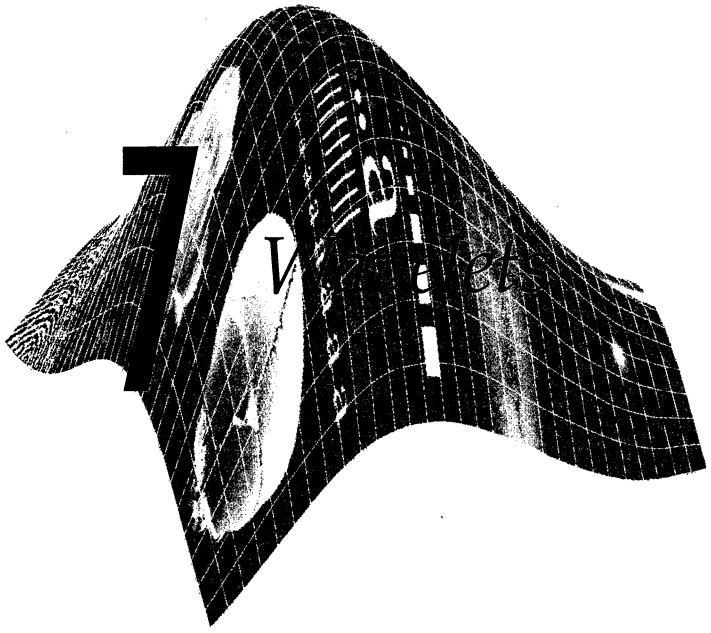


Preview

When digital images are to be viewed or processed at multiple resolutions, the *discrete wavelet transform* (DWT) is the mathematical tool of choice. In addition to being an efficient, highly intuitive framework for the representation and storage of *multiresolution* images, the DWT provides powerful insight into an image's spatial and frequency characteristics. The Fourier transform, on the other hand, reveals only an image's frequency attributes.

In this chapter, we explore both the computation and use of the discrete wavelet transform. We introduce the *Wavelet Toolbox*, a collection of MathWorks' functions designed for wavelet analysis but not included in MATLAB's *Image Processing Toolbox* (IPT), and develop a compatible set of routines that allow basic wavelet-based processing using IPT alone; that is, without the Wavelet Toolbox. These custom functions, in combination with IPT, provide the tools needed to implement all the concepts discussed in Chapter 7 of *Digital Image Processing* by Gonzalez and Woods [2002]. They are applied in much the same way—and provide a similar range of capabilities—as IPT functions `fft2` and `ifft2` in Chapter 4.

7.1 Background

Consider an image $f(x, y)$ of size $M \times N$ whose forward, discrete transform, $T(u, v, \ldots)$, can be expressed in terms of the general relation

$$T(u, v, \ldots) = \sum_{x, y} f(x, y) g_{u, v, \ldots}(x, y)$$

where x and y are spatial variables and $u, v, \ldots$ are *transform domain variables*. Given $T(u, v, \ldots)$, $f(x, y)$ can be obtained using the generalized inverse discrete transform

$$f(x, y) = \sum_{u, v, \ldots} T(u, v, \ldots) h_{u, v, \ldots}(x, y)$$

The $g_{u, v, \ldots}$ and $h_{u, v, \ldots}$ in these equations are called *forward* and *inverse transformation kernels*, respectively. They determine the nature, computational complexity, and ultimate usefulness of the *transform pair*. *Transform coefficients* $T(u, v, \ldots)$ can be viewed as the *expansion coefficients* of a series expansion of f with respect to $\{h_{u, v, \ldots}\}$. That is, the inverse transformation kernel defines a set of *expansion functions* for the series expansion of f.

The discrete Fourier transform (DFT) of Chapter 4 fits this series expansion formulation well.† In this case

$$h_{u, v}(x, y) = g^*_{u, v}(x, y) = \frac{1}{\sqrt{MN}} e^{j2\pi(ux/M + vy/N)}$$

where $j = \sqrt{-1}$, $*$ is the complex conjugate operator, $u = 0, 1, \ldots, M - 1$, and $v = 0, 1, \ldots, N - 1$. Transform domain variables v and u represent horizontal and vertical frequency, respectively. The kernels are *separable* since

$$h_{u, v}(x, y) = h_u(x) h_v(y)$$

for

$$h_u(x) = \frac{1}{\sqrt{M}} e^{j2\pi ux/M} \quad \text{and} \quad h_v(y) = \frac{1}{\sqrt{N}} e^{j2\pi vy/N}$$

and *orthonormal* since

$$\langle h_r, h_s \rangle = \delta_{rs} = \begin{cases} 1 & r = s \\ 0 & \text{otherwise} \end{cases}$$

where $\langle\ \rangle$ is the inner product operator. The separability of the kernels simplifies the computation of the 2-D transform by allowing row-column or column-row passes of a 1-D transform to be used; orthonormality causes the forward and inverse kernels to be the complex conjugates of one another (they would be identical if the functions were real).

Unlike the discrete Fourier transform, which can be completely defined by two straightforward equations that revolve around a single pair of transformation kernels (given previously), the term *discrete wavelet transform* refers to a class of transformations that differ not only in the transformation kernels employed (and thus the expansion functions used), but also in the fundamental nature of those functions (e.g., whether they constitute an orthonormal or biorthogonal basis) and in the way in which they are applied (e.g., how many different resolutions are computed). Since the DWT encompasses a variety of unique but related transformations, we cannot write a single equation that

†In the DFT formulation of Chapter 4, a $1/MN$ term is placed in the inverse transform equation. Equivalently, it can be incorporated into the forward transform only, or split, as we do here, between the forward and inverse transformations as $1/\sqrt{MN}$.

