figure 2.8(b). Now, look closely at the window blinds behind the people. At higher resolution, in Figure 2.8(a), these appear as normal window blinds. In Figure 2.8(b), which is sampled at a much lower resolution, a new pattern appears: the pattern appears to be curved - and if you consider the blinds' relative size the shapes actually appear to be much larger than normal window blinds. So by reducing the resolution, we are seeing something different, an **alias** of the true information – something that is not actually there at all, but appears to be there by result of sampling. This is the result of sampling at too low a frequency: if we sample at high frequency, the interpolated result matches the original signal; if we sample at **too low** a frequency we can get the **wrong** signal. (For these reasons people on television tend to wear non-chequered clothes – or should not!). Note that this effect can be seen, in the way described, in the printed version of this book. This is because the printing is technology is very high resolution. If you were to print this page offline (and sometimes even to view it), e.g. from a Google Books sample, the nature of the effect of aliasing depends on the resolution of the printed image, and so the aliasing effect might also be seen in the high resolution image as well as in the low resolution version – which rather spoils the point.

(a) high resolution

(c) low resolution (aliased)

Figure 2.8 Aliasing in Sampled Imagery (Digital Print causes problems here)

In art, Dali's picture "*Gala Contemplating the Mediterranean Sea, which at 20 meters becomes the portrait of Abraham Lincoln*" (Homage to Rothko) is a classic illustration of sampling. At high resolution you see a detailed surrealist image, with Mrs Dali as a central figure. Viewed from a distance - or for the shortsighted, without your spectacles on - the image becomes a (low resolution) picture of Abraham Lincoln. For a more modern view of sampling [Unser00] is well worth a look. The *compressive sensing* approach [Donoho06] takes advantage of the fact that many signals have components that are significant, or nearly zero, leading to cameras which acquire significantly fewer elements to represent an image. This provides an alternative basis for compressed image acquisition without loss of resolution.



(a) sampled signal

(b) oversampled spectra

(c) sampling at the Nyquist rate

(d) undersampled, aliased, spectra

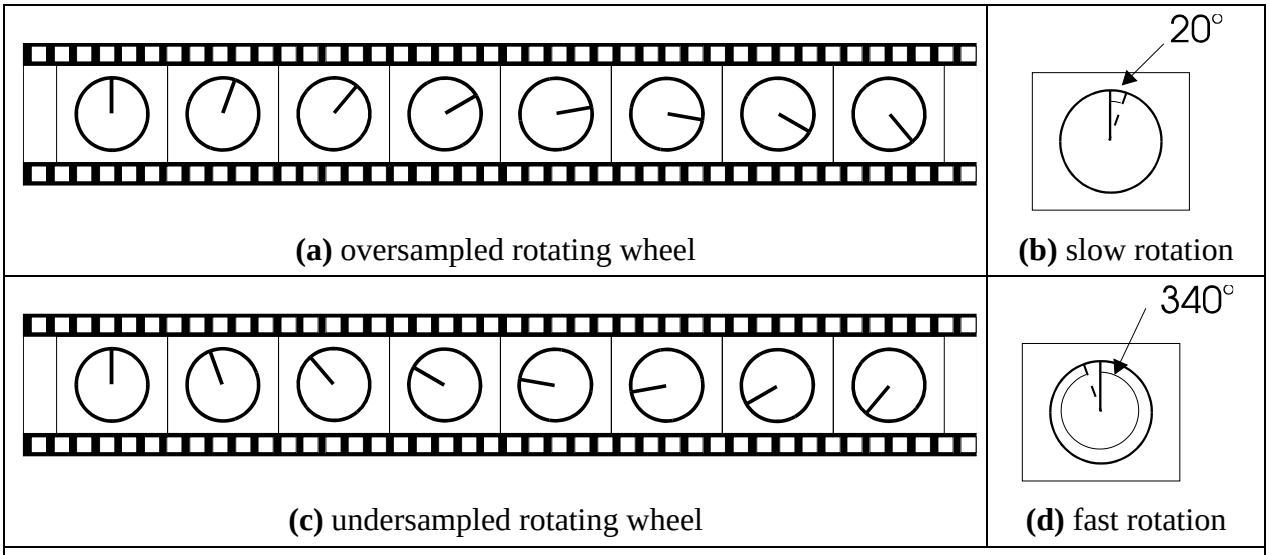
Figure 2.9 Sampled Spectra

Obtaining the wrong signal is called *aliasing*: our interpolated signal is an alias of its proper form. Clearly, we want to **avoid** aliasing, so according to the sampling theorem we must sample at twice the maximum frequency of the signal coming out of the camera. The maximum frequency

is defined to be 5.5 MHz so we must sample the camera signal at 11 MHz. (For information, when using a computer to analyse speech we must sample the speech at a minimum frequency of 12 kHz since the maximum speech frequency is 6 kHz.) Given the timing of a video signal, sampling at 11 MHz implies a minimum image resolution of 576×576 pixels. This is unfortunate: 576 is not an integer power of two which has poor implications for storage and processing. Accordingly, since many image processing systems have a maximum resolution of 512×512, they must anticipate aliasing. This is mitigated somewhat by the observations that:

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

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



(a) oversampled rotating wheel **(b)** slow rotation

(c) undersampled rotating wheel **(d)** fast rotation

Figure 2.10 Correct and Incorrect Apparent Wheel Motion

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

2.5 The Discrete Fourier Transform (DFT)

2.5.1 One Dimensional Transform

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

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

where the scaling coefficient $1/N$ ensures the d.c. coefficient $\mathbf{Fp}_0$ is the average of all samples. Eq. 2.15 is a discrete analogue of the continuous Fourier transform: the continuous signal is replaced by a set of samples, the continuous frequencies by sampled ones, and the integral is replaced by a summation. If the DFT is applied to samples of a pulse in a window from sample 0 to sample $N/2-1$ (when the pulse ceases), the equation becomes:

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

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

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

the discrete Fourier transform of a sampled pulse is given by:

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

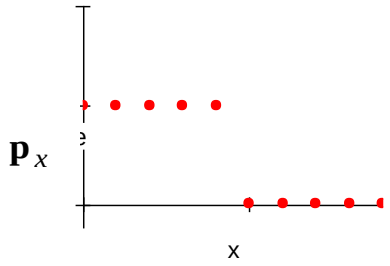
By rearrangement, we obtain:

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

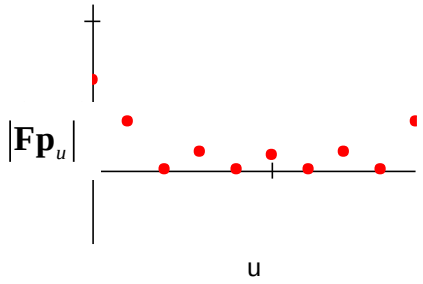
The modulus of the transform is:

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

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



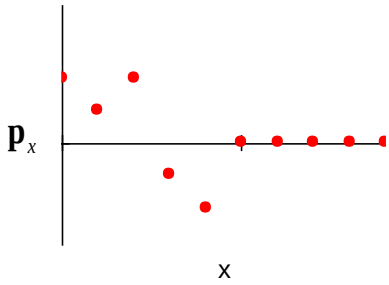
(a) sampled pulse



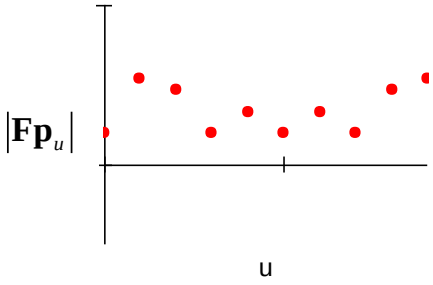
(b) DFT of sampled pulse

Figure 2.11 Transform Pair for Sampled Pulse

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



(a) sampled signal



(b) transform of sampled signal

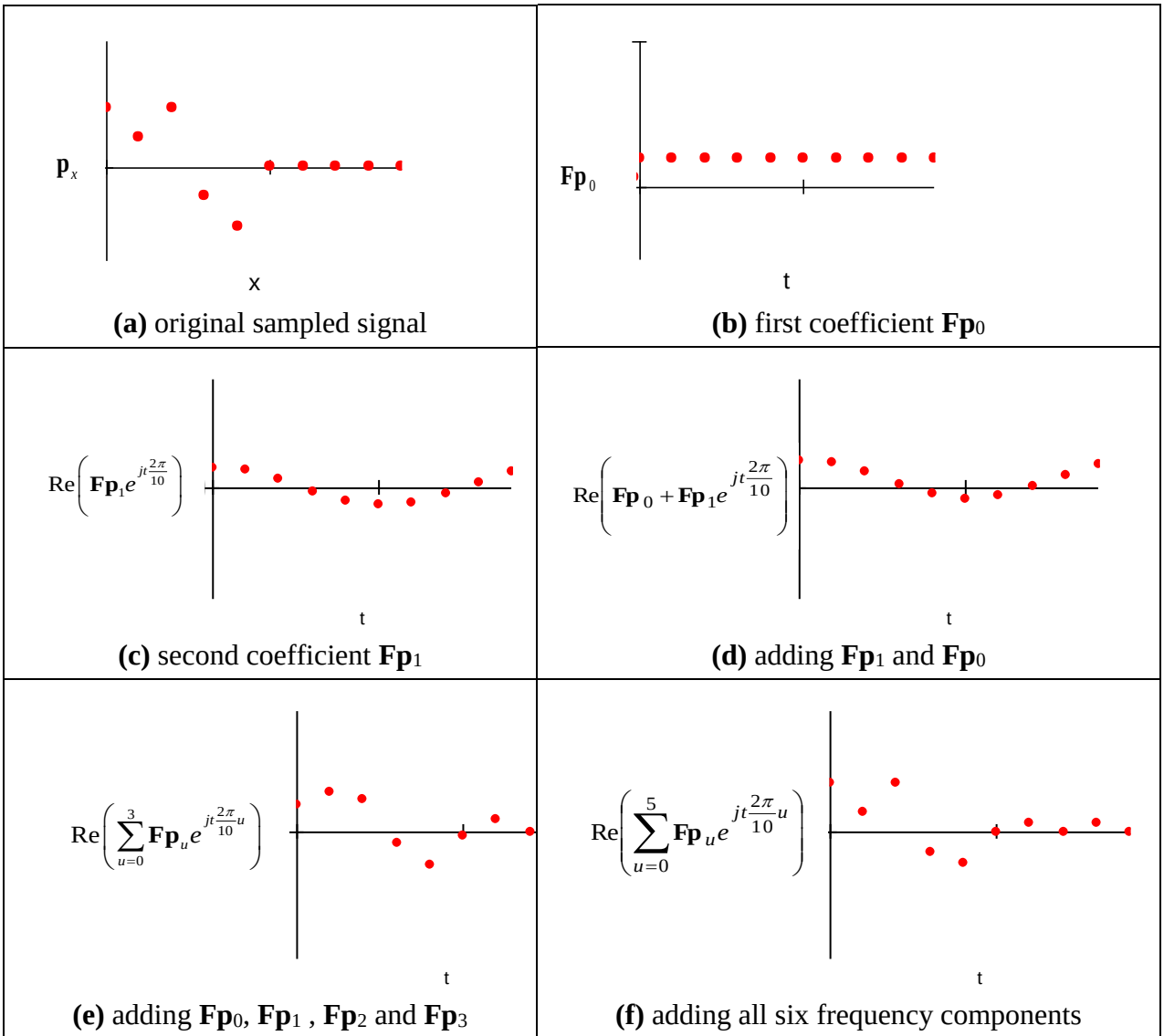
Figure 2.12 A Sampled Signal and its Discrete Transform

