

We have considered a representation of images as that of functions mapping from a “spatial” domain to a “value” range (intensities, color, etc.). Our studies have meandered through operations on images and numerous types of invariance properties that we seek from images. However, we have also come across some examples where the representation of images as function has limited our ability to process an image in a certain way. For example, we sought to implement a magicwand operator that looks for contiguous groups of pixels that have values in a similar range. We could not easily think of a formulation to satisfy that goal.

We need alternative representations of the image. We do not seek to change the underlying image data. Rather, we seek a different interpretation of how the data is defined.

Here, we will consider images as points.

**Definition 1.** Let  $\Omega$  define the set of all images of a certain *size* which means they have the same size domain (e.g., number of rows and columns) and the same range-space (e.g., real-values).

*Remark 1.1.* We can relate this set directly back to images as functions. The domain of the function and the range of the function of all images in a certain set  $\Omega$  are equivalent. It is odd to use the terms *domain* and *range* when discussing the elements of a set, unless they are functions. However, we will see, below, how the discussion moves away from images as functions. It nevertheless helps the discussion to use these terms initially.

*Remark 1.2.* The sets of images we consider in this representation have discrete domains but can have either continuous or discrete ranges.

*Remark 1.3.* When the domain and range are discrete, and specifically, the range is grayscale values from  $\{0, \dots, 255\}$ , we denote the set as  $\mathcal{G}$ . Unless otherwise specified,  $\Omega$  is the set of images of a certain size domain and a range of continuous values from  $\mathbb{R}$ .

**Example 1 — Cartesian Basis** Let us construct a concrete example. Consider the pixel values of an  $m \times n$  image as a rectangular matrix of pixel values; the *naïve* interpretation of the image, in some sense. This is our starting point. Next, consider an  $m \times n$  vector of coefficients  $\alpha_1, \alpha_2, \dots, \alpha_{mn}$ . If we lay out the vector of coefficients such that each coefficient corresponds to one of the pixel values in the image, we have

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \alpha_{mn} \end{bmatrix}. \quad (1)$$

Now, consider a set of images  $\{\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_{mn}\}$  that have zeros everywhere except for one pixel location, such as  $\mathbf{V}_i$  has zeros everywhere and a 1 in the  $i$ th pixel location. For example,

the image  $V_2$  would be defined as

$$V_2 = \begin{bmatrix} 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ \cdots & \cdots & \cdots & 0 \end{bmatrix} . \quad (2)$$

We can then exactly reconstruct the image matrix  $I$  using the set of images  $V_1, V_2, \dots, V_{mn}$  and the coefficient vector  $\alpha_1, \alpha_2, \dots, \alpha_{mn}$ :

$$I = \sum_{i=1}^{mn} V_i \alpha_i . \quad (3)$$

Although much of this costly computation results in multiplication with a zero (basis element), the interpretation is nevertheless valuable, as we will see next.

*Remark 1.4.* For a linear operator  $Q$ , we can change the basis  $\{V\}$  and maintain the same information in our image:

$$I = \sum_{i=1}^{mn} V_i \alpha_i \quad (4)$$

$$Q \circ I = Q \circ \sum_{i=1}^{mn} V_i \alpha_i \quad (5)$$

$$= \sum_{i=1}^{mn} Q \circ V_i \alpha_i \quad (6)$$

where recall, again, the  $\circ$  notation implies the application of the operator to the image.

■ **Example 2 — Rotating the Bases**    *see hand-written notes for this example.*

■ **Example 3 — Fourier Transform**    *see slides notes for this example.*

■ **Example 4 — Haar-Wavelet Transform**    *see hand-written notes for this example.*

**Definition 2.** The set of images, e.g.  $\alpha = \{\alpha_1, \dots, \alpha_{mn}\} \in \Omega$ , vector addition and scalar multiplication, and the field of reals  $\mathcal{F} = \mathbb{R}$ , form a *vector space*,  $\mathcal{V}_\Omega$ .

*Remark 2.1.* This definition satisfies the axioms of a vector space.

1. Closed under vector addition and scalar multiplication:  $\alpha, \beta \in \Omega$  and, for  $a \in \mathbb{R}$ ,

$$a\alpha \in \Omega \quad (7)$$

$$\alpha + \beta \in \Omega . \quad (8)$$

2. Addition commutes: for any  $\alpha, \beta \in \Omega$ ,

$$\alpha + \beta = \beta + \alpha . \quad (9)$$

3. Addition and scalar multiplication are associative: for any  $\alpha, \beta, \gamma \in \Omega$  and  $a, b \in \mathbb{R}$ ,

$$(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma) \quad (10)$$

$$a(b\alpha) = (ab)\alpha \quad (11)$$

4. Identities exist for addition and multiplication. The image of zeros,  $\mathbf{0}$ , and the image of ones,  $\mathbf{1}$ , leads to, for any  $\alpha \in \Omega$ ,  $\alpha + \mathbf{0} = \alpha$  and  $\alpha \cdot \mathbf{1} = \alpha$ .
5. Additive inverse exists. For any  $\alpha \in \Omega$ , there exists an element  $\beta \in \Omega$  such that  $\alpha + \beta = \mathbf{0}$  (Note that this does not hold if we directly work with the digital image representation,  $\mathcal{G}$ , of 8-bit grayscale values, which is why we have assumed the set of reals for the pixel values.)
6. Scalar multiplication is distributive with respect to vector addition and scalar addition: For any  $a, b \in \mathbb{R}$  and any  $\alpha, \beta \in \Omega$ ,

$$a(\alpha + \beta) = a\alpha + a\beta \quad (12)$$

$$(a + b)\alpha = a\alpha + b\alpha . \quad (13)$$

**Example 5 — Euclidean distance between images** When images are points  $\alpha$ , we can naturally begin to relate them to each other directly by computing their distance. Consider two images  $\alpha$  and  $\beta$ , a simple way of computing their distance is with the  $\ell_2$ -norm  $\|\alpha - \beta\|_2$ . Although plausible, this simple approach may not yield a good metric on images: although, we can interpret the space of images as a vector space (next), it is not clear that this is the best representation of the *space of natural images* or space of images from a particular application area with which we work. The specific shape of the sub-space of images we are consider may occupy a rather small fraction of this overall space.