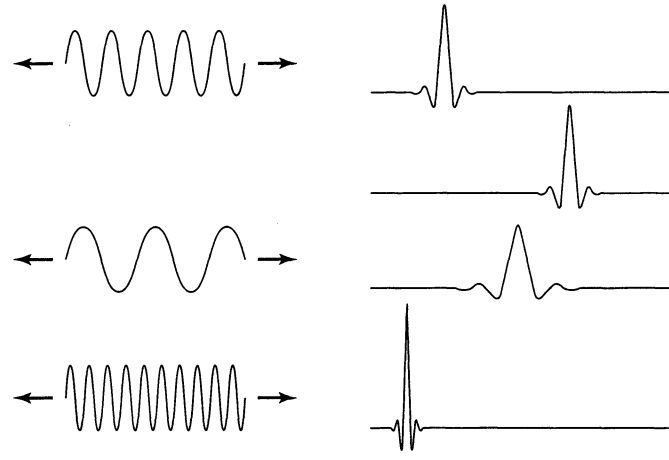


FIGURE 7.1
(a) The familiar Fourier expansion functions are sinusoids of varying frequency and infinite duration.
(b) DWT expansion functions are "small waves" of finite duration and varying frequency.

completely describes them all. Instead, we characterize each DWT by a transform kernel pair *or* set of parameters that defines the pair. The various transforms are related by the fact that their expansion functions are "small waves" (hence the name *wavelets*) of varying frequency and limited duration [see Fig. 7.1(b)]. In the remainder of the chapter, we introduce a number of these "small wave" kernels. Each possesses the following general properties:

Property 1: *Separability, Scalability, and Translatability.* The kernels can be represented as three separable 2-D *wavelets*

$$\psi^H(x, y) = \psi(x)\varphi(y)$$
$$\psi^V(x, y) = \varphi(x)\psi(y)$$
$$\psi^D(x, y) = \psi(x)\psi(y)$$

where $\psi^H(x, y)$, $\psi^V(x, y)$, and $\psi^D(x, y)$ are called *horizontal, vertical*, and *diagonal wavelets*, respectively, and one separable 2-D *scaling function*

$$\varphi(x, y) = \varphi(x)\varphi(y)$$

Each of these 2-D functions is the product of two 1-D real, square-integrable scaling and wavelet functions

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k)$$
$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k)$$

Translation k determines the position of these 1-D functions along the x-axis, *scale* j determines their width—how broad or narrow they are along x—and $2^{j/2}$ controls their height or amplitude. Note that the associated expansion functions are binary scalings and integer translates of *mother* wavelet $\psi(x) = \psi_{0,0}(x)$ and scaling function $\varphi(x) = \varphi_{0,0}(x)$.

Property 2: *Multiresolution Compatibility.* The 1-D scaling function just introduced satisfies the following requirements of multiresolution analysis:

a. $\varphi_{j,k}$ is orthogonal to its integer translates.
b. The set of functions that can be represented as a series expansion of $\varphi_{j,k}$ at low scales or resolutions (i.e., small j) is contained within those that can be represented at higher scales.
c. The only function that can be represented at every scale is $f(x) = 0$.
d. Any function can be represented with arbitrary precision as $j \rightarrow \infty$.

When these conditions are met, there is a companion wavelet $\psi_{j,k}$ that, together with its integer translates and binary scalings, spans—that is, can represent—the difference between any two sets of $\varphi_{j,k}$-representable functions at adjacent scales.

Property 3: *Orthogonality*. The expansion functions [i.e.,$\{\varphi_{j,k}(x)\}$] form an orthonormal or biorthogonal *basis* for the set of 1-D measurable, square-integrable functions. To be called a basis, there must be a unique set of expansion coefficients for every representable function. As was noted in the introductory remarks on Fourier kernels, $g_{u,v,\ldots} = h_{u,v,\ldots}$ for real, orthonormal kernels. For the biorthogonal case,

$$\langle h_r, g_s \rangle = \delta_{rs} = \begin{cases} 1 & r = s \\ 0 & \text{otherwise} \end{cases}$$

and g is called the *dual* of h. For a biorthogonal wavelet transform with scaling and wavelet functions $\varphi_{j,k}(x)$ and $\psi_{j,k}(x)$, the duals are denoted $\tilde{\varphi}_{j,k}(x)$ and $\tilde{\psi}_{j,k}(x)$, respectively.

7.2 The Fast Wavelet Transform

An important consequence of the above properties is that both $\varphi(x)$ and $\psi(x)$ can be expressed as linear combinations of double-resolution copies of themselves. That is, via the series expansions

$$\varphi(x) = \sum_n h_\varphi(n)\sqrt{2}\varphi(2x - n)$$
$$\psi(x) = \sum_n h_\psi(n)\sqrt{2}\varphi(2x - n)$$

where h_φ and h_ψ—the expansion coefficients—are called *scaling* and *wavelet vectors*, respectively. They are the filter coefficients of the *fast wavelet transform* (FWT), an iterative computational approach to the DWT shown in Fig. 7.2. The $W_\varphi(j, m, n)$ and $\{W_\psi^i(j, m, n)$ for $i = H, V, D\}$ outputs in this figure are the DWT coefficients at scale j. Blocks containing time-reversed scaling and wavelet vectors—the $h_\varphi(-n)$ and $h_\psi(-m)$—are *lowpass* and *highpass decomposition filters*, respectively. Finally, blocks containing a 2 and a down arrow represent *downsampling*—extracting every other point from a sequence of points. Mathematically, the series of filtering and downsampling operations used to compute $W_\psi^H(j, m, n)$ in Fig. 7.2 is, for example,

$$W_\psi^H(j, m, n) = h_\psi(-m) * [h_\varphi(-n) * W_\varphi(j + 1, m, n)|_{n=2k,\, k\geq 0}]|_{m=2k,\, k\geq 0}$$