The Fourier transform in Figure 2.12(b) can be used to reconstruct the original signal in Figure 2.12(a), as illustrated in Figure 2.13. Essentially, the coefficients of the Fourier transform tell us how much there is of each of a set of sinewaves (at different frequencies), in the original signal. The lowest frequency component $\mathbf{Fp}_0$, for zero frequency, is called the *d.c. component* (it is constant and equivalent to a sinewave with no frequency) and it represents the **average** value of the samples. Adding the contribution of the first coefficient $\mathbf{Fp}_0$, Figure 2.13(b), to the contribution of the second coefficient $\mathbf{Fp}_1$, Figure 2.13(c), is shown in Figure 2.13(d). This shows how addition of the first two frequency components approaches the original sampled pulse. The approximation improves when the contribution due to the fourth component, $\mathbf{Fp}_3$, is included, as shown in Figure 2.13(e). Finally, adding up all six frequency components gives a close approximation to the original signal, as shown in Figure 2.13(f).

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

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

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



(a) original sampled signal

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

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

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

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

(f) adding all six frequency components

Figure 2.13 Signal Reconstruction from its Transform Components

.

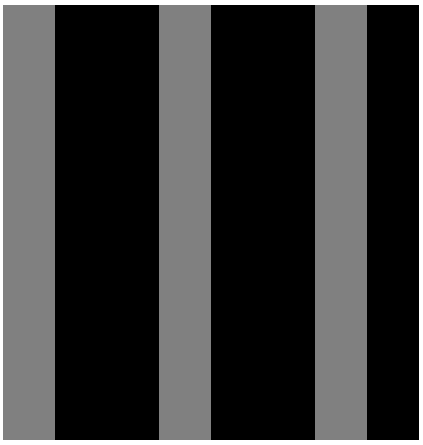
2.5.2 Two Dimensional Transform

Equation 2.15 gives the DFT of a one-dimensional signal. We need to generate Fourier transforms of images so we need a *two-dimensional discrete Fourier transform*. This is a transform of pixels (sampled picture points) with a two dimensional spatial location indexed by coordinates *x* and *y*.

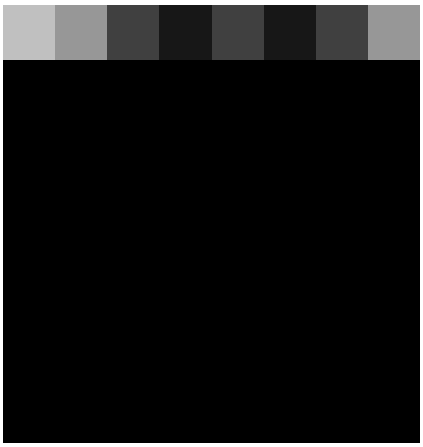
This implies that we have two dimensions of frequency, u and v, which are the horizontal and vertical spatial frequencies, respectively. Given an image of a set of vertical lines, the Fourier transform will show only horizontal spatial frequency. The vertical spatial frequencies are zero since there is no vertical variation along the y axis. The two dimensional Fourier transform evaluates the frequency data, $\mathbf{FP}_{u,v}$, from the $N \times N$ pixels $\mathbf{P}_{x,y}$ as:

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

Where the scaling coefficient $1/N^2$ makes the d.c. coefficient $\mathbf{FP}_{0,0}$ equal the average of all points in the image (in Matlab the scaling coefficient is 1.0). The Fourier transform of an image can actually be obtained **optically** by transmitting a laser through a photographic slide and forming an image using a lens. The Fourier transform of the image of the slide is formed in the front focal plane of the lens. This is still restricted to transmissive systems whereas reflective formation would widen its application potential considerably (since optical computation is just slightly faster than its digital counterpart). The magnitude of the 2D DFT to an image of vertical bars (Figure 2.14(a)) is shown in Figure 2.14(b). This shows that there are only horizontal spatial frequencies; the image is constant in the vertical axis and there are no vertical spatial frequencies.



(a) image of vertical bars



(b) Fourier transform of bars

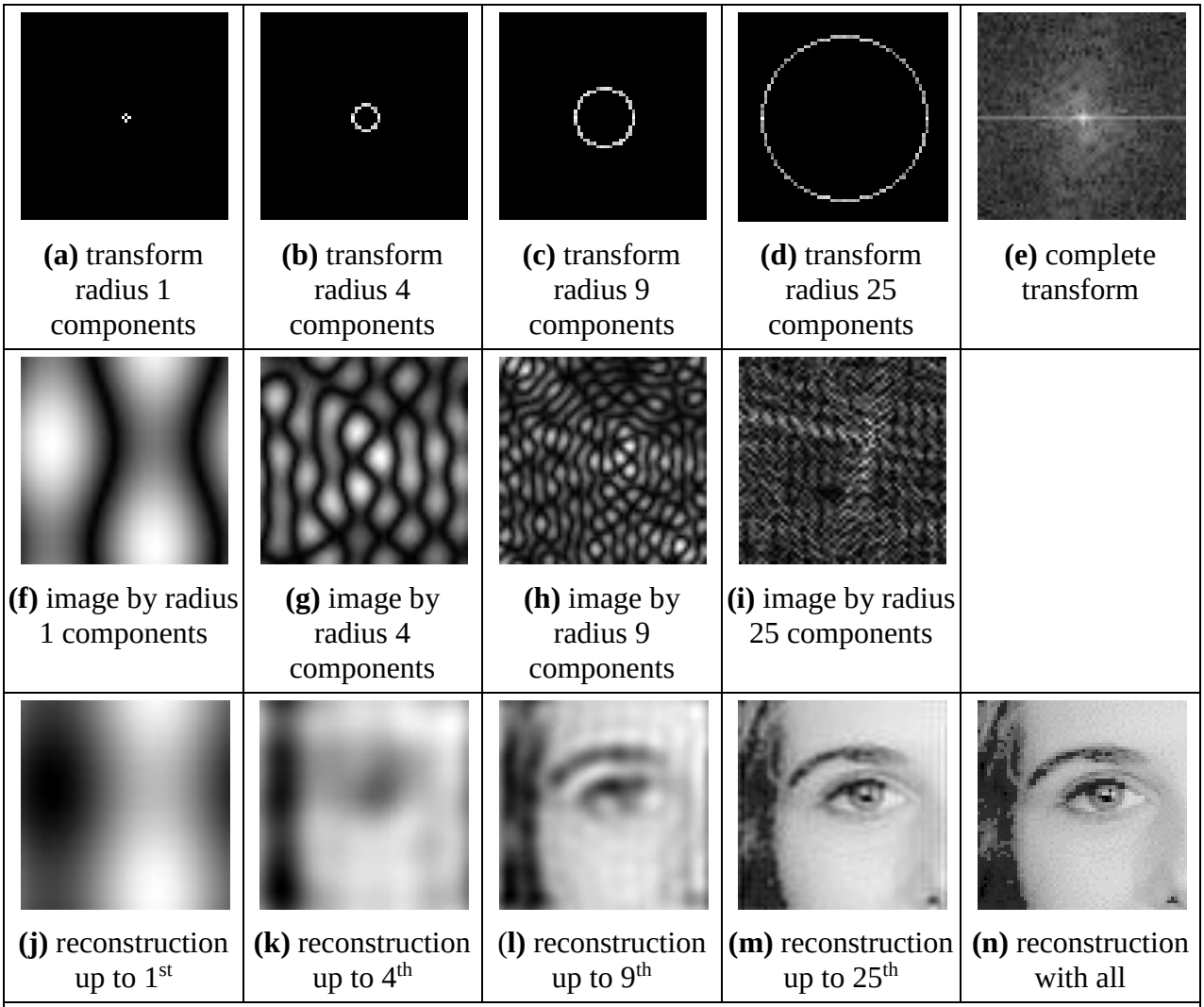
Figure 2.14 Applying the 2D Discrete Fourier Transform

The *two-dimensional (2D) inverse DFT* transforms from the frequency domain back to the image domain, to **reconstruct** the image. The 2D inverse DFT is given by:

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

The contribution of different frequencies is illustrated in Figure 2.15 where we have images showing in the first row (a)-(d) the position of the image transform components (presented as log[magnitude]), in the second row (f)-(i) the image constructed from that single component, and in the third row (j)-(m) the reconstruction (by the inverse FT) using frequencies up to and including that component. There is also the image of the magnitude of the Fourier transform, (e). We shall take the transform components from a circle centred at the middle of the transform image. In Figure 2.15 the first column is the transform components at radius 1 (which are low frequency components), the second column is the radius 4 components, the third column is at radius 9 and the fourth column is the radius 25 components (the higher frequency components).