that avoid the creation of a potentially large re-coloring (equivalence) table. Well-debugged connected component algorithms are also available in most image processing libraries.

Once a binary or multi-valued image has been segmented into its connected components, it is often useful to compute the area statistics for each individual region  $\mathcal{R}$ . Such statistics include:

- the area (number of pixels);
- the perimeter (number of boundary pixels);
- the centroid (average  $x$  and  $y$  values);
- the second moments,

$$\mathbf{M} = \sum_{(x,y) \in \mathcal{R}} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix}, \quad (3.46)$$

from which the major and minor axis orientation and lengths can be computed using eigenvalue analysis.<sup>7</sup>

These statistics can then be used for further processing, e.g., for sorting the regions by the area size (to consider the largest regions first) or for preliminary matching of regions in different images.

### 3.4 Fourier transforms

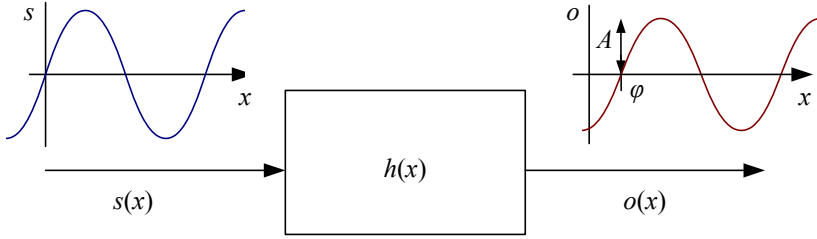
In Section 3.2, we mentioned that Fourier analysis could be used to analyze the frequency characteristics of various filters. In this section, we explain both how Fourier analysis lets us determine these characteristics (or equivalently, the frequency *content* of an image) and how using the Fast Fourier Transform (FFT) lets us perform large-kernel convolutions in time that is independent of the kernel's size. More comprehensive introductions to Fourier transforms are provided by Bracewell (1986); Glassner (1995); Oppenheim and Schaffer (1996); Oppenheim, Schaffer, and Buck (1999).

How can we analyze what a given filter does to high, medium, and low frequencies? The answer is to simply pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated. Let

$$s(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i) \quad (3.47)$$

---

<sup>7</sup> Moments can also be computed using Green's theorem applied to the boundary pixels (Yang and Albrechtsen 1996).



**Figure 3.24** The Fourier Transform as the response of a filter  $h(x)$  to an input sinusoid  $s(x) = e^{j\omega x}$  yielding an output sinusoid  $o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$ .

be the input sinusoid whose *frequency* is  $f$ , *angular frequency* is  $\omega = 2\pi f$ , and *phase* is  $\phi_i$ . Note that in this section, we use the variables  $x$  and  $y$  to denote the spatial coordinates of an image, rather than  $i$  and  $j$  as in the previous sections. This is both because the letters  $i$  and  $j$  are used for the *imaginary* number (the usage depends on whether you are reading complex variables or electrical engineering literature) and because it is clearer how to distinguish the horizontal ( $x$ ) and vertical ( $y$ ) components in frequency space. In this section, we use the letter  $j$  for the imaginary number, since that is the form more commonly found in the signal processing literature (Bracewell 1986; Oppenheim and Schaffer 1996; Oppenheim, Schaffer, and Buck 1999).

If we convolve the sinusoidal signal  $s(x)$  with a filter whose impulse response is  $h(x)$ , we get another sinusoid of the same frequency but different magnitude  $A$  and phase  $\phi_o$ ,

$$o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o), \quad (3.48)$$

as shown in Figure 3.24. To see that this is the case, remember that a convolution can be expressed as a weighted summation of shifted input signals (3.14) and that the summation of a bunch of shifted sinusoids of the same frequency is just a single sinusoid at that frequency.<sup>8</sup> The new magnitude  $A$  is called the *gain* or *magnitude* of the filter, while the phase difference  $\Delta\phi = \phi_o - \phi_i$  is called the *shift* or *phase*.

In fact, a more compact notation is to use the complex-valued sinusoid

$$s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x. \quad (3.49)$$

In that case, we can simply write,

$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}. \quad (3.50)$$

<sup>8</sup> If  $h$  is a general (non-linear) transform, additional *harmonic* frequencies are introduced. This was traditionally the bane of audiophiles, who insisted on equipment with no *harmonic distortion*. Now that digital audio has introduced pure distortion-free sound, some audiophiles are buying retro tube amplifiers or digital signal processors that simulate such distortions because of their “warmer sound”.