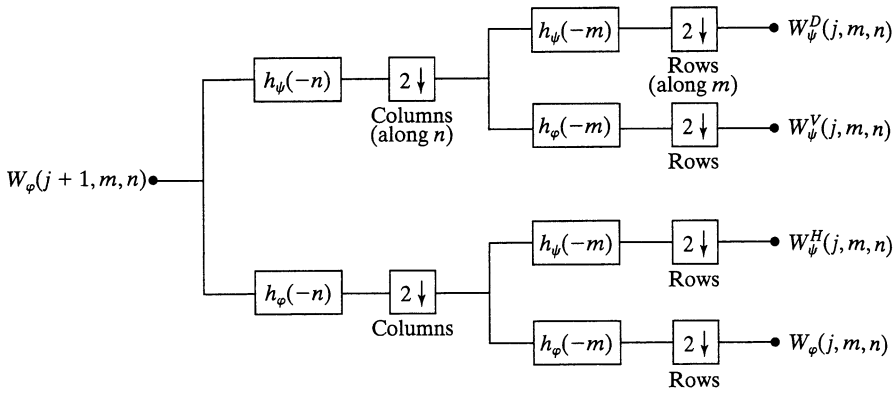


FIGURE 7.2 The 2-D fast wavelet transform (FWT) filter bank. Each pass generates one DWT scale. In the first iteration, $W_\varphi(j+1,m,n) = f(x,y)$.

where * denotes convolution. Evaluating convolutions at nonnegative, even indices is equivalent to filtering and downsampling by 2.

Each pass through the filter bank in Fig. 7.2 decomposes the input into four lower resolution (or lower scale) components. The W_φ coefficients are created via two lowpass (i.e., h_φ-based) filters and are thus called *approximation coefficients*; $\{W_\psi^i$ for $i = H, V, D\}$ are *horizontal, vertical,* and *diagonal detail coefficients*, respectively. Since $f(x,y)$ is the highest resolution representation of the image being transformed, it serves as the $W_\varphi(j+1,m,n)$ input for the first iteration. Note that the operations in Fig. 7.2 use *neither* wavelets nor scaling functions—only their associated wavelet and scaling vectors. In addition, three transform domain variables are involved—scale j and horizontal and vertical translation, n and m. These variables correspond to $u, v, \ldots$ in the first two equations of Section 7.1.

7.2.1 FWTs Using the Wavelet Toolbox

In this section, we use MATLAB's Wavelet Toolbox to compute the FWT of a simple 4 × 4 test image. In the next section, we will develop custom functions to do this without the Wavelet Toolbox (i.e., with IPT alone). The material here lays the groundwork for their development.

The Wavelet Toolbox provides decomposition filters for a wide variety of fast wavelet transforms. The filters associated with a specific transform are accessed via the function `wfilters`, which has the following general syntax:

The **W** *on the icon is used to denote a MATLAB Wavelet Toolbox function, as opposed to a MATLAB or Image Processing Toolbox function.*

```
[Lo_D, Hi_D, Lo_R, Hi_R] = wfilters(wname)
```

Here, input parameter `wname` determines the returned filter coefficients in accordance with Table 7.1; outputs `Lo_D`, `Hi_D`, `Lo_R`, and `Hi_R` are row vectors that return the lowpass decomposition, highpass decomposition, lowpass reconstruction, and highpass reconstruction filters, respectively. (Reconstruction filters are discussed in Section 7.4.) Frequently coupled filter pairs can alternately be retrieved using

```
[F1, F2] = wfilters(wname, type)
```

Wavelet	wfamily	wname
Haar	'haar'	'haar'
Daubechies	'db'	'db2', 'db3',..., 'db45'
Coiflets	'coif'	'coif1','coif2',..., 'coif5'
Symlets	'sym'	'sym2','sym3',..., 'sym45'
Discrete Meyer	'dmey'	'dmey'
Biorthogonal	'bior'	'bior1.1','bior1.3','bior1.5','bior2.2', 'bior2.4','bior2.6','bior2.8','bior3.1', 'bior3.3','bior3.5','bior3.7','bior3.9', 'bior4.4','bior5.5','bior6.8'
Reverse Biorthogonal	'rbio'	'rbio1.1','rbio1.3','rbio1.5','rbio2.2', 'rbio2.4','rbio2.6','rbio2.8','rbio3.1', 'rbio3.3','rbio3.5','rbio3.7','rbio3.9', 'rbio4.4','rbio5.5','rbio6.8'

TABLE 7.1 Wavelet Toolbox FWT filters and filter family names.

with `type` set to `'d'`, `'r'`, `'l'`, or `'h'` to obtain a pair of decomposition, reconstruction, lowpass, or highpass filters, respectively. If this syntax is employed, a decomposition or lowpass filter is returned in `F1`, and its companion is placed in `F2`.

Table 7.1 lists the FWT filters included in the Wavelet Toolbox. Their properties—and other useful information on the associated scaling and wavelet functions—is available in the literature on digital filtering and multiresolution analysis. Some of the more important properties are provided by the Wavelet Toolbox's `waveinfo` and `wavefun` functions. To print a written description of wavelet family `wfamily` (see Table 7.1) on MATLAB's Command Window, for example, enter

```
waveinfo(wfamily)
```

at the MATLAB prompt. To obtain a digital approximation of an orthonormal transform's scaling and/or wavelet functions, type

```
[phi, psi, xval] = wavefun(wname, iter)
```

which returns approximation vectors, `phi` and `psi`, and evaluation vector `xval`. Positive integer `iter` determines the accuracy of the approximations by controlling the number of iterations used in their computation. For biorthogonal transforms, the appropriate syntax is

```
[phi1, psi1, phi2, psi2, xval] = wavefun(wname, iter)
```

where `phi1` and `psi1` are decomposition functions and `phi2` and `psi2` are reconstruction functions.

EXAMPLE 7.1: Haar filters, scaling, and wavelet functions.