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



(a) transform radius 1 components	**(b)** transform radius 4 components	**(c)** transform radius 9 components	**(d)** transform radius 25 components	**(e)** complete transform
(f) image by radius 1 components	**(g)** image by radius 4 components	**(h)** image by radius 9 components	**(i)** image by radius 25 components	
(j) reconstruction up to 1st	**(k)** reconstruction up to 4th	(**l**) reconstruction up to 9th	**(m)** reconstruction up to 25th	**(n)** reconstruction with all

Figure 2.15 Image Reconstruction and Different Frequency Components

One of the important properties of the FT is *replication* which implies that the transform **repeats** in frequency up to **infinity**, as indicated in Figure 2.9 for 1D signals. To show this for 2D signals, we need to investigate the Fourier transform, originally given by $\mathbf{FP}_{u,v}$, at integer multiples of the

number of sampled points $\mathbf{FP}_{u+mM,v+nN}$ (where m and n are integers). The Fourier transform $\mathbf{FP}_{u+mM,v+nN}$ is, by substitution in Equation 2.22:

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

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

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

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

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

$$\frac{1}{MN}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} = \frac{1}{MN}\sum_{x=0}^{N-1}\left\{\sum_{y=0}^{N-1}\mathbf{P}_{x,y}e^{-j\left(\frac{2\pi}{N}\right)(vy)}\right\}e^{-j\left(\frac{2\pi}{M}\right)(ux)} \tag{2.27}$$

where M and N are the numbers of columns and rows, respectively. Equation 2.27 shows how separability is achieved, since the inner term expresses transformation along one axis (the y axis), and the outer term transforms this along the other (the x axis).

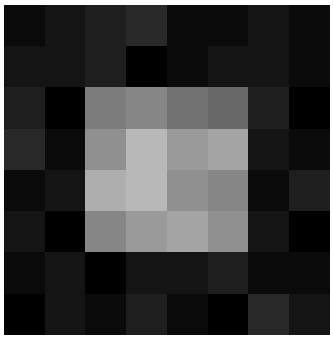
```
function [Fourier] = F_transform(image)

% New image is Discrete Fourier Transform (DFT) of image
% Usage: new image = F_transform(image)

image=double(image);
[rows, cols] = size(image);
%we deploy equation 2.27, so that we can handle non square images
for u=1:cols % along the horizontal axis
    for v=1:rows % down the vertical axis
        sumx=0;
        for x=1:cols
            %first we transform the rows
            sumy=0;
            for y=1:rows
                sumy=sumy+image(y,x)*exp(-1j*2*pi*(v*y)/rows);
            end
            %then we do the columns
            sumx=sumx+sumy*exp(-1j*2*pi*(u*x)/cols);
        end
        Fourier(v,u) = sumx/(rows*cols);
    end
```

```
end
```

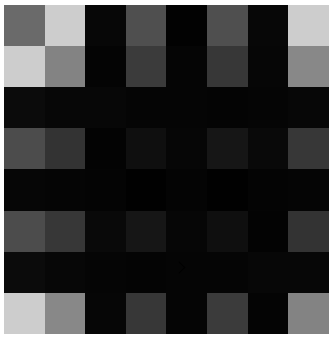
Code 2.1 2D DFT, implementing Equation 2.27

Since the computational cost of a 1D FFT of N points is O(Nlog(N)), the cost (by separability) for the 2D FFT is O(N^2log(N)) whereas the computational cost of the 2D DFT is O(N^3). This implies a considerable saving since it suggests that the FFT requires much less time, particularly for large image sizes (so for a 1024×1024 image, if the FFT takes seconds, the DFT will take minutes). The 2D FFT is available in Matlab using the `fft2` function which gives a result equivalent to Equations 2.22 or 2.27. You can note the difference in time between executing the code in Code 2.1 (or our Python version) and Matlab's own FFT operator (note there is some difference due to compiled vs interpreted code; do not run the basic version on a large image as it will take a very long time). The inverse 2D FFT, Equation 2.23, can be implemented using the Matlab `ifft2` function. (The difference between many Fourier transform implementations essentially concerns the chosen scaling factor, though the order of the frequency components differs from the basic equations in the Matlab functions.) The direct Matlab implementation of the 2D DFT in Equation 2.27 is given in Code 2.1. This is simply called using the command `b=F_Transform(a)` and the routine enforces the change from an integer format to double precision, as needed when using complex numbers in Matlab. It is easier to work in double precision throughout when developing code; integer formats can be used to speed real implementations (with caution: an early ARIANE space rocket blew up given erroneous conversion of a 32-bit integer to a 16-bit version).

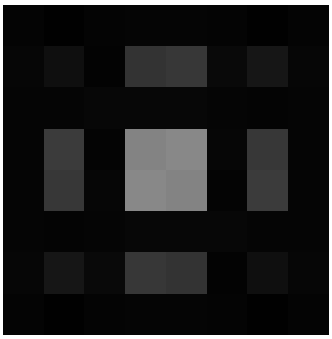
One difficulty is that the nature of the Fourier transform produces an image which, at first, is difficult to interpret. The Fourier transform of an image gives the frequency components. The position of each component reflects its frequency: **low** frequency components are **near** the origin and **high**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



(a) image of square



(b) original DFT



(c) rearranged DFT

Figure 2.16 Rearranging the 2D DFT for Display Purposes.

To rearrange the image so that the d.c. component is in the centre, the frequency components need to be reordered. This can be achieved simply by multiplying each image point $\mathbf{P}_{x,y}$ by $-1^{(x+y)}$. Since $\cos(-\pi) = -1$, then $-1 = e^{-j\pi}$ (the minus sign is introduced just to keep the analysis neat) so we obtain the transform of the multiplied image as:

$$\begin{aligned}\frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)}\times -1^{(x+y)} &= \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)}\times e^{-j\pi(x+y)}\\ &= \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}e^{-j\left(\frac{2\pi}{N}\right)\left(\left(u+\frac{N}{2}\right)x+\left(v+\frac{N}{2}\right)y\right)}\\ &= \mathbf{FP}_{u+\frac{N}{2},v+\frac{N}{2}}\end{aligned} \tag{2.28}$$

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

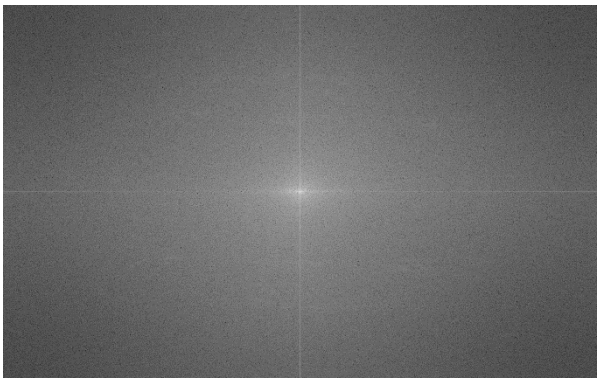
```
'''
Feature Extraction and Image Processing
Mark S. Nixon & Alberto S. Aguado
Chapter 2: Intensity rearrangement
'''
# Set utility functions
from ImageSupport import imageReadL, showImageL, createImageL,
# Iteration
from timeit import itertools
'''
Parameters:
    pathToDir = Input image directory
    imageName = Input image name
'''
pathToDir = "../../Images/Chapter2/Input/"
imageName = "group.png"
# Read image into array
inputImage, width, height  = imageReadL(pathToDir + imageName)
# Show input image
showImageL(inputImage)
# Create image to store the normalization
outputImage = createImageL(width, height)
# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):
    # Normalize the pixel value according to the range
    outputImage[y,x] = inputImage[y,x]*-1^(x+y)
# Show output image and plot histogram
showImageL(outputImage)
```

Code 2.2 Reordering for Transform Calculation

The full effect of the Fourier transform is shown by application to an image of much higher resolution. Figure 2.17(a) shows the image of a group of people and Figure 2.17(b) shows its transform. The transform reveals that much of the information is carried in the **lower** frequencies

since this is where most of the spectral components concentrate. This is because the image has many regions where the brightness does not change a lot, such as in the sky. The **high** frequency components reflect **change** in intensity. Accordingly, the higher frequency components arise from the hair (and that awful feather!) and from the borders of features of the human face, such as the nose and eyes.

(a) image of group

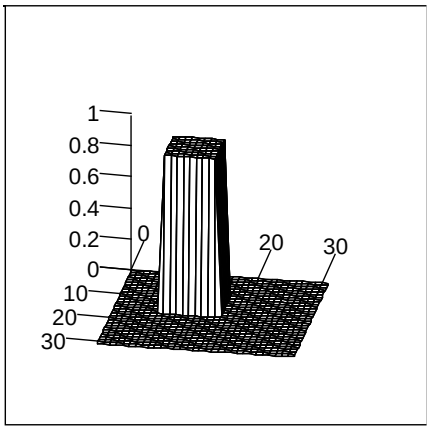


(b) transform of image of group

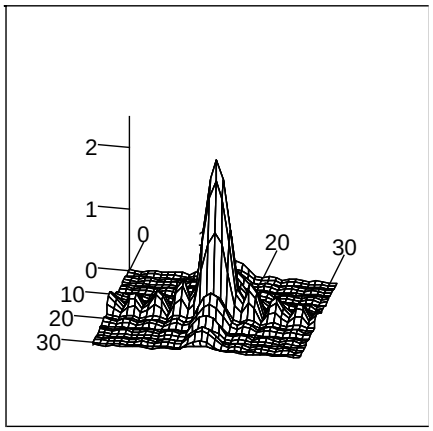
Figure 2.17 Applying the Fourier Transform to the Image of a Group of People