The *Fourier transform* is simply a tabulation of the magnitude and phase response at each frequency,

$$H(\omega) = \mathcal{F}\{h(x)\} = Ae^{j\phi}, \quad (3.51)$$

i.e., it is the response to a complex sinusoid of frequency  $\omega$  passed through the filter  $h(x)$ . The Fourier transform pair is also often written as

$$h(x) \xleftrightarrow{\mathcal{F}} H(\omega). \quad (3.52)$$

Unfortunately, (3.51) does not give an actual *formula* for computing the Fourier transform. Instead, it gives a *recipe*, i.e., convolve the filter with a sinusoid, observe the magnitude and phase shift, repeat. Fortunately, closed form equations for the Fourier transform exist both in the continuous domain,

$$H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx, \quad (3.53)$$

and in the discrete domain,

$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi kx}{N}}, \quad (3.54)$$

where  $N$  is the length of the signal or region of analysis. These formulas apply both to filters, such as  $h(x)$ , and to signals or images, such as  $s(x)$  or  $g(x)$ .

The discrete form of the Fourier transform (3.54) is known as the *Discrete Fourier Transform* (DFT). Note that while (3.54) can be evaluated for any value of  $k$ , it only makes sense for values in the range  $k \in [-\frac{N}{2}, \frac{N}{2}]$ . This is because larger values of  $k$  *alias* with lower frequencies and hence provide no additional information, as explained in the discussion on aliasing in Section 2.3.1.

At face value, the DFT takes  $O(N^2)$  operations (multiply-adds) to evaluate. Fortunately, there exists a faster algorithm called the *Fast Fourier Transform* (FFT), which requires only  $O(N \log_2 N)$  operations (Bracewell 1986; Oppenheim, Schafer, and Buck 1999). We do not explain the details of the algorithm here, except to say that it involves a series of  $\log_2 N$  stages, where each stage performs small  $2 \times 2$  transforms (matrix multiplications with known coefficients) followed by some semi-global permutations. (You will often see the term *butterfly* applied to these stages because of the pictorial shape of the signal processing graphs involved.) Implementations for the FFT can be found in most numerical and signal processing libraries.

Now that we have defined the Fourier transform, what are some of its properties and how can they be used? Table 3.1 lists a number of useful properties, which we describe in a little more detail below:

Property	Signal	Transform
superposition	$f_1(x) + f_2(x)$	$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$	$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$	$F^*(\omega)$
convolution	$f(x) * h(x)$	$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$	$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$	$F(\omega) * H(\omega)$
differentiation	$f'(x)$	$j\omega F(\omega)$
domain scaling	$f(ax)$	$1/a F(\omega/a)$
real images	$f(x) = f^*(x)$	$\Leftrightarrow F(\omega) = F^*(-\omega)$
Parseval's Theorem	$\sum_x [f(x)]^2$	$= \sum_\omega [F(\omega)]^2$

**Table 3.1** Some useful properties of Fourier transforms. The original transform pair is  $F(\omega) = \mathcal{F}\{f(x)\}$ .

- **Superposition:** The Fourier transform of a sum of signals is the sum of their Fourier transforms. Thus, the Fourier transform is a linear operator.
- **Shift:** The Fourier transform of a shifted signal is the transform of the original signal multiplied by a *linear phase shift* (complex sinusoid).
- **Reversal:** The Fourier transform of a reversed signal is the complex conjugate of the signal's transform.
- **Convolution:** The Fourier transform of a pair of convolved signals is the product of their transforms.
- **Correlation:** The Fourier transform of a correlation is the product of the first transform times the complex conjugate of the second one.
- **Multiplication:** The Fourier transform of the product of two signals is the convolution of their transforms.
- **Differentiation:** The Fourier transform of the derivative of a signal is that signal's transform multiplied by the frequency. In other words, differentiation linearly emphasizes (magnifies) higher frequencies.
- **Domain scaling:** The Fourier transform of a stretched signal is the equivalently compressed (and scaled) version of the original transform and *vice versa*.

- **Real images:** The Fourier transform of a real-valued signal is symmetric around the origin. This fact can be used to save space and to double the speed of image FFTs by packing alternating scanlines into the real and imaginary parts of the signal being transformed.
- **Parseval's Theorem:** The energy (sum of squared values) of a signal is the same as the energy of its Fourier transform.

All of these properties are relatively straightforward to prove (see Exercise 3.15) and they will come in handy later in the book, e.g., when designing optimum Wiener filters (Section 3.4.3) or performing fast image correlations (Section 8.1.2).

### 3.4.1 Fourier transform pairs

Now that we have these properties in place, let us look at the Fourier transform pairs of some commonly occurring filters and signals, as listed in Table 3.2. In more detail, these pairs are as follows:

- **Impulse:** The impulse response has a constant (all frequency) transform.
- **Shifted impulse:** The shifted impulse has unit magnitude and linear phase.
- **Box filter:** The box (moving average) filter

$$\text{box}(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{else} \end{cases} \quad (3.55)$$

has a sinc Fourier transform,

$$\text{sinc}(\omega) = \frac{\sin \omega}{\omega}, \quad (3.56)$$

which has an infinite number of side lobes. Conversely, the sinc filter is an ideal low-pass filter. For a non-unit box, the width of the box  $a$  and the spacing of the zero crossings in the sinc  $1/a$  are inversely proportional.

- **Tent:** The piecewise linear tent function,

$$\text{tent}(x) = \max(0, 1 - |x|), \quad (3.57)$$








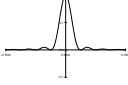

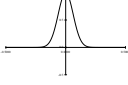
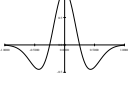
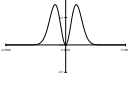


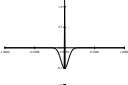
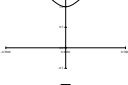
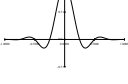
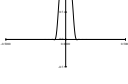
has a  $\text{sinc}^2$  Fourier transform.