■ The oldest and simplest wavelet transform is based on the Haar scaling and wavelet functions. The decomposition and reconstruction filters for a Haar-based transform are of length 2 and can be obtained as follows:

```
>> [Lo_D, Hi_D, Lo_R, Hi_R] = wfilters('haar')
Lo_D =
    0.7071    0.7071
Hi_D =
   -0.7071    0.7071
Lo_R =
    0.7071    0.7071
Hi_R =
    0.7071   -0.7071
```

Their key properties (as reported by the `waveinfo` function) and plots of the associated scaling and wavelet functions can be obtained using

```
>> waveinfo('haar');
HAARINFO Information on Haar wavelet.

    Haar Wavelet

    General characteristics: Compactly supported
    wavelet, the oldest and the simplest wavelet.

    scaling function phi = 1 on [0 1] and 0 otherwise.
    wavelet function psi = 1 on [0 0.5], = -1 on [0.5 1] and 0
    otherwise.

    Family                      Haar
    Short name                  haar
    Examples                    haar is the same as db1
    Orthogonal                  yes
    Biorthogonal                yes
    Compact support             yes
    DWT                         possible
    CWT                         possible

    Support width               1
    Filters length              2
    Regularity                  haar is not continuous
    Symmetry                    yes
    Number of vanishing
    moments for psi             1

    Reference: I. Daubechies,
    Ten lectures on wavelets,
    CBMS, SIAM, 61, 1994, 194-202.

>> [phi, psi, xval] = wavefun('haar', 10);
>> xaxis = zeros(size(xval));
>> subplot(121); plot(xval, phi, 'k', xval, xaxis, '--k');
>> axis([0 1 -1.5 1.5]); axis square;
>> title('Haar Scaling Function');
```

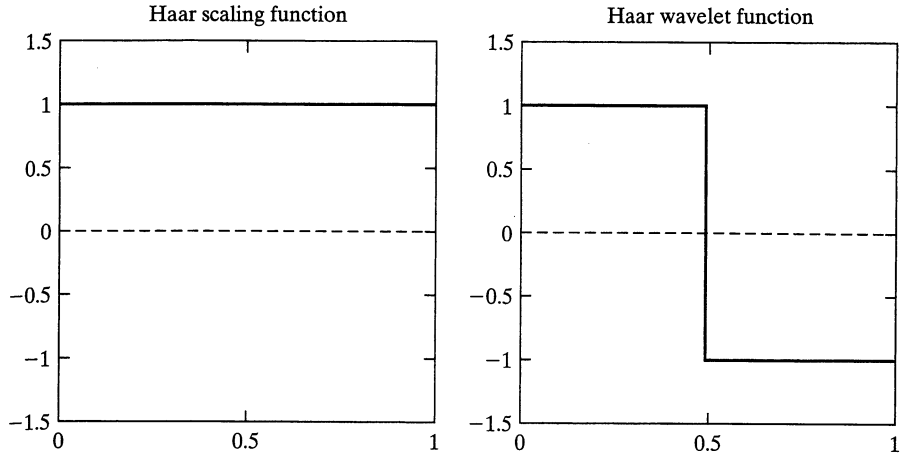


FIGURE 7.3 The Haar scaling and wavelet functions.

```
>> subplot(122); plot(xval, psi, 'k', xval, xaxis, '--k');
>> axis([0 1 -1.5 1.5]); axis square;
>> title('Haar Wavelet Function');
```

Figure 7.3 shows the display generated by the final six commands. Functions `title`, `axis`, and `plot` were described in Chapters 2 and 3; function `subplot` is used to subdivide the figure window into an array of axes or subplots. It has the following generic syntax:

```
H = subplot(m, n, p) or H = subplot(mnp)
```

where `m` and `n` are the number of rows and columns in the subplot array, respectively. Both `m` and `n` must be greater than 1. Optional output variable `H` is the handle of the subplot (i.e., axes) selected by `p`, with incremental values of `p` (beginning at 1) selecting axes along the top row of the figure window, then the second row, and so on. With or without `H`, the `p`th axes is made the current plot. Thus, the `subplot(122)` function in the commands given previously selects the plot in row 1 and column 2 of a 1×2 subplot array as the current plot; the subsequent `axis` and `title` functions then apply only to it.

The Haar scaling and wavelet functions shown in Figure 7.3 are discontinuous and *compactly supported*, which means they are 0 outside a finite interval called the *support*. Note that the support is 1. In addition, the `waveinfo` data reveals that the Haar expansion functions are orthogonal, so that the forward and inverse transformation kernels are identical. ■

Given a set of decomposition filters, whether user provided or generated by the `wfilters` function, the simplest way of computing the associated wavelet transform is through the Wavelet Toolbox's `wavedec2` function. It is invoked using

```
[C, S] = wavedec2(X, N, Lo_D, Hi_D)
```

where `X` is a 2-D image or matrix, `N` is the number of scales to be computed (i.e., the number of passes through the FWT filter bank in Fig. 7.2), and `Lo_D` and `Hi_D` are decomposition filters. The slightly more efficient syntax

```
[C, S] = wavedec2(X, N, wname)
```

in which `wname` assumes a value from Table 7.1, can also be used. Output data structure `[C, S]` is composed of row vector `C` (class `double`), which contains the computed wavelet transform coefficients, and bookkeeping matrix `S` (also class `double`), which defines the arrangement of the coefficients in `C`. The relationship between `C` and `S` is introduced in the next example and described in detail in Section 7.3.

EXAMPLE 7.2: A simple FWT using Haar filters.

■ Consider the following single-scale wavelet transform with respect to Haar wavelets:

```
>> f = magic(4)
f =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> [c1, s1] = wavedec2(f, 1, 'haar')
c1 =
  Columns 1 through 9
   17.0000   17.0000   17.0000   17.0000    1.0000
   -1.0000   -1.0000    1.0000    4.0000

  Columns 10 through 16
   -4.0000   -4.0000    4.0000   10.0000    6.0000
   -6.0000  -10.0000

s1 =
     2     2
     2     2
     4     4
```