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

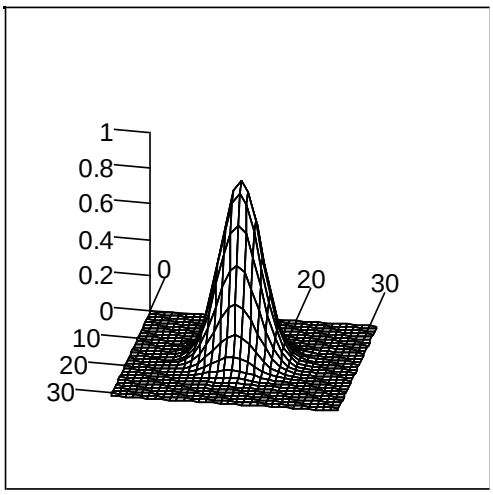
Image Domain	**Transform Domain**



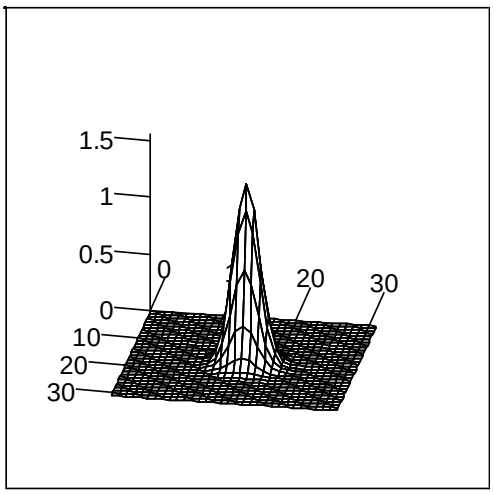
(a) square



(b) 2D sinc function



(c) Gaussian



(d) Gaussian

Figure 2.18 2D Fourier Transform Pairs

2.6 Properties of the Fourier Transform

2.6.1 Shift Invariance

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

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

Accordingly, the magnitude of the Fourier transform is:

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

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

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

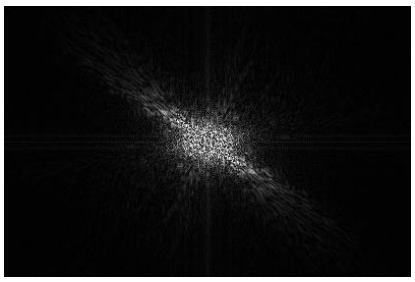
The implementation of a `shift` operator, Code 2.3, uses the modulus operation `mod` to enforce the cyclic shift. The arguments fed to the function are: the image to be shifted (pic), the horizontal shift along the *x* axis (shift_x), and the vertical shift along the *y* axis (shift_y). (As the indices to matrices go from 1 to N in Matlab, there is an odd manoevre in the modulus function.)

```
function [shifted] = Shift(image,shift_x,shift_y)
```

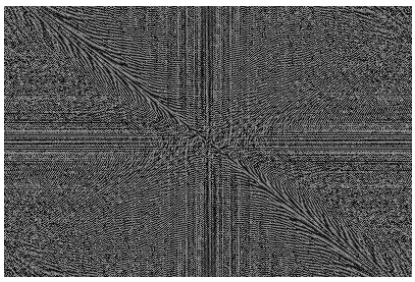
```
function shifted = shift(image,shift_x,shift_y)
%Shifts an image horizontally, and wraps round
%
%  Usage: new image = shift_y(image,amount x axis, amount y axis)
%
%  Parameters: image      - array of points
%              shift_x    - amount of x shift
%              shift_y    - amount of y shift

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

%and shift it (with side to side and top to bottom wrap around)
for x=1:cols %% along the horizontal axis
    for y=1:rows %% down the vertical axis
        shifted(y,x)=image(mod(y+shift_y-1,rows)+1,mod(x+shift_x-1,cols)+1);
    end
end
```

Code 2.3 Shifting an Image

(a) original image

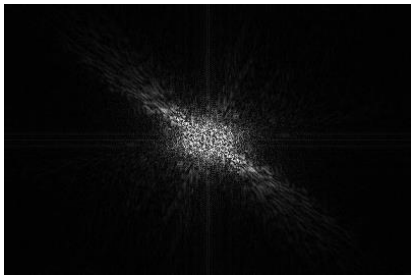


(b) magnitude of Fourier transform of original image

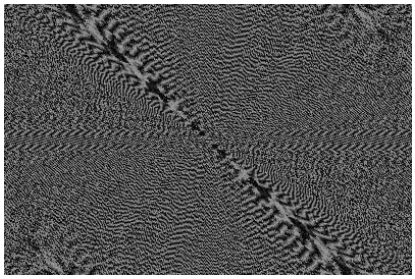


(c) phase of Fourier transform of original image

(d) shifted image



(e) magnitude of Fourier transform of shifted image



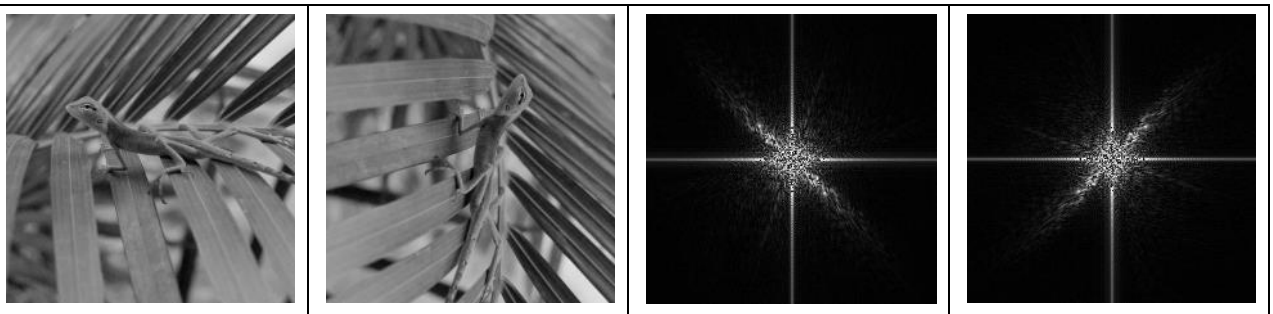
(f) phase of Fourier transform of shifted image

Figure 2.19 Illustrating Shift Invariance

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

The differing phase implies that, in application, the magnitude of the Fourier transform of a face, say, will be the same irrespective of the position of the face in the image (i.e. the camera or the subject can move up and down), assuming that the face is much larger than its image version.

This implies that if the Fourier transform is used to analyse an image of a human face or one of cloth, to describe it by its spatial frequency, we do not need to control the position of the camera, or the object, precisely.



(a) original image	**(b)** rotated image	**(c)** transform of original image	**(d)** transform of rotated image

Figure 2.20 Illustrating Rotation

2.6.2 Rotation

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

This implies that if the frequency domain properties are to be used in image analysis, via the Fourier transform, the orientation of the original image needs to be known, or fixed. It is often possible to fix orientation, or to estimate its value when a feature's orientation cannot be fixed. Alternatively, there are techniques to impose invariance to rotation, say by translation to a polar representation, though this can prove to be complex.

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

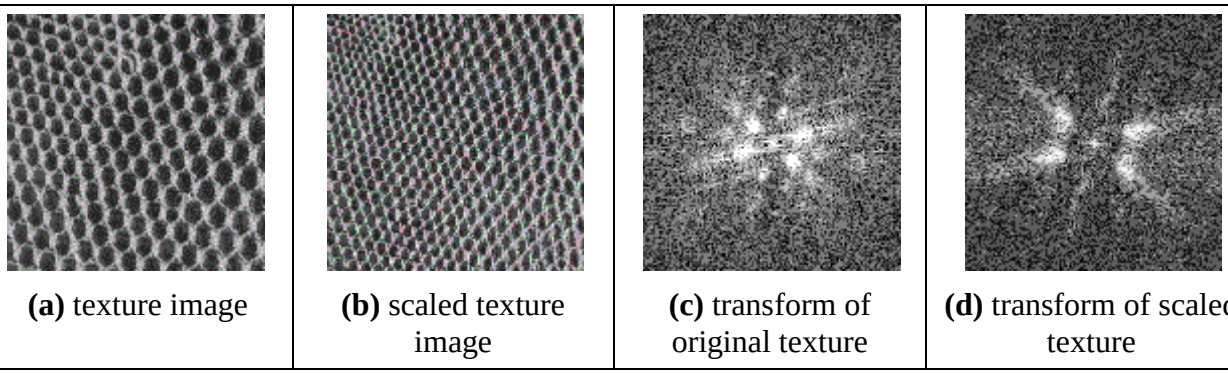
2.6.3 Frequency Scaling

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

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

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

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



(a) texture image	**(b)** scaled texture image	**(c)** transform of original texture	**(d)** transform of scaled texture

Figure 2.21 Illustrating Frequency Scaling

2.6.4 Superposition (Linearity)

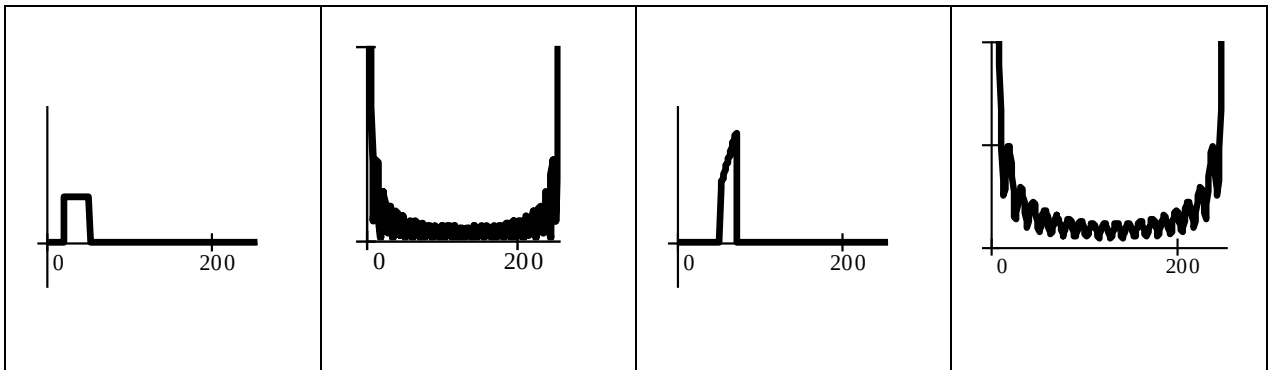
The *principle of superposition* is very important in systems analysis. Essentially, it states that a system is linear if its response to two combined signals equals the sum of the responses to the individual signals. Given an output O which is a function of two inputs I_1 and I_2, the response to signal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

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

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

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

In application this suggests that we can separate images by looking at their frequency domain components. This is illustrated for one-dimensional signals in Figure 2.22. One signal is shown in Figure 2.22(a) and a second is shown in Figure 2. 22(c). The Fourier transforms of these signals are shown in Figures 2.22(b) and (d). The addition of these signals is shown in Figure 2.22(e) and its transform in Figure 2.22(f). The Fourier transform of the added signals differs little from the addition of their transforms, Figure 2.22(g). This is confirmed by subtraction of the two, Figure 2.22(d) (some slight differences can be seen, but these are due to numerical error).