- **Gaussian:** The (unit area) Gaussian of width  $\sigma$ ,

$$G(x; \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \quad (3.58)$$

has a (unit height) Gaussian of width  $\sigma^{-1}$  as its Fourier transform.



Name	Signal	Transform
impulse	 $\delta(x)$	$\Leftrightarrow 1$ 
shifted impulse	 $\delta(x - u)$	$\Leftrightarrow e^{-j\omega u}$ 
box filter	 $\text{box}(x/a)$	$\Leftrightarrow \text{asinc}(a\omega)$ 
tent	 $\text{tent}(x/a)$	$\Leftrightarrow \text{asinc}^2(a\omega)$ 
Gaussian	 $G(x; \sigma)$	$\Leftrightarrow \frac{\sqrt{2\pi}}{\sigma} G(\omega; \sigma^{-1})$ 
Laplacian of Gaussian	 $(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})G(x; \sigma)$	$\Leftrightarrow -\frac{\sqrt{2\pi}}{\sigma} \omega^2 G(\omega; \sigma^{-1})$ 
Gabor	 $\cos(\omega_0 x)G(x; \sigma)$	$\Leftrightarrow \frac{\sqrt{2\pi}}{\sigma} G(\omega \pm \omega_0; \sigma^{-1})$ 
unsharp mask	 $(1 + \gamma)\delta(x) - \gamma G(x; \sigma)$	$\Leftrightarrow \frac{(1 + \gamma) - \frac{\sqrt{2\pi}\gamma}{\sigma} G(\omega; \sigma^{-1})}{\sigma}$ 
windowed sinc	 $\text{rcos}(x/(aW)) \text{sinc}(x/a)$	$\Leftrightarrow$ (see Figure 3.29) 

**Table 3.2** Some useful (continuous) Fourier transform pairs: The dashed line in the Fourier transform of the shifted impulse indicates its (linear) phase. All other transforms have zero phase (they are real-valued). Note that the figures are not necessarily drawn to scale but are drawn to illustrate the general shape and characteristics of the filter or its response. In particular, the Laplacian of Gaussian is drawn inverted because it resembles more a “Mexican hat”, as it is sometimes called.

- **Laplacian of Gaussian:** The second derivative of a Gaussian of width  $\sigma$ ,

$$LoG(x; \sigma) = \left( \frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) G(x; \sigma) \quad (3.59)$$

has a band-pass response of

$$-\frac{\sqrt{2\pi}}{\sigma} \omega^2 G(\omega; \sigma^{-1}) \quad (3.60)$$

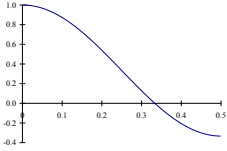
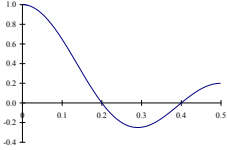
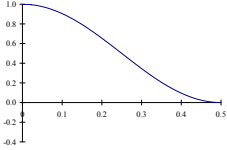
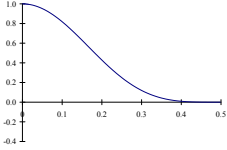
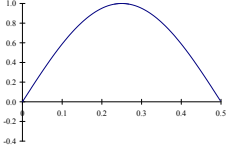
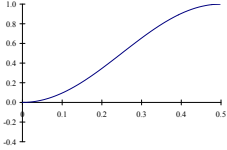
as its Fourier transform.

- **Gabor:** The even Gabor function, which is the product of a cosine of frequency  $\omega_0$  and a Gaussian of width  $\sigma$ , has as its transform the sum of the two Gaussians of width  $\sigma^{-1}$  centered at  $\omega = \pm\omega_0$ . The odd Gabor function, which uses a sine, is the difference of two such Gaussians. Gabor functions are often used for oriented and band-pass filtering, since they can be more frequency selective than Gaussian derivatives.
- **Unsharp mask:** The unsharp mask introduced in (3.22) has as its transform a unit response with a slight boost at higher frequencies.
- **Windowed sinc:** The windowed (masked) sinc function shown in Table 3.2 has a response function that approximates an ideal low-pass filter better and better as additional side lobes are added ( $W$  is increased). Figure 3.29 shows the shapes of these such filters along with their Fourier transforms. For these examples, we use a one-lobe raised cosine,

$$\text{rcos}(x) = \frac{1}{2}(1 + \cos \pi x) \text{box}(x), \quad (3.61)$$

also known as the *Hann window*, as the windowing function. Wolberg (1990) and Oppenheim, Schafer, and Buck (1999) discuss additional windowing functions, which include the *Lanczos* window, the positive first lobe of a sinc function.

We can also compute the Fourier transforms for the small discrete kernels shown in Figure 3.14 (see Table 3.3). Notice how the moving average filters do not uniformly dampen higher frequencies and hence can lead to ringing artifacts. The binomial filter (Gomes and Velho 1997) used as the “Gaussian” in Burt and Adelson’s (1983a) Laplacian pyramid (see Section 3.5), does a decent job of separating the high and low frequencies, but still leaves a fair amount of high-frequency detail, which can lead to aliasing after downsampling. The Sobel edge detector at first linearly accentuates frequencies, but then decays at higher frequencies, and hence has trouble detecting fine-scale edges, e.g., adjacent black and white columns. We look at additional examples of small kernel Fourier transforms in Section 3.5.2, where we study better kernels for pre-filtering before decimation (size reduction).