Here, a 4 × 4 magic square `f` is transformed into a 1 × 16 wavelet decomposition vector `c1` and 3 × 2 bookkeeping matrix `s1`. The entire transformation is performed with a single execution (with `f` used as the input) of the operations depicted in Fig. 7.2. Four 2 × 2 outputs—a downsampled approximation and three directional (horizontal, vertical, and diagonal) detail matrices—are generated. Function `wavedec2` concatenates these 2 × 2 matrices columnwise in row vector `c1` beginning with the approximation coefficients and continuing with the horizontal, vertical, and diagonal details. That is, `c1(1)` through `c1(4)` are approximation coefficients $W_\varphi(1, 0, 0)$, $W_\varphi(1, 1, 0)$, $W_\varphi(1, 0, 1)$, and $W_\varphi(1, 1, 1)$ from Fig. 7.2 with the scale of `f` assumed arbitrarily to be 2; `c1(5)` through `c1(8)` are $W_\psi^H(1, 0, 0)$, $W_\psi^H(1, 1, 0)$, $W_\psi^H(1, 0, 1)$, and $W_\psi^H(1, 1, 1)$;

and so on. It we were to extract the horizontal detail coefficient matrix from vector `c1`, for example, we would get

$$W_\psi^H = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Bookkeeping matrix `s1` provides the sizes of the matrices that have been concatenated a column at a time into row vector `c1`—plus the size of the original image `f` [in vector `s1(end, :)`]. Vectors `s1(1, :)` and `s1(2, :)` contain the sizes of the computed approximation matrix and three detail coefficient matrices, respectively. The first element of each vector is the number of rows in the referenced detail or approximation matrix; the second element is the number of columns.

When the single-scale transform described above is extended to two scales, we get

```
>> [c2, s2] = wavedec2(f, 2, 'haar')
c2 =
    Columns 1 through 9
        34.0000         0         0    0.0000    1.0000
        -1.0000   -1.0000    1.0000    4.0000

    Columns 10 through 16
        -4.0000   -4.0000    4.0000   10.0000    6.0000
        -6.0000  -10.0000

s2 =
     1     1
     1     1
     2     2
     4     4
```

Note that `c2(5:16) = c1(5:16)`. Elements `c1(1:4)`, which were the approximation coefficients of the single-scale transform, have been fed into the filter bank of Fig. 7.2 to produce four 1×1 outputs: $W_\varphi(0, 0, 0)$, $W_\psi^H(0, 0, 0)$, $W_\psi^V(0, 0, 0)$, and $W_\psi^D(0, 0, 0)$. These outputs are concatenated columnwise (though they are 1×1 matrices here) in the same order that was used in the preceding single-scale transform and substituted for the approximation coefficients from which they were derived. Bookkeeping matrix `s2` is then updated to reflect the fact that the single 2×2 approximation matrix in `c1` has been replaced by four 1×1 detail and approximation matrices in `c2`. Thus, `s2(end, :)` is once again the size of the original image, `s2(3, :)` is the size of the three detail coefficient matrices at scale 1, `s2(2, :)` is the size of the three detail coefficient matrices at scale 0, and `s2(1, :)` is the size of the final approximation. ■

To conclude this section, we note that because the FWT is based on digital filtering techniques and thus convolution, border distortions can arise. To minimize these distortions, the border must be treated differently from the other

TABLE 7.2 Wavelet Toolbox image extension or padding modes.

STATUS	Description
'sym'	The image is extended by mirror reflecting it across its borders. This is the normal default mode.
'zpd'	The image is extended by padding with a value of 0.
'spd', 'sp1'	The image is extended by first-order derivative extrapolation—or padding with a linear extension of the outmost two border values.
'sp0'	The image is extended by extrapolating the border values—that is, by boundary value replication.
'ppd'	The image is extended by periodic padding.
'per'	The image is extended by periodic padding after it has been padded (if necessary) to an even size using 'sp0' extension.

parts of the image. When filter elements fall outside the image during the convolution process, values must be assumed for the area, which is about the size of the filter, outside the image. Many Wavelet Toolbox functions, including the `wavedec2` function, extend or pad the image being processed based on global parameter `dwtmode`. To examine the active extension mode, enter `st = dwtmode('status')` or simply `dwtmode` at the MATLAB command prompt (e.g., `>> dwtmode`). To set the extension mode to `STATUS`, enter `dwtmode(STATUS);` to make `STATUS` the default extension mode, use `dwtmode('save', STATUS)`. The supported extension modes and corresponding `STATUS` values are listed in Table 7.2.

7.2.2 FWTs without the Wavelet Toolbox

In this section, we develop a pair of custom functions, `wavefilter` and `wavefast`, to replace the Wavelet Toolbox functions, `wfilters` and `wavedec2`, of the previous section. Our goal is to provide additional insight into the mechanics of computing FWTs, and to begin the process of building a "stand-alone package" for basic wavelet-based image processing without the Wavelet Toolbox. This process is completed in Sections 7.3 and 7.4, and the resulting set of functions is used to generate the examples in Section 7.5.

The first step is to devise a function for generating wavelet decomposition and reconstruction filters. The following function, which we call `wavefilter`, uses a standard `switch` construct, together with `case` and `otherwise`, to do this in a readily extendable manner. Although `wavefilter` provides only the filters examined in Chapters 7 and 8 of *Digital Image Processing* (Gonzalez and Woods [2002]), other wavelet transforms can be accommodated by adding (as new "cases") the appropriate decomposition and reconstruction filters from the literature.