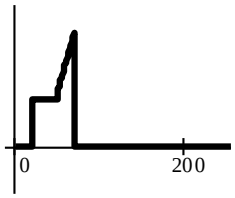
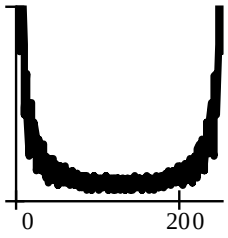
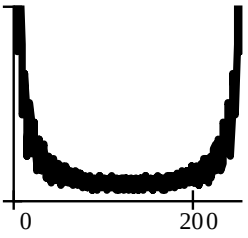
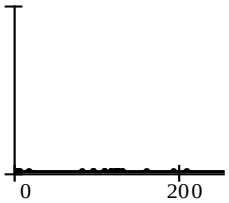
(a) signal 1	**(b)** $\Im$ (signal 1)	**(c)** signal 2	**(d)** $\Im$ (signal 2)
			
(e) signal 1 + signal 2	**(f)** $\Im$ (signal 1 + signal 2)	**(g)** $\Im$ (signal 1) + $\Im$ (signal 2)	**(h)** difference: (f)-(g)

Figure 2.22 Illustrating Superposition

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

2.6.5 The Importance of Phase

You might reasonably ask: "what is the importance of phase?". It is actually a pretty reasonable question. Phase is about how signals are **arranged** (or add up), whereas magnitude is about how **big** the signals are. As such, one might intuitively think that the magnitude is more important than the phase. It is more complex than this: phase can actually be viewed to be more important (though you need both magnitude and phase to reconstruct a signal). To illustrate this we shall take two images, one of the eye and the other of an ear (OK, ears are rather ugly but they are unique to their owner) as shown in Figure 2.**23**. Here we have reconstructed images by taking the magnitude of one image and the phase (the argument) of another. From Eq 2.6 we have

$$\begin{aligned} Fp(\omega) &= \mathrm{Re}\left(Fp(\omega)\right) + j\,\mathrm{Im}\left(Fp(\omega)\right) \\ &= magnitude \times \cos\left(phase\right) + j \times magnitude \times \sin\left(phase\right) \end{aligned} \tag{2.35}$$

where magnitude and phase are calculated according to Eqs. 2.7 and 2.8, respectively. The rearranged Fourier transform is then

$$Fp(\omega) = \left|Fp(\text{image 1})\right| \times \cos\left(arg\left(Fp(\text{image 2})\right)\right) + j \times \left|Fp(\text{image 1})\right| \times \sin\left(arg\left(Fp(\text{image 2})\right)\right)$$

When image 1 is the eye and image 2 is the ear then the reconstructed image (via the inverse FT) looks most like the image of the ear, Figure 2.5(c); when it is the other way found, the image is much closer to the eye, Figure 2.5(d). So it is the phase that is controlling the reconstruction in this case, not the magnitude (the bit represented by the magnitude hardly shows in the reconstructions here). Clearly the phase is very important in the representation of a signal.

(a) eye image

(b) ear image

(c) reconstruction from magnitude(eye) and phase(ear)

(d) reconstruction from magnitude(ear) and phase(eye)

Figure 2.23 Illustrating the Importance of Phase

2.7 Transforms other than Fourier

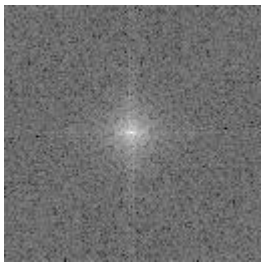
2.7.1 Discrete Cosine Transform

The *Discrete Cosine Transform* (DCT) [Ahmed74] is a real transform that has great advantages in **energy compaction**. Its definition for spectral components $\mathbf{DP}_{u,v}$ is:

$$\mathbf{DP}_{u,v} = \begin{cases} \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y} & \text{if} \quad u=0 \quad \text{and} \quad v=0 \\ \frac{2}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y}\times\cos\left(\frac{(2x+1)u\pi}{2N}\right)\times\cos\left(\frac{(2y+1)v\pi}{2N}\right) & \text{otherwise} \end{cases} \tag{2.36}$$

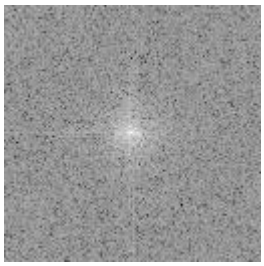
There are many variants of the definition of the DCT and we are concerned only with principles here. The inverse DCT is defined by

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



(a) Fourier transform magnitude

(b) discrete cosine transform



(c) Hartley transform

Figure 2.24 Comparing Transforms

A fast version of the DCT is available, like the FFT, and calculation can be based on the FFT. Both implementations offer about the same speed. The Fourier transform is not actually optimal for *image coding* since the Discrete Cosine transform can give a higher compression rate, for the same image quality. This is because the cosine basis functions can afford for high energy compaction. This can be seen by comparison of Figure 2.24(b) with Figure 2.24(a), which reveals that the DCT components are much more concentrated around the origin, than those for the Fourier Transform. This is the compaction property associated with the DCT. The DCT has

actually been considered as optimal for image coding, and this is why it is found in the JPEG and MPEG standards for coded image transmission.

The DCT is actually shift variant, due to its cosine basis functions. In other respects, its properties are very similar to the DFT, with one important exception: it has not yet proved possible to implement convolution with the DCT. It is actually possible to calculate the DCT via the FFT. This has been performed in Figure 2.24(b).

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

2.7.2 Discrete Hartley Transform

The Hartley transform [Hartley42] is a form of the Fourier transform, but without complex arithmetic, with result for the face image shown in Figure 2.24(c). Oddly, though it sounds like a very rational development, the Hartley transform was first invented in 1942, but not rediscovered and then formulated in discrete form until 1983 [Bracewell83]. One advantage of the Hartley transform is that the forward and inverse transform is the same operation; a disadvantage is that phase is built into the order of frequency components since it is not readily available as the argument of a complex number. The definition of the Discrete Hartley Transform (DHT) is that transform components $\mathbf{HP}_{u,v}$ are:

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

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

$$\mathbf{P}_{x,y} = \frac{1}{N^2}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\mathbf{HP}_{u,v} \times \left(\cos\left(\frac{2\pi}{N} \times (ux+vy) \right) + \sin\left(\frac{2\pi}{N} \times (ux+vy) \right) \right) \tag{2.39}$$

The implementation is then the same for both the forward and the inverse transforms. Again, a fast implementation is available, the Fast Hartley Transform [Bracewell84] (though some suggest that it should be called the Bracewell transform, eponymously). It is actually possible to calculate the DFT of a function, $F(u)$, from its Hartley transform, $H(u)$. The analysis here is based on 1-dimensional data, but only for simplicity since the argument extends readily to two dimensions. By splitting the Hartley transform into its odd and even parts, $O(u)$ and $E(u)$, respectively we obtain:

$$H(u) = O(u) + E(u) \tag{2.40}$$

where:

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

and

$$O(u) = \frac{H(u) - H(N-u)}{2} \tag{2.42}$$

The DFT can then be calculated from the DHT simply by

$$F(u) = E(u) - j \times O(u) \tag{2.43}$$

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

$$H(u) = \mathrm{Re}[F(u)] - \mathrm{Im}[F(u)] \tag{2.44}$$

where Re[ ] and Im[ ] denote the real and the imaginary parts, respectively. This emphasises the natural relationship between the Fourier and the Hartley transform. The image of Figure 2.24(c) has been calculated via the 2D FFT using Equation 2.44. Note that the transform in Figure 2.24(c) is the complete transform whereas the Fourier transform in Figure 2.24(a) shows magnitude only. Naturally, as with the DCT, the properties of the Hartley transform mirror those of the Fourier transform. Unfortunately, the Hartley transform does not have shift invariance but there are ways to handle this. Also, convolution requires manipulation of the odd and even parts.

2.7.3 Introductory Wavelets

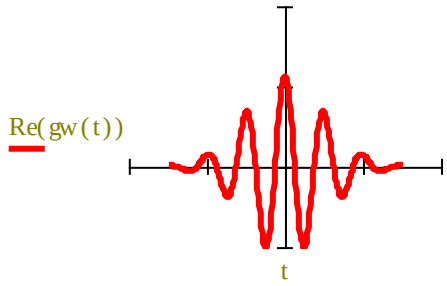
2.7.3.1 Gabor Wavelet

Wavelets are a comparatively recent approach to signal processing, being introduced only in the last decade [Daubechies90]. Their main advantage is that they allow multi-resolution analysis (analysis at different scales, or resolution). Furthermore, wavelets allow decimation in space and frequency, **simultaneously**. Earlier transforms actually allow decimation in frequency, in the forward transform, and in time (or position) in the inverse. In this way, the Fourier transform gives a measure of the frequency content of the whole image: the contribution of the image to a particular frequency component. Simultaneous decimation allows us to describe an image in terms of frequency which occurs at a position, as opposed to an ability to measure frequency content across the whole image. Clearly this gives us a greater descriptional power, which can be used to good effect.