Name	Kernel	Transform	Plot
box-3	$\frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$	$\frac{1}{3}(1 + 2 \cos \omega)$	
box-5	$\frac{1}{5} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$	$\frac{1}{5}(1 + 2 \cos \omega + 2 \cos 2\omega)$	
linear	$\frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$	$\frac{1}{2}(1 + \cos \omega)$	
binomial	$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	$\frac{1}{4}(1 + \cos \omega)^2$	
Sobel	$\frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$	$\sin \omega$	
corner	$\frac{1}{2} \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$	$\frac{1}{2}(1 - \cos \omega)$	

**Table 3.3** Fourier transforms of the separable kernels shown in Figure 3.14.

### 3.4.2 Two-dimensional Fourier transforms

The formulas and insights we have developed for one-dimensional signals and their transforms translate directly to two-dimensional images. Here, instead of just specifying a horizontal or vertical frequency  $\omega_x$  or  $\omega_y$ , we can create an oriented sinusoid of frequency  $(\omega_x, \omega_y)$ ,

$$s(x, y) = \sin(\omega_x x + \omega_y y). \quad (3.62)$$

The corresponding two-dimensional Fourier transforms are then

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} dx dy, \quad (3.63)$$

and in the discrete domain,

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi \frac{k_x x + k_y y}{MN}}, \quad (3.64)$$

where  $M$  and  $N$  are the width and height of the image.

All of the Fourier transform properties from Table 3.1 carry over to two dimensions if we replace the scalar variables  $x$ ,  $\omega$ ,  $x_0$  and  $a$  with their 2D vector counterparts  $\mathbf{x} = (x, y)$ ,  $\boldsymbol{\omega} = (\omega_x, \omega_y)$ ,  $\mathbf{x}_0 = (x_0, y_0)$ , and  $\mathbf{a} = (a_x, a_y)$ , and use vector inner products instead of multiplications.

### 3.4.3 Wiener filtering

While the Fourier transform is a useful tool for analyzing the frequency characteristics of a filter kernel or image, it can also be used to analyze the frequency spectrum of a whole *class* of images.

A simple model for images is to assume that they are random noise fields whose expected magnitude at each frequency is given by this *power spectrum*  $P_s(\omega_x, \omega_y)$ , i.e.,

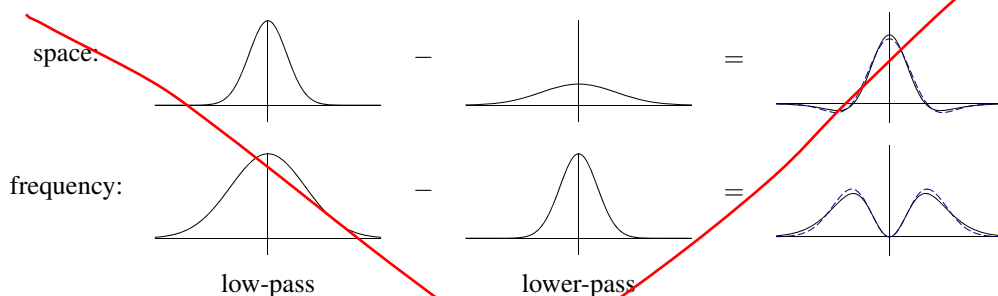
$$\langle [S(\omega_x, \omega_y)]^2 \rangle = P_s(\omega_x, \omega_y), \quad (3.65)$$

where the angle brackets  $\langle \cdot \rangle$  denote the expected (mean) value of a random variable.<sup>9</sup> To generate such an image, we simply create a random Gaussian noise image  $S(\omega_x, \omega_y)$  where each “pixel” is a zero-mean Gaussian<sup>10</sup> of variance  $P_s(\omega_x, \omega_y)$  and then take its inverse FFT.

The observation that signal spectra capture a first-order description of spatial statistics is widely used in signal and image processing. In particular, assuming that an image is a

<sup>9</sup> The notation  $E[\cdot]$  is also commonly used.

<sup>10</sup> We set the DC (i.e., constant) component at  $S(0, 0)$  to the mean grey level. See Algorithm C.1 in Appendix C.2 for code to generate Gaussian noise.



**Figure 3.35** The difference of two low-pass filters results in a band-pass filter. The dashed blue lines show the close fit to a half-octave Laplacian of Gaussian.

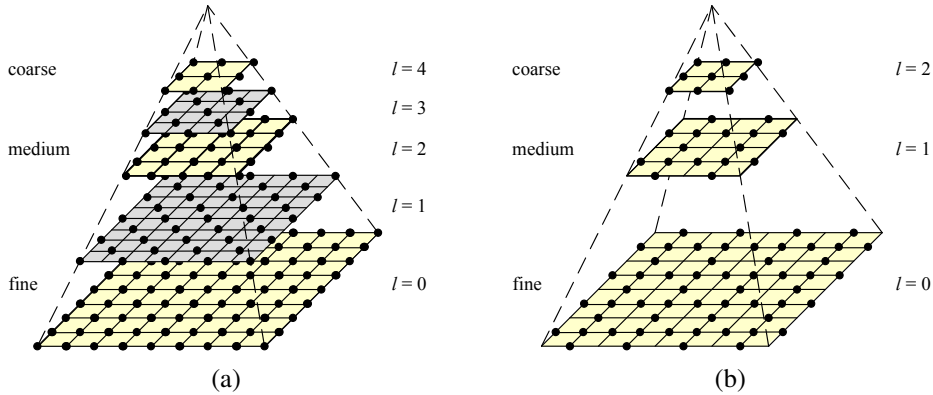
cent levels, the authors claim that coarse-to-fine algorithms perform better. In the image-processing community, half-octave pyramids combined with checkerboard sampling grids are known as *quincunx* sampling (Feilner, Van De Ville, and Unser 2005). In detecting multi-scale features (Section 4.1.1), it is often common to use half-octave or even quarter-octave pyramids (Lowe 2004; Triggs 2004). However, in this case, the subsampling only occurs at every octave level, i.e., the image is repeatedly blurred with wider Gaussians until a full octave of resolution change has been achieved (Figure 4.11).