wavefilter

```
function [varargout] = wavefilter(wname, type)
%WAVEFILTER Create wavelet decomposition and reconstruction filters.
%   [VARARGOUT] = WAVEFILTER(WNAME, TYPE) returns the decomposition
%   and/or reconstruction filters used in the computation of the
%   forward and inverse FWT (fast wavelet transform).
```

```
%
%   EXAMPLES:
%     [ld, hd, lr, hr] = wavefilter('haar') Get the low and highpass
%                                           decomposition (ld, hd)
%                                           and reconstruction
%                                           (lr, hr) filters for
%                                           wavelet 'haar'.
%     [ld, hd] = wavefilter('haar','d')     Get decomposition filters
%                                           ld and hd.
%     [lr, hr] = wavefilter('haar','r')     Get reconstruction
%                                           filters lr and hr.
%
%   INPUTS:
%     WNAME                       Wavelet Name
%     ---------------------------------------------------------------
%     'haar' or 'db1'             Haar
%     'db4'                       4th order Daubechies
%     'sym4'                      4th order Symlets
%     'bior6.8'                   Cohen-Daubechies-Feauveau biorthogonal
%     'jpeg9.7'                   Antonini-Barlaud-Mathieu-Daubechies
%
%     TYPE                        Filter Type
%     ---------------------------------------------------------------
%     'd'                         Decomposition filters
%     'r'                         Reconstruction filters
%
%   See also WAVEFAST and WAVEBACK.

% Check the input and output arguments.
error(nargchk(1, 2, nargin));

if (nargin == 1 & nargout ~= 4) | (nargin == 2 & nargout ~= 2)
   error('Invalid number of output arguments.');
end

if nargin == 1 & ~ischar(wname)
   error('WNAME must be a string.');
end

if nargin == 2 & ~ischar(type)
   error('TYPE must be a string.');
end

% Create filters for the requested wavelet.
switch lower(wname)
case {'haar', 'db1'}
   ld = [1 1]/sqrt(2);     hd = [-1 1]/sqrt(2);
   lr = ld;                hr = -hd;

case 'db4'
   ld = [-1.059740178499728e-002 3.288301166698295e-002 ...
         3.084138183598697e-002 -1.870348117188811e-001 ...
         -2.798376941698385e-002 6.308807679295904e-001 ...
         7.148465705525415e-001 2.303778133088552e-001];
```

```
        t = (0:7);
        hd = ld;   hd(end:-1:1) = cos(pi * t) .* ld;
        lr = ld;   lr(end:-1:1) = ld;
        hr = cos(pi * t) .* ld;

    case 'sym4'
        ld = [-7.576571478927333e-002 -2.963552764599851e-002 ...
              4.976186676320155e-001 8.037387518059161e-001 ...
              2.978577956052774e-001 -9.921954357684722e-002 ...
              -1.260396726203783e-002 3.222310060404270e-002];
        t = (0:7);
        hd = ld;   hd(end:-1:1) = cos(pi * t) .* ld;
        lr = ld;   lr(end:-1:1) = ld;
        hr = cos(pi * t) .* ld;

    case 'bior6.8'
        ld = [0 1.908831736481291e-003 -1.914286129088767e-003 ...
              -1.699063986760234e-002 1.193456527972926e-002 ...
              4.973290349094079e-002 -7.726317316720414e-002 ...
              -9.405920349573646e-002 4.207962846098268e-001 ...
              8.259229974584023e-001 4.207962846098268e-001 ...
              -9.405920349573646e-002 -7.726317316720414e-002 ...
              4.973290349094079e-002 1.193456527972926e-002 ...
              -1.699063986760234e-002 -1.914286129088767e-003 ...
              1.908831736481291e-003];
        hd = [0 0 0 1.442628250562444e-002 -1.446750489679015e-002 ...
              -7.872200106262882e-002 4.036797903033992e-002 ...
              4.178491091502746e-001 -7.589077294536542e-001 ...
              4.178491091502746e-001 4.036797903033992e-002 ...
              -7.872200106262882e-002 -1.446750489679015e-002 ...
              1.442628250562444e-002 0 0 0 0];
        t = (0:17);
        lr = cos(pi * (t + 1)) .* hd;
        hr = cos(pi * t) .* ld;

    case 'jpeg9.7'
        ld = [0 0.02674875741080976 -0.01686411844287495 ...
              -0.07822326652898785 0.2668641184428723 ...
              0.6029490182363579 0.2668641184428723 ...
              -0.07822326652898785 -0.01686411844287495 ...
              0.02674875741080976];
        hd = [0 -0.09127176311424948 0.05754352622849957 ...
              0.5912717631142470 -1.115087052456994 ...
              0.5912717631142470 0.05754352622849957 ...
              -0.09127176311424948 0 0];
        t = (0:9);
        lr = cos(pi * (t + 1)) .* hd;
        hr = cos(pi * t) .* ld;

    otherwise
        error('Unrecognizable wavelet name (WNAME).');
    end
```

```
% Output the requested filters.
if (nargin == 1)
   varargout(1:4) = {ld, hd, lr, hr};
else
   switch lower(type(1))
   case 'd'
      varargout = {ld, hd};
   case 'r'
      varargout = {lr, hr};
   otherwise
      error('Unrecognizable filter TYPE.');
   end
end
```

Note that for each orthonormal filter in `wavefilter` (i.e., `'haar'`, `'db4'`, and `'sym4'`), the reconstruction filters are time-reversed versions of the decomposition filters and the highpass decomposition filter is a modulated version of its lowpass counterpart. Only the lowpass decomposition filter coefficients need to be explicitly enumerated in the code. The remaining filter coefficients can be computed from them. In `wavefilter`, time reversal is carried out by reordering filter vector elements from last to first with statements like `lr(end:-1:1) = ld`. Modulation is accomplished by multiplying the components of a known filter by `cos(pi*t)`, which alternates between 1 and -1 as `t` increases from 0 in integer steps. For each biorthogonal filter in `wavefilter` (i.e., `'bior6.8'` and `'jpeg9.7'`), both the lowpass and highpass decomposition filters are specified; the reconstruction filters are computed as modulations of them. Finally, we note that the filters generated by `wavefilter` are of even length. Moreover, zero padding is used to ensure that the lengths of the decomposition and reconstruction filters of each wavelet are identical.