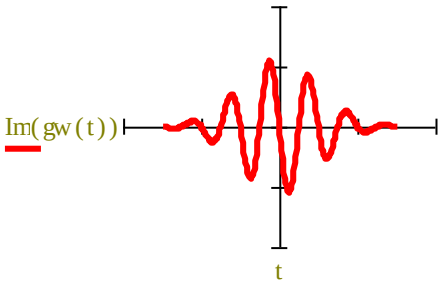
First though, we need a basis function, so that we can decompose a signal. The basis functions in the Fourier transform are sinusoidal waveforms at different frequencies. The function of the Fourier transform is to convolve these sinusoids with a signal to determine how much of each is present. The *Gabor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

$$gw(t, \omega_0, t_0, a) = e^{-j\omega_0 t} e^{-\left(\frac{t-t_0}{a}\right)^2} \tag{2.45}$$

where $\omega_0 = 2\pi f_0$ is the modulating frequency, t_0 dictates position and a controls the width of the Gaussian envelope which embraces the oscillating signal. An example Gabor wavelet is shown in Figure 2.25 which shows the real and the imaginary parts (the modulus is the Gaussian envelope). Increasing the value of ω_0 increases the frequency content within the envelope whereas increasing the value of a spreads the envelope without affecting the frequency. So why does this allow simultaneous analysis of time and frequency? Given that this function is the one convolved with the test data, then we can compare it with the Fourier transform. In fact, if we remove the term on the right hand side of Equation 2.45, we return to the sinusoidal basis function of the Fourier transform, the exponential in Equation 2.1. Accordingly, we can return to the Fourier transform by setting a to be very large. Alternatively, setting f_0 to zero removes frequency information. Since we operate in between these extremes, we obtain position and frequency information simultaneously.



(a) real part



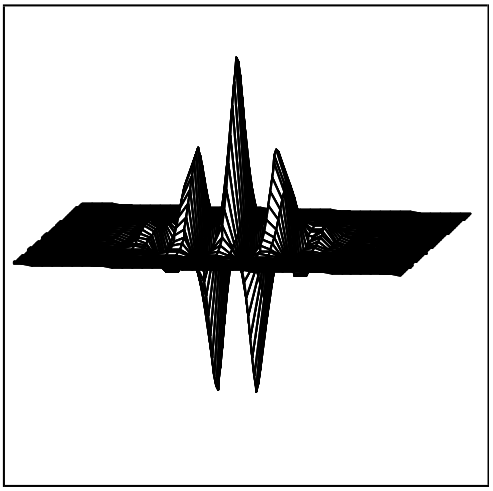
(b) imaginary part

Figure 2.25 An Example Gabor Wavelet

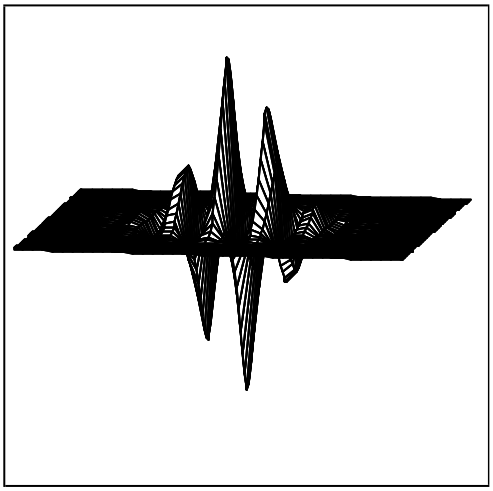
Actually, an infinite class of wavelets exists which can be used as an expansion basis in signal decimation. One approach [Daugman88] has generalised the Gabor function to a 2D form aimed to be optimal in terms of spatial and spectral resolution. These 2D Gabor wavelets are given by

$$gw2D(x,y) = \frac{1}{\sigma\sqrt{\pi}} e^{-\left(\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}\right)} e^{-j2\pi f_0\left((x-x_0)\cos(\theta)+(y-y_0)\sin(\theta)\right)} \tag{2.46}$$

where x_0 and y_0 control position, f_0 controls the frequency of modulation along either axis, and θ controls the **direction** (orientation) of the wavelet (as implicit in a two dimensional system). Naturally, the shape of the area imposed by the 2D Gaussian function could be elliptical if different variances were allowed along the x and y axes (the frequency can also be modulated differently along each axis). Figure 2.26, of an example 2D Gabor wavelet, shows that the real and imaginary parts are even and odd functions, respectively; again, different values for f_0 and σ control the frequency and envelope's spread respectively, the extra parameter θ controls rotation.



(a) real part



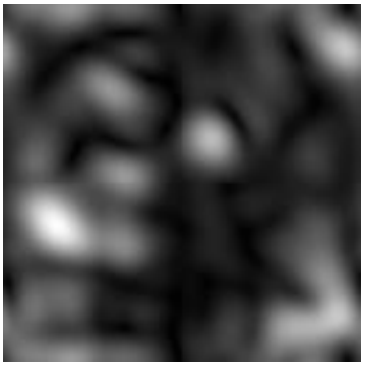
(b) imaginary part

Figure 2.26 Example 2-Dimensional Gabor Wavelet

The function of the wavelet transform is to determine where and how each wavelet specified by the range of values for each of the free parameters occurs in the image. Clearly, there is a wide

choice which depends on application. An example transform is given in Figure 2.27. Here, the Gabor wavelet parameters have been chosen in such a way as to select face features: the eyes, nose and mouth have come out very well. These features are where there is local frequency content with orientation according to the head's inclination. Naturally, these are not the only features with these properties, the cuff of the sleeve is highlighted too! But this does show the Gabor wavelet's ability to select and analyse localised variation in image intensity.

(a) original image



(b) after Gabor wavelet transform

Figure 2.27 An Example Gabor Wavelet Transform

However, the conditions under which a set of continuous Gabor wavelets will provide a complete representation of any image (i.e. that any image can be reconstructed) have only recently been developed. However, the theory is naturally very powerful, since it accommodates frequency and position simultaneously, and further it facilitates multi-resolution analysis - the analysis is then sensitive to scale which is advantageous since objects which are **far** from the camera appear smaller than those which are **close**. We shall find wavelets again, when processing images to find low-level features. Amongst applications of Gabor wavelets, we can find measurement of iris texture to give a very powerful security system [Daugman93] and face feature extraction for automatic face recognition [Lades93]. Wavelets continue to develop [Daubechies90] and have found applications in image texture analysis [Laine93], in coding [daSilva96] and in image restoration [Banham96]. Unfortunately, the discrete wavelet transform is not shift invariant, though there are approaches aimed to remedy this (see for example [Donoho95]). As such, we shall not study it further and just note that there is an important class of transforms that combine spatial and spectral sensitivity, and it is likely that this importance will continue to grow.

2.7.3.2 Haar Wavelet

Though Fourier laid the basis for frequency decomposition, the original wavelet approach is now attributed to Alfred Haar's work in 1909. This uses a binary approach, rather than a continuous signal, and has led to fast methods for finding features in images [Oren97] (especially the object detection part of the Viola-Jones face detection approach [Viola01]). Essentially, the binary functions can be considered to form averages over sets of points, thereby giving means for **compression** and for **feature detection**. If we are to form a new vector (at level h+1) by taking averages of pairs of elements (and retaining the integer representation) of the N points in the previous vector (at level h of the $\log_2(N)$ levels) as,

$$\mathbf{p}_i^{h+1} = \frac{\mathbf{p}_{2\times i}^h + \mathbf{p}_{2\times i+1}^h}{2} \quad i \in 0...\frac{N}{2}-1; h \in 1,...\log_2(N) \tag{2.47}$$

By way of example, consider a vector of points at level 0 as

$$\mathbf{p}^0 = \begin{bmatrix} 1 & 3 & 21 & 19 & 17 & 19 & 1 & -1 \end{bmatrix} \tag{2.48}$$

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

$$\mathbf{p}^1 = \begin{bmatrix} 2 & 20 & 18 & 0 \end{bmatrix} \tag{2.49}$$

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

$$\mathbf{d}^1 = \begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix} \tag{2.50}$$

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

$$\begin{bmatrix} 2+(-1) & 2-(-1) & 20+1 & 20-1 & 18+(-1) & 18-(-1) & 0+1 & 0-1 \end{bmatrix}$$

which returns us to the original vector $\mathbf{p}^0$ (Eqn. 2.48). If we continue to similarly form a series of decompositions (averages of adjacent points), together with the detail at each point, we generate

$$\mathbf{p}^2 = \begin{bmatrix} 11 & 9 \end{bmatrix}; \quad \mathbf{d}^2 = \begin{bmatrix} -9 & 9 \end{bmatrix} \tag{2.51}$$

$$\mathbf{p}^3 = \begin{bmatrix} 10 \end{bmatrix}; \quad \mathbf{d}^3 = \begin{bmatrix} 1 \end{bmatrix} \tag{2.52}$$

We can then store the image as a code

$$\begin{bmatrix} \mathbf{p}^3 & \mathbf{d}^3 & \mathbf{d}^2 & \mathbf{d}^1 \end{bmatrix} = \begin{bmatrix} 10 & 1 & -9 & 9 & -1 & 1 & -1 & 1 \end{bmatrix} \tag{2.53}$$

The process is illustrated in Fig. 2.28 for a sinewave. Fig. 2.28(a) shows the original sinewave, (b) shows the decomposition to level 3, and it is a close but discrete representation whereas (c) shows the decomposition to level 6, which is very coarse. The original signal can be reconstructed from the final code, and this is without error. If the signal is reconstructed by filtering the detail to reduce the amount of stored data, the reconstruction of the original signal (d) at level 0 is quite close to the original signal, and the reconstruction at the other levels is similarly close as expected. The reconstruction error is also shown in (d) to (f). Components of the detail (of magnitude less than one) were removed, achieving a compression ratio of approximately 50%. Naturally, a Fourier transform would encode the signal better, as the Fourier transform is best suited to representing a sinewave. Like Fourier this discrete approach can encode the signal, we can also reconstruct the original signal (reverse the process), and shows how the signal can be represented at different scales, since there are less points in the higher levels.