### 3.5.4 Wavelets

While pyramids are used extensively in computer vision applications, some people use *wavelet* decompositions as an alternative. Wavelets are filters that localize a signal in both space and frequency (like the Gabor filter in Table 3.2) and are defined over a hierarchy of scales. Wavelets provide a smooth way to decompose a signal into frequency components without blocking and are closely related to pyramids.

Wavelets were originally developed in the applied math and signal processing communities and were introduced to the computer vision community by Mallat (1989). Strang (1989); Simoncelli and Adelson (1990b); Rioul and Vetterli (1991); Chui (1992); Meyer (1993) all provide nice introductions to the subject along with historical reviews, while Chui (1992) provides a more comprehensive review and survey of applications. Sweldens (1997) describes the more recent *lifting* approach to wavelets that we discuss shortly.

Wavelets are widely used in the computer graphics community to perform multi-resolution geometric processing (Stollnitz, DeRose, and Salesin 1996) and have also been used in computer vision for similar applications (Szeliski 1990b; Pentland 1994; Gortler and Cohen 1995; Yaou and Chang 1994; Lai and Vemuri 1997; Szeliski 2006b), as well as for multi-scale oriented filtering (Simoncelli, Freeman, Adelson *et al.* 1992) and denoising (Portilla, Strela,



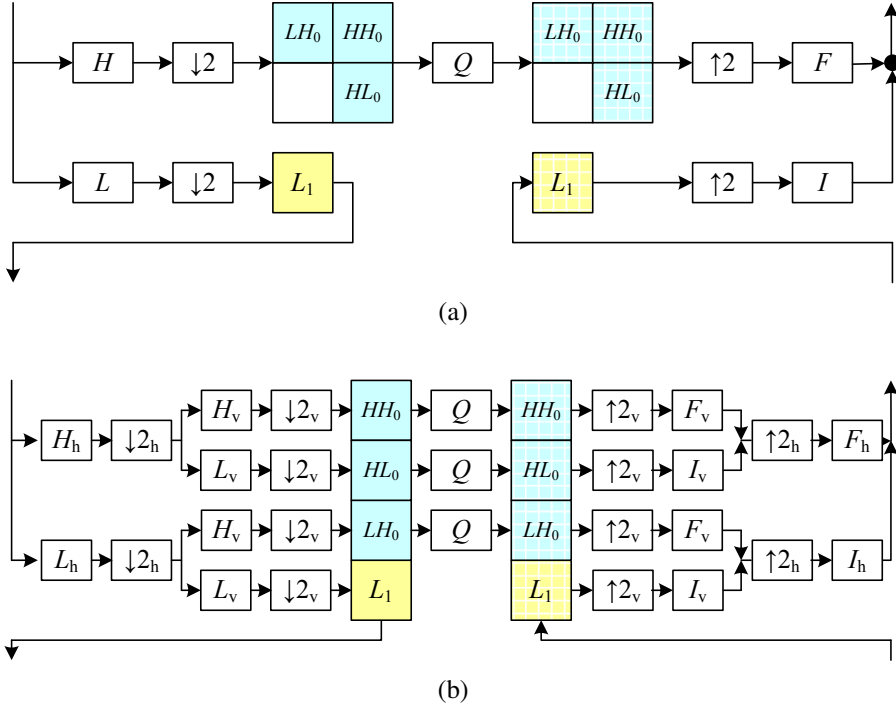
**Figure 3.36** Multiresolution pyramids: (a) pyramid with half-octave (*quincunx*) sampling (odd levels are colored gray for clarity). (b) wavelet pyramid—each wavelet level stores  $3/4$  of the original pixels (usually the horizontal, vertical, and mixed gradients), so that the total number of wavelet coefficients and original pixels is the same.

Wainwright *et al.* 2003).

Since both image pyramids and wavelets decompose an image into multi-resolution descriptions that are localized in both space and frequency, how do they differ? The usual answer is that traditional pyramids are *overcomplete*, i.e., they use more pixels than the original image to represent the decomposition, whereas wavelets provide a *tight frame*, i.e., they keep the size of the decomposition the same as the image (Figure 3.36b). However, some wavelet families *are*, in fact, overcomplete in order to provide better shiftability or steering in orientation (Simoncelli, Freeman, Adelson *et al.* 1992). A better distinction, therefore, might be that wavelets are more orientation selective than regular band-pass pyramids.

How are two-dimensional wavelets constructed? Figure 3.37a shows a high-level diagram of one stage of the (recursive) coarse-to-fine construction (analysis) pipeline alongside the complementary re-construction (synthesis) stage. In this diagram, the high-pass filter followed by decimation keeps  $3/4$  of the original pixels, while  $1/4$  of the low-frequency coefficients are passed on to the next stage for further analysis. In practice, the filtering is usually broken down into two separable sub-stages, as shown in Figure 3.37b. The resulting three wavelet images are sometimes called the high-high (*HH*), high-low (*HL*), and low-high (*LH*) images. The high-low and low-high images accentuate the horizontal and vertical edges and gradients, while the high-high image contains the less frequently occurring mixed derivatives.