Given a pair of `wavefilter` generated decomposition filters, it is easy to write a general-purpose routine for the computation of the related fast wavelet transform. The goal is to devise an efficient algorithm based on the filtering and downsampling operations in Fig. 7.2. To maintain compatibility with the existing Wavelet Toolbox, we employ the same decomposition structure (i.e., `[C, S]` where `C` is a decomposition vector and `S` is a bookkeeping matrix). The following routine, which we call `wavefast`, uses symmetric image extension to reduce the border distortion associated with the computed FWT:

wavefast

```
function [c, s] = wavefast(x, n, varargin)
%WAVEFAST Perform multi-level 2-dimensional fast wavelet transform.
%   [C, L] = WAVEFAST(X, N, LP, HP) performs a 2D N-level FWT of
%   image (or matrix) X with respect to decomposition filters LP and
%   HP.
%
%   [C, L] = WAVEFAST(X, N, WNAME) performs the same operation but
%   fetches filters LP and HP for wavelet WNAME using WAVEFILTER.
%
%   Scale parameter N must be less than or equal to log2 of the
%   maximum image dimension. Filters LP and HP must be even. To
```

```
%   reduce border distortion, X is symmetrically extended. That is,
%   if X = [c1 c2 c3 ... cn] (in 1D), then its symmetric extension
%   would be [... c3 c2 c1 c1 c2 c3 ... cn cn cn–1 cn–2 ...].
%
%   OUTPUTS:
%     Matrix C is a coefficient decomposition vector:
%
%      C = [ a(n) h(n) v(n) d(n) h(n–1)  ... v(1) d(1) ]
%
%     where a, h, v, and d are columnwise vectors containing
%     approximation, horizontal, vertical, and diagonal coefficient
%     matrices, respectively. C has 3n + 1 sections where n is the
%     number of wavelet decompositions.
%
%     Matrix S is an (n+2) x 2 bookkeeping matrix:
%
%      S = [ sa(n, :); sd(n, :); sd(n–1, :); ...; sd(1, :); sx ]
%
%     where sa and sd are approximation and detail size entries.
%
%   See also WAVEBACK and WAVEFILTER.

%   Check the input arguments for reasonableness.
error(nargchk(3, 4, nargin));

if nargin == 3
   if ischar(varargin{1})
      [lp, hp] = wavefilter(varargin{1}, 'd');
   else
      error('Missing wavelet name.');
   end
else
      lp = varargin{1};      hp = varargin{2};
end

fl = length(lp);       sx = size(x);

if (ndims(x) ~= 2) | (min(sx) < 2) | ~isreal(x) | ~isnumeric(x)
   error('X must be a real, numeric matrix.');
end

if (ndims(lp) ~= 2) | ~isreal(lp) | ~isnumeric(lp) ...
       | (ndims(hp) ~= 2) | ~isreal(hp) | ~isnumeric(hp) ...
       | (fl ~= length(hp)) | rem(fl, 2) ~= 0
   error(['LP and HP must be even and equal length real, ' ...
          'numeric filter vectors.']);
end

if ~isreal(n) | ~isnumeric(n) | (n < 1) | (n > log2(max(sx)))
   error(['N must be a real scalar between 1 and ' ...
          'log2(max(size((X))).']);
end
```

`rem (X, Y)` *returns the remainder of the division of* X *by* Y.

```
% Init the starting output data structures and initial approximation.
c = [];     s = sx;     app = double(x);

% For each decomposition ...
for i = 1:n
   % Extend the approximation symmetrically.
   [app, keep] = symextend(app, fl);

   % Convolve rows with HP and downsample. Then convolve columns
   % with HP and LP to get the diagonal and vertical coefficients.
   rows = symconv(app, hp, 'row', fl, keep);
   coefs = symconv(rows, hp, 'col', fl, keep);
   c = [coefs(:)' c];    s = [size(coefs); s];
   coefs = symconv(rows, lp, 'col', fl, keep);
   c = [coefs(:)' c];

   % Convolve rows with LP and downsample. Then convolve columns
   % with HP and LP to get the horizontal and next approximation
   % coefficients.
   rows = symconv(app, lp, 'row', fl, keep);
   coefs = symconv(rows, hp, 'col', fl, keep);
   c = [coefs(:)' c];
   app = symconv(rows, lp, 'col', fl, keep);
end

% Append final approximation structures.
c = [app(:)' c];     s = [size(app); s];

%--------------------------------------------------------------------%
function [y, keep] = symextend(x, fl)
% Compute the number of coefficients to keep after convolution
% and downsampling. Then extend x in both dimensions.

keep = floor((fl + size(x) - 1) / 2);
y = padarray(x, [(fl - 1) (fl - 1)], 'symmetric', 'both');

%--------------------------------------------------------------------%
function y = symconv(x, h, type, fl, keep)
% Convolve the rows or columns of x with h, downsample,
% and extract the center section since symmetrically extended.

if strcmp(type, 'row')
   y = conv2(x, h);
   y = y(:, 1:2:end);
   y = y(:, fl / 2 + 1:fl / 2 + keep(2));
else
   y = conv2(x, h');
   y = y(1:2:end, :);
   y = y(fl / 2 + 1:fl / 2 + keep(1), :);
end
```

C = conv2 (A, B) *performs the 2-D convolution of matrices* A *and* B.

As can be seen in the main routine, only one `for` loop, which cycles through the decomposition levels (or scales) that are generated, is used to orchestrate the entire forward transform computation. For each execution of the loop, the current approximation image, `app`, which is initially set to `x`, is symmetrically extended by internal function `symextend`. This function calls `padarray`, which was introduced in Section 3.4.2, to extend `app` in two dimensions by mirror reflecting `f1 – 1` of its elements (the length of the decomposition filter minus 1) across its border.

Function `symextend` returns an extended matrix of approximation coefficients and the number of pixels that should be extracted from the center of any subsequently convolved and downsampled results. The rows of the extended approximation are next convolved with highpass decomposition filter `hp` and downsampled via `symconv`. This function is described in the following paragraph. Convolved output, `rows`, is then submitted to `symconv` to convolve and downsample its columns with filters `hp` and `lp`—generating the diagonal and vertical detail coefficients of the top two branches of Fig. 7.2. These results are inserted into decomposition vector `c` (working from the last element toward the first) and the process is repeated in accordance with Fig. 7.2 to generate the horizontal detail and approximation coefficients (the bottom two branches of the figure).

Function `symconv` uses the `conv2` function to do the bulk of the transform computation work. It convolves filter `h` with the rows or columns of `x` (depending on `type`), discards the even indexed rows or columns (i.e., downsamples by 2), and extracts the center `keep` elements of each row or column. Invoking `conv2` with matrix `x` and row filter vector `h` initiates a row-by-row convolution; using column filter vector `h'` results in a columnwise convolution.