Binary decomposition	Reconstruction after filtering/ compression

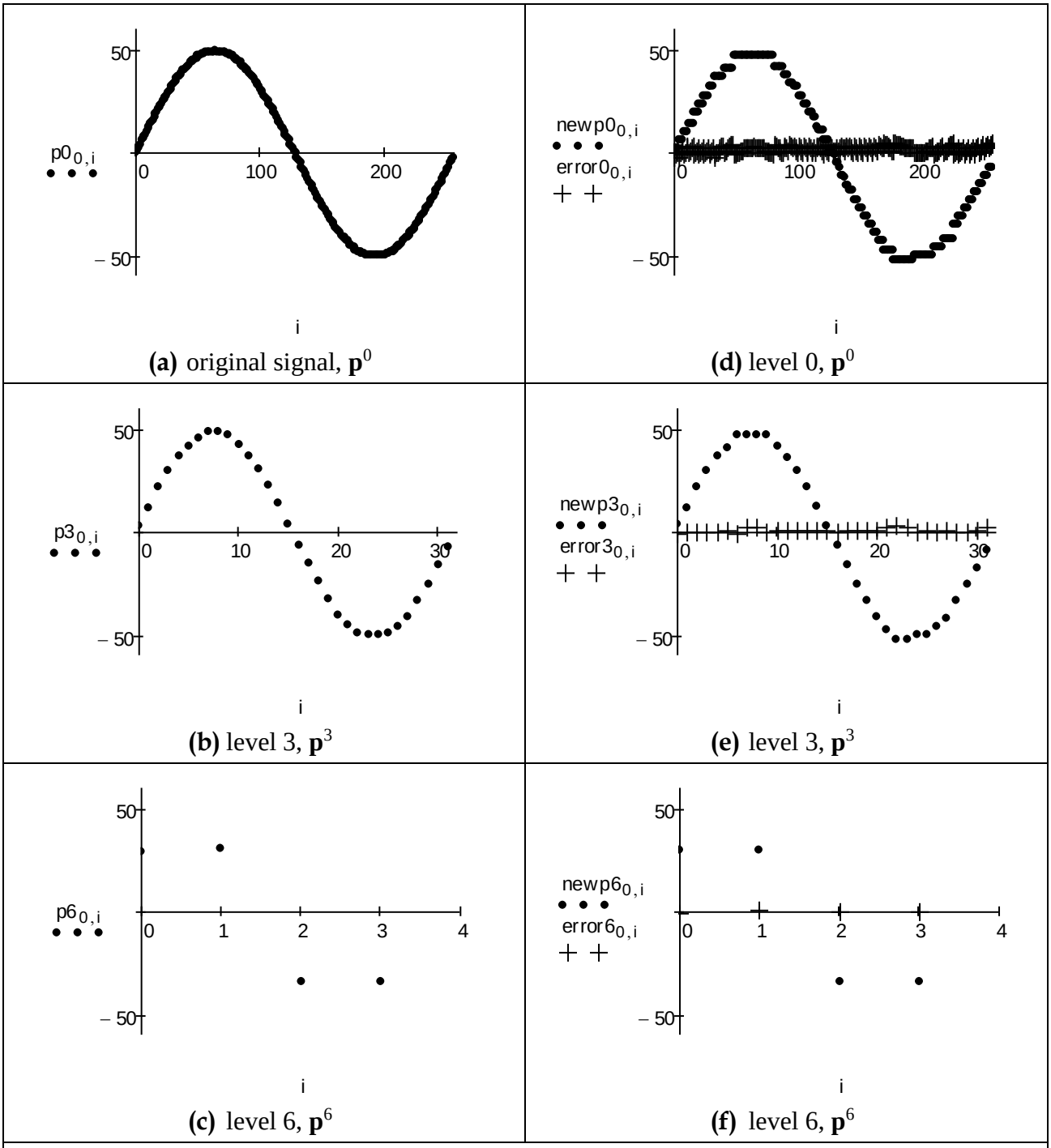


Figure 2.28 Binary Signal Decomposition and Reconstruction

By Eqn. 2.53 this gives a set of numbers of the same size as the original data, and is an alternative representation from which we can reconstruct the original data. There are two important differences:

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

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

A process of successive averaging and differencing can expressed as a function of the form in Fig. 2. 29. This is a mother wavelet which can be applied at different scales, but retains the same shape at those scales. So we now have a binary decomposition rather than the sinewaves of the Fourier transform.

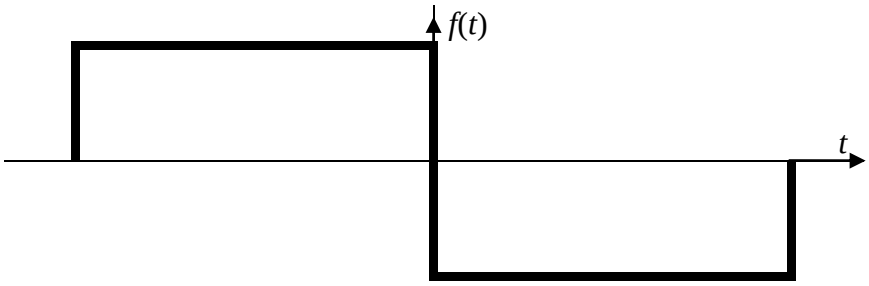
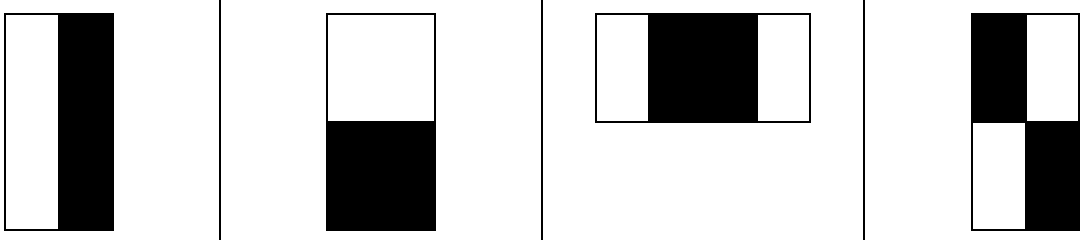


Figure 2.29 Example Haar Wavelet Function

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

$$\mathbf{sp}(x,y) = \sum_{x'<x,y'<y} \mathbf{p}(x',y') \tag{2.54}$$

Then we can achieve wavelet type features which are derived by using these summations. Four of these wavelets are shown in Fig. 2.30. These are placed at selected positions in the image to which they are applied. There are white and black areas: the sum of the pixels under the white area(s) is subtracted from of the sum of the pixels under the dark area(s), in a way similar to the earlier averaging operation in Eqn. 2.48. The first template, Fig. 2.30(a), will detect shapes which are brighter on one side than the other; the second (b) will detect shapes which are brighter in a vertical sense; the third will detect a dark object which has brighter areas on either side. There is a family of these arrangements, and that can apply at selected levels of scale. By collecting the analysis, we can determine objects whatever their position, size (objects further away will appear smaller) or rotation. We will dwell on these topics later and how we find and classify shapes. The point here is that we can achieve some form of binary decomposition in two dimensions, as opposed to the sine/ cosine decomposition of the Gabor wavelet whilst retaining selectivity to scale and position (similar to the Gabor wavelet). This is also simpler, so the binary functions can be processed more quickly.



(a) horizontal | **(b)** vertical | **(c)** bar | **(d)** diagonal

Figure 2.30 Example Haar Wavelet Image Functions

2.7.4 Other Transforms

Decomposing a signal into sinusoidal components was actually one of the first approaches to transform calculus, and this is why the Fourier transform is so important. The sinusoidal functions are actually called *basis functions*, the implicit assumption is that the basis functions map well to the signal components. As such, the Haar wavelets are binary basis functions. There is (theoretically) an infinite range of basis functions. Discrete signals can map better into collections of binary components rather than sinusoidal ones. These collections (or sequences) of binary data are called sequency components and form the basis of the *Walsh transform* [Walsh23], which is a global transform when compared with the Haar functions (like Fourier compared with Gabor). This has found wide application in the interpretation of digital signals, though it is less widely used in image processing (one disadvantage is the lack of shift invariance). The Karhunen-Loéve transform [Karhunen47] [Loéve48] (also called the *Hotelling* transform from which it was derived, or more popularly *Principal Components Analysis* - see Chapter 12, Appendix 3) is a way of analysing (statistical) data to reduce it to those data which are **informative**, discarding those which are not.

2.8 Applications using Frequency Domain Properties

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

The implementation of a low-pass filter which retains frequency components within a circle of specified radius is the function `low_filter`, given in Code 2.4. This operator assumes that the radius and centre co-ordinates of the circle are specified prior to its use. Points within the circle remain unaltered, whereas those outside the circle are set to zero, black.

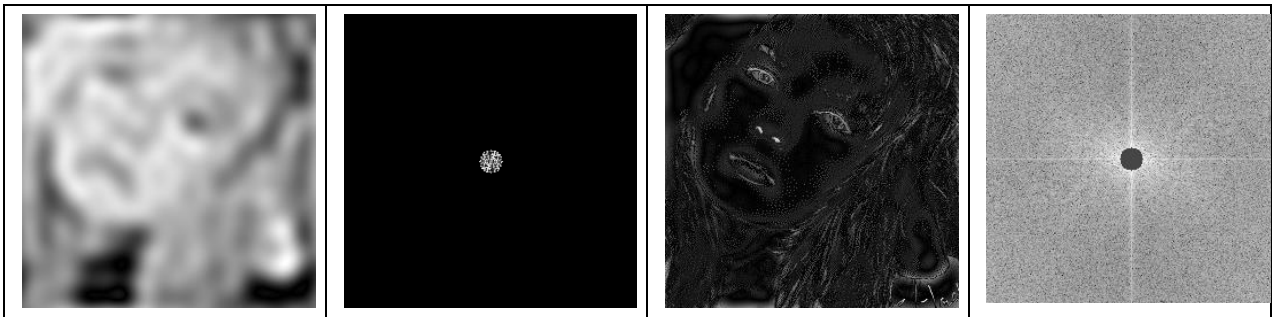
```
function output = low_filter(image,value)
%Retain centred transform components inside circle of radius
%
%  Usage: new image = add(image,number)
%
%  Parameters: image      - array of points
%              value      - radius of filter

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

%filter the transform
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    if (((y-(rows/2))^2)+((x-(cols/2))^2)-(value^2))>0
      output(y,x)=0;
    else
```

```
        output(y,x)=image(y,x);
    end
  end
end
```

Code 2.4 Implementing Low-Pass Filtering

When applied to an image we obtain a low-pass filtered version. In application to an image of a face, the low spatial frequencies are the ones which change slowly as reflected in the resulting, blurred image, Figure 2.31(a). The high frequency components have been removed as shown in the transform, Figure 2.31(b). The radius of the circle controls how much of the original image is retained. In this case, the radius is 10 pixels (and the image resolution is 256×256). If a larger circle were to be used, more of the high frequency detail would be retained (and the image would look more like its original version); if the circle was very small, an even more blurred image would result, since only the lowest spatial frequencies would be retained. This differs from the earlier Gabor wavelet approach which allows for **localised** spatial frequency analysis. Here, the analysis is **global**: we are filtering the frequency across the **whole** image.