How are the high-pass *H* and low-pass *L* filters shown in Figure 3.37b chosen and how can the corresponding reconstruction filters *I* and *F* be computed? Can filters be designed

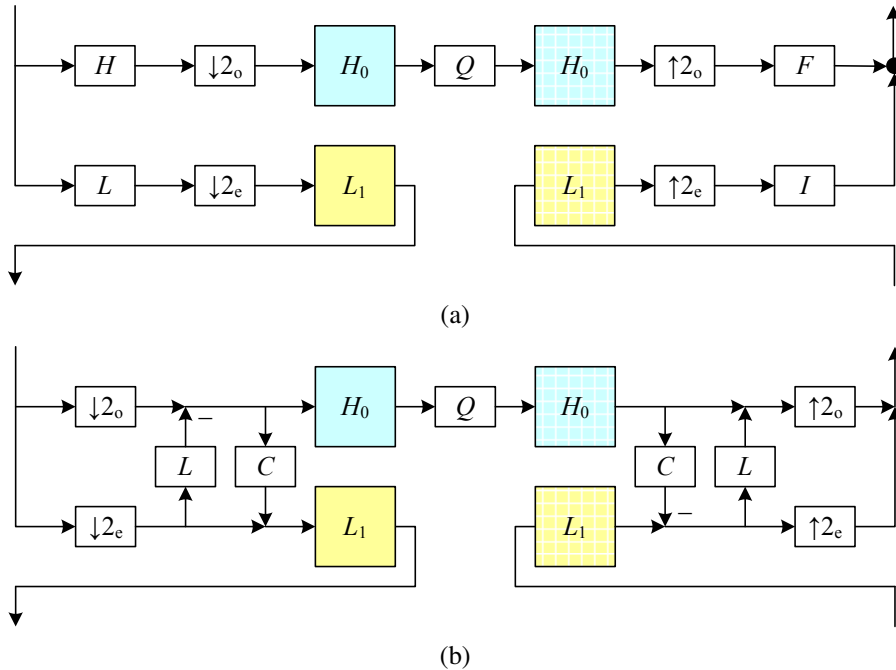


**Figure 3.37** Two-dimensional wavelet decomposition: (a) high-level diagram showing the low-pass and high-pass transforms as single boxes; (b) separable implementation, which involves first performing the wavelet transform horizontally and then vertically. The  $I$  and  $F$  boxes are the interpolation and filtering boxes required to re-synthesize the image from its wavelet components.

that all have finite impulse responses? This topic has been the main subject of study in the wavelet community for over two decades. The answer depends largely on the intended application, e.g., whether the wavelets are being used for compression, image analysis (feature finding), or denoising. [Simoncelli and Adelson \(1990b\)](#) show (in Table 4.1) some good odd-length quadrature mirror filter (QMF) coefficients that seem to work well in practice.

Since the design of wavelet filters is such a tricky art, is there perhaps a better way? Indeed, a simpler procedure is to split the signal into its even and odd components and then perform trivially reversible filtering operations on each sequence to produce what are called *lifted wavelets* (Figures 3.38 and 3.39). [Sweldens \(1996\)](#) gives a wonderfully understandable introduction to the *lifting scheme* for *second-generation wavelets*, followed by a comprehensive review ([Sweldens 1997](#)).

As Figure 3.38 demonstrates, rather than first filtering the whole input sequence (image)

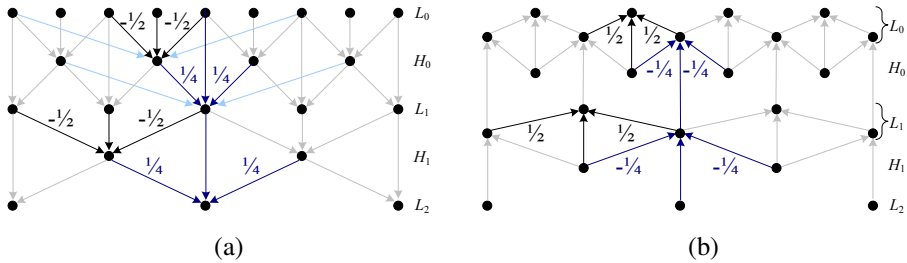


**Figure 3.38** One-dimensional wavelet transform: (a) usual high-pass + low-pass filters followed by odd ( $\downarrow 2_o$ ) and even ( $\downarrow 2_e$ ) downsampling; (b) lifted version, which first selects the odd and even subsequences and then applies a low-pass prediction stage  $L$  and a high-pass correction stage  $C$  in an easily reversible manner.

with high-pass and low-pass filters and then keeping the odd and even sub-sequences, the lifting scheme first splits the sequence into its even and odd sub-components. Filtering the even sequence with a low-pass filter  $L$  and subtracting the result from the even sequence is trivially reversible: simply perform the same filtering and then add the result back in. Furthermore, this operation can be performed in place, resulting in significant space savings. The same applies to filtering the even sequence with the correction filter  $C$ , which is used to ensure that the even sequence is low-pass. A series of such *lifting* steps can be used to create more complex filter responses with low computational cost and guaranteed reversibility.

This process can perhaps be more easily understood by considering the signal processing diagram in Figure 3.39. During analysis, the average of the even values is subtracted from the odd value to obtain a high-pass wavelet coefficient. However, the even samples still contain an aliased sample of the low-frequency signal. To compensate for this, a small amount of the high-pass wavelet is added back to the even sequence so that it is properly low-pass filtered. (It is easy to show that the effective low-pass filter is  $[-1/8, 1/4, 3/4, 1/4, -1/8]$ , which is in-





**Figure 3.39** Lifted transform shown as a signal processing diagram: (a) The analysis stage first predicts the odd value from its even neighbors, stores the difference wavelet, and then compensates the coarser even value by adding in a fraction of the wavelet. (b) The synthesis stage simply reverses the flow of computation and the signs of some of the filters and operations. The light blue lines show what happens if we use four taps for the prediction and correction instead of just two.

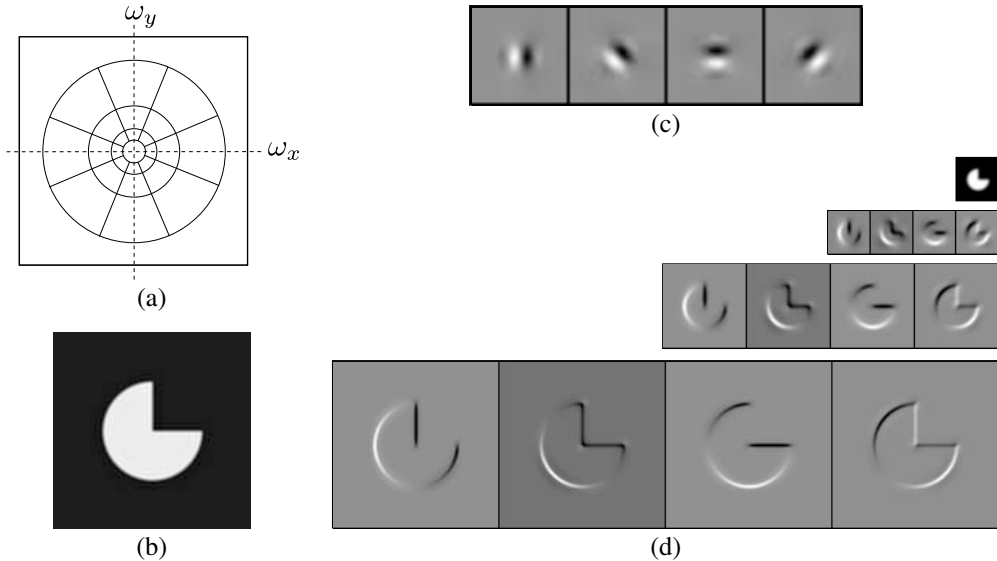
deed a low-pass filter.) During synthesis, the same operations are reversed with a judicious change in sign.

Of course, we need not restrict ourselves to two-tap filters. Figure 3.39 shows as light blue arrows additional filter coefficients that could optionally be added to the lifting scheme without affecting its reversibility. In fact, the low-pass and high-pass filtering operations can be interchanged, e.g., we could use a five-tap cubic low-pass filter on the odd sequence (plus center value) first, followed by a four-tap cubic low-pass predictor to estimate the wavelet, although I have not seen this scheme written down.

Lifted wavelets are called *second-generation wavelets* because they can easily adapt to non-regular sampling topologies, e.g., those that arise in computer graphics applications such as multi-resolution surface manipulation (Schröder and Sweldens 1995). It also turns out that lifted *weighted wavelets*, i.e., wavelets whose coefficients adapt to the underlying problem being solved (Fattal 2009), can be extremely effective for low-level image manipulation tasks and also for preconditioning the kinds of sparse linear systems that arise in the optimization-based approaches to vision algorithms that we discuss in Section 3.7 (Szeliski 2006b).

An alternative to the widely used “separable” approach to wavelet construction, which decomposes each level into horizontal, vertical, and “cross” sub-bands, is to use a representation that is more rotationally symmetric and orientationally selective and also avoids the aliasing inherent in sampling signals below their Nyquist frequency.<sup>17</sup> Simoncelli, Freeman, Adelson *et al.* (1992) introduce such a representation, which they call a *pyramidal radial frequency*

<sup>17</sup> Such aliasing can often be seen as the signal content moving between bands as the original signal is slowly shifted.



**Figure 3.40** Steerable shiftable multiscale transforms (Simoncelli, Freeman, Adelson *et al.* 1992) © 1992 IEEE: (a) radial multi-scale frequency domain decomposition; (b) original image; (c) a set of four steerable filters; (d) the radial multi-scale wavelet decomposition.

implementation of *shiftable multi-scale transforms* or, more succinctly, *steerable pyramids*. Their representation is not only overcomplete (which eliminates the aliasing problem) but is also orientationally selective and has identical analysis and synthesis basis functions, i.e., it is *self-inverting*, just like “regular” wavelets. As a result, this makes steerable pyramids a much more useful basis for the structural analysis and matching tasks commonly used in computer vision.

Figure 3.40a shows how such a decomposition looks in frequency space. Instead of recursively dividing the frequency domain into  $2 \times 2$  squares, which results in checkerboard high frequencies, radial arcs are used instead. Figure 3.40b illustrates the resulting pyramid sub-bands. Even though the representation is *overcomplete*, i.e., there are more wavelet coefficients than input pixels, the additional frequency and orientation selectivity makes this representation preferable for tasks such as texture analysis and synthesis (Portilla and Simoncelli 2000) and image denoising (Portilla, Strele, Wainwright *et al.* 2003; Lyu and Simoncelli 2009).