Alternatively, we can retain high frequency components and remove low frequency ones. This is a *high-pass filter*. If we remove components near the d.c. component and retain all the others, the result of applying the inverse Fourier transform to the filtered image will be to emphasise the features that were removed in low-pass filtering. This can lead to a popular application of the high-pass filter: to 'crispen' an image by emphasising its high frequency components. An implementation using a circular region merely requires selection of the set of points outside the circle, rather than inside as for the low-pass operator. The effect of high-pass filtering can be observed in Figure **2.31**(c) which shows removal of the low frequency components: this emphasises the hair and the borders of a face's features since these are where brightness varies rapidly. The retained components are those which were removed in low-pass filtering, as illustrated in the transform, Figure **2.31**(d).



(a) low-pass filtered image | **(b)** low-pass filtered transform | **(c)** high-pass filtered image | **(d)** high-pass filtered transform

Figure 2.31 Illustrating Low- and High-Pass Filtering

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

This leads to digital signal processing theory. There are many considerations to be made in the way you select, and the manner in which frequency components are retained or excluded. This is beyond a text on Computer Vision. For further study in this area, Rabiner and Gold [Rabiner75], or Oppenheim and Schafer [Oppenheim09], although published (in their original form) a long

time ago now, remain as popular introductions to Digital Signal Processing theory and applications.

It is actually possible to recognise the object within the low-pass filtered image. Intuitively, this implies that we could just store the frequency components selected from the transform data, rather than all the image points. In this manner a fraction of the information would be stored, and still provide a recognizable image, albeit slightly blurred. This concerns *image coding* which is a popular target for image processing techniques, for further information see [Clarke85] or a newer text, like [Woods11] or [Sayood17]. Note that the JPEG coding approach uses frequency domain decomposition, and is arguably the most uibiquitous image coding technique used today.

2.9 Further Reading

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

Given these advantages, it is well worth looking more deeply. Mark's copy of Fourier's original book has a review "Fourier's treatise is one of the very few scientific books which can never be rendered antiquated by the progress of science" – penned by James Clerk Maxwell no less. For introductory study, there is *Who is Fourier* [Lex12] which offers a lighthearted and completely digestible overview of the Fourier transform, it's simply excellent for a starter view of the topic. For further study (and entertaining study too!) of the Fourier transform, try *The Fourier Transform and its Applications* by R. N. Bracewell [Bracewell86]. A number of the standard image processing texts include much coverage of transform calculus, such as Jain [Jain89], Gonzalez and Wintz [Gonzalez17], and Pratt [Pratt13]. There is a relatively new text concentrating on image processing using Python [Chityala15]. For more coverage of the DCT try Jain [Jain89]; for an excellent coverage of the Walsh transform try Beauchamp's superb text [Beauchamp75]. On compressed sensing, [Eldar12] is worth a read. For wavelets, try the book by Wornell that introduces wavelets from a signal processing standpoint [Wornell96], there's Mallat's classic text [Mallat08] or a new text that includes images [Broughton18]. For general signal processing theory there are introductory texts (see for example Meade and Dillon [Meade86], or Ifeachor's excellent book [Ifeachor02]), for more complete coverage try Rabiner and Gold [Rabiner75] or Oppenheim and Schafer [Oppenheim09] (as mentioned earlier). Finally, on the implementation side of the FFT (and for many other signal processing algorithms) *Numerical Recipes in C* [Press07] is an excellent book. It is extremely readable, full of practical detail – well worth a look. Numerical Recipes is on the web too, together with other signal processing sites, as listed in Table 1.4.

2.10 Chapter 2 References

[Ahmed74] Ahmed, N., Natarajan, T. and Rao, K. R., Discrete Cosine Transform, *IEEE Trans. on Computers*, pp 90-93, 1974

[Banham96] Banham, M. R., and Katsaggelos, K., Spatially Adaptive Wavelet-Based Multiscale Image Restoration, *IEEE Trans. on Image Processing*, **5**(4), pp 619-634 , 1996

[Beauchamp75] Beauchamp, K. G., *Walsh Functions and Their Applications*, Academic Press, London UK, 1975

[Bracewell84] Bracewell, R. N., The Fast Hartley Transform, *Proc. IEEE*, **72**(8), pp 1010-1018, 1984

[Bracewell83] Bracewell, R. N., The Discrete Hartley Transform, *J. Opt. Soc. Am.*, **73**(12), pp 1832-1835, 1984

[Bracewell86] Bracewell, R. N., *The Fourier Transform and its Applications*, Revised 2nd Edition, McGraw-Hill, Book Co., Singapore, 1986

[Broughton18] Broughton, S. A., Bryan, K., *Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing*, Wiley, Chichester UK, 2nd Edition 2018

[Chityala15] Chityala, R., and Pudipeddi, S., *Image Processing and Acquisition using Python*, CRC Press, Boca Raton FL USA, 2015

[Clarke85] Clarke, R. J., *Transform Coding of Images*, Addison Wesley, Reading MA USA, 1985

[daSilva96] da Silva, E. A. B., and Ghanbari, M., On the Performance of Linear Phase Wavelet Transforms in Low Bit-Rate Image Coding, *IEEE Trans. on Image Processing*, **5**(5), pp 689-704, 1996

[Daubechies90] Daubechies, I., The Wavelet Transform, Time Frequency Localisation and Signal Analysis, *IEEE Trans. on Information Theory*, **36**(5), pp 961-1004, 1990

[Daugman88] Daugman, J. G., Complete Discrete 2D Gabor Transforms by Neural Networks for Image Analysis and Compression, *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7), pp 1169-1179, 1988

[Daugman93] Daugman, J. G., High Confidence Visual Recognition of Persons by a Test of Statistical Independence, *IEEE Trans. on PAMI*, **15**(11), pp1148-1161, 1993

[Donoho95] Donoho, D. L., Denoising by Soft Thresholding, *IEEE Trans. on Information Theory*, **41**(3), pp 613-627, 1995

[Donoho06] Donoho, D. L., Compressed Sensing, *IEEE Trans. on Information Theory*, **52**(4), pp 1289–1306, 2006

[Eldar12] Eldar, Y. C., and Kutyniok, G., Editors, *Compressed Sensing: Theory and Applications*, Cambridge University Press, Cambridge UK, 2012

[Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4th Edition, Pearson Education, 2017

[Hartley42] Hartley, R. L. V., A More Symmetrical Fourier Analysis Applied to Transmission Problems, *Proc. IRE*, **144**, pp 144-150, 1942

[Ifeachor02] Ifeachor, E. C., and Jervis, B. W., *Digital Signal Processing*, Prentice Hall, Hemel Hempstead UK, 2nd Edition 2002

[Jain89] Jain A. K., *Fundamentals of Computer Vision*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 1989

[Karhunen47] Karhunen, K., Über Lineare Methoden in der Wahrscheinlich-Keitsrechnung, *Ann. Acad. Sci. Fennicae*, Ser A.I.37, 1947 (Translation in I. Selin, On Linear Methods in Probability Theory, Doc. T-131, The RAND Corp., Santa Monica CA, 1960.)

[Lades93] Lades, M., Vorbruggen, J. C., Buhmann, J., and Lange, J., Madsburg, C. V. D., Wurtz, R. P., and Konen, W., Distortion Invariant Object Recognition in the Dynamic Link Architecture, *IEEE Trans. on Computers*, **42**, pp 300-311, 1993

[Laine93] Laine, A., and Fan, J., Texture Classification by Wavelet Packet Signatures, *IEEE Trans. on PAMI*, **15**, pp 1186-1191, 1993

[Lex12] Lex, T. C. O. L. T. (!!), *Who is Fourier, a Mathematical Adventure*, Language Research Foundation, Boston MA USA, 2nd Edition 2012

[Loéve48] Loéve, M., Fonctions Alétoires de Seconde Ordre, in: P. Levy, Ed., *Processus Stochastiques et Mouvement Brownien*, Hermann, Paris 1948

[Mallat08] Mallat, S., *A Wavelet Tour of Signal Processing*, 3rd Edition, Academic Press, Burlington MA USA, 2008

[Meade86] Meade, M. L. and Dillon, C. R., *Signals and Systems, Models and Behaviour*, Van Nostrand Reinhold (UK) Co. Ltd., Wokingham UK, 1986

[Oppenheim09] Oppenheim, A. V., Schafer, R. W., and Buck, J. R. *Discrete-Time Signal Processing*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 3rd Edition, 2009

[Oren97] Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., and Poggio, T., Pedestrian Detection Using Wavelet Templates, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pp193-199, 1997

[Pratt13] Pratt, W. K., *Introduction to Digital Image Processing*, CRC Press, Boca Raton FL USA, 2013

[Press07] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, 3rd Edition, 2007

[Rabiner75] Rabiner, L. R. and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice Hall Inc., Englewood Cliffs NJ USA, 1975

[Sayood17] Sayood, K., *Introduction to Data Compression*, Morgan Kaufmann, Cambridge MA USA, 5th Edition 2017

[Unser00] Unser, M., Sampling - 50 Years after Shannon, *Proceedings of the IEEE*, **88**(4), pp 569-587, 2000

[Viola01] Viola, P., and Jones, M., Rapid Object Detection using a Boosted Cascade of Simple Features, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)*, **1**, pp.511-519, 2001

[Walsh23] Walsh, J. L., A Closed Set of Normal Orthogonal Functions, *Am. J. Math.*, **45**(1), pp 5-24, 1923

[Woods11] Woods, J. W., *Multidimensional Signal, Image, and Video Processing and Coding*, Academic Press, Wakham MA USA, 2nd Edition 2011

[Wornell96] Wornell, G. W., *Signal Processing with Fractals, a Wavelet-Based Approach*, Prentice Hall Inc., Upper Saddle River NJ USA, 1996