EXAMPLE 7.3: Comparing the execution times of `wavefast` and `wavedec2`.

■ The following test routine uses functions `tic` and `toc` to compare the execution times of the Wavelet Toolbox function `wavedec2` and custom function `wavefast`:

```
function [ratio, maxdiff] = fwtcompare(f, n, wname)
%FWTCOMPARE Compare wavedec2 and wavefast.
%   [RATIO, MAXDIFF] = FWTCOMPARE(F, N, WNAME) compares the operation
%   of toolbox function WAVEDEC2 and custom function WAVEFAST.
%
%   INPUTS:
%     F          Image to be transformed.
%     N          Number of scales to compute.
%     WNAME      Wavelet to use.
%
%   OUTPUTS:
%     RATIO      Execution time ratio (custom/toolbox)
%     MAXDIFF    Maximum coefficient difference.

% Get transform and computation time for wavedec2.
tic;
[c1, s1] = wavedec2(f, n, wname);
reftime = toc;
```

FIGURE 7.4
A 512 × 512 image of a vase.

```
% Get transform and computation time for wavefast.
tic;
[c2, s2] = wavefast(f, n, wname);
t2 = toc;

% Compare the results.
ratio = t2 / (reftime + eps);
maxdiff = abs(max(c1 - c2));
```

For the 512 × 512 image of Fig. 7.4 and a five-scale wavelet transform with respect to 4th order Daubechies' wavelets, `fwtcompare` yields

```
>> f = imread('Vase', 'tif');
>> [ratio, maxdifference] = fwtcompare(f, 5, 'db4')

ratio =
    0.5508

maxdifference =
    3.2969e-012
```

Note that custom function `wavefast` was almost twice as fast as its Wavelet Toolbox counterpart while producing virtually identical results. ■

7.3 Working with Wavelet Decomposition Structures

The wavelet transformation functions of the previous two sections produce *nondisplayable* data structures of the form $\{\mathbf{c}, \mathbf{S}\}$, where $\mathbf{c}$ is a transform coefficient vector and $\mathbf{S}$ is a bookkeeping matrix that defines the arrangement of coefficients in $\mathbf{c}$. To process images, we must be able to examine and/or modify $\mathbf{c}$. In this section, we formally define $\{\mathbf{c}, \mathbf{S}\}$, examine some of the Wavelet Toolbox functions for manipulating it, and develop a set of custom functions that can be used without the Wavelet Toolbox. These functions are then used to build a general purpose routine for displaying $\mathbf{c}$.

The representation scheme introduced in Example 7.2 integrates the coefficients of a multiscale two-dimensional wavelet transform into a single, one-dimensional vector

$$\mathbf{c} = [\mathbf{A}_N(:)' \quad \mathbf{H}_N(:)' \quad \cdots \quad \mathbf{H}_i(:)' \quad \mathbf{V}_i(:)' \quad \mathbf{D}_i(:)' \quad \cdots \quad \mathbf{V}_1(:)' \quad \mathbf{D}_1(:)']$$

where $\mathbf{A}_N$ is the approximation coefficient matrix of the Nth decomposition level and $\mathbf{H}_i$, $\mathbf{V}_i$, and $\mathbf{D}_i$ for $i = 1, 2, \ldots N$ are the horizontal, vertical, and diagonal transform coefficient matrices for level i. Here, $\mathbf{H}_i(:)'$, for example, is the row vector formed by concatenating the transposed columns of matrix $\mathbf{H}_i$. That is, if

$$\mathbf{H}_i = \begin{bmatrix} 3 & -2 \\ 1 & 6 \end{bmatrix}$$

then

$$\mathbf{H}_i(:) = \begin{bmatrix} 3 \\ 1 \\ -2 \\ 6 \end{bmatrix} \quad \text{and} \quad \mathbf{H}_i(:)' = [3 \quad 1 \quad -2 \quad 6]$$

Because the equation for **c** assumes N decompositions (or passes through the filter bank in Fig. 7.2), **c** contains $3N + 1$ sections—one approximation and N groups of horizontal, vertical, and diagonal details. Note that the highest scale coefficients are computed when $i = 1$; the lowest scale coefficients are associated with $i = N$. Thus, the coefficients of **c** are ordered from low to high scale.

Matrix **S** of the decomposition structure is an $(N + 2) \times 2$ bookkeeping array of the form

$$\mathbf{S} = [\mathbf{sa}_N; \quad \mathbf{sd}_N; \quad \mathbf{sd}_{N-1}; \quad \cdots \quad \mathbf{sd}_i; \quad \cdots \quad \mathbf{sd}_1; \quad \mathbf{sf}]$$

where $\mathbf{sa}_N$, $\mathbf{sd}_i$, and **sf** are 1×2 vectors containing the horizontal and vertical dimensions of Nth-level approximation $\mathbf{A}_N$, ith-level details ($\mathbf{H}_i$, $\mathbf{V}_i$, and $\mathbf{D}_i$ for $i = 1, 2, \ldots N$), and original image **F**, respectively. The information in **S** can be used to locate the individual approximation and detail coefficients in **c**. Note that the semicolons in the preceding equation indicate that the elements of **S** are organized as a column vector.

EXAMPLE 7.4: Wavelet Toolbox functions for manipulating transform decomposition vector **c**.

■ The Wavelet Toolbox provides a variety of functions for locating, extracting, reformatting, and/or manipulating the approximation and horizontal, vertical, and diagonal coefficients of **c** as a function of decomposition level. We introduce them here to illustrate the concepts just discussed and to prepare the way for the alternative functions that will be developed in the next section. Consider, for example, the following sequence of commands:

```
>> f = magic(8);
>> [c1, s1] = wavedec2(f, 3, 'haar');
>> size(c1)
```

```
ans =
    1    64
>> s1
s1 =
    1     1
    1     1
    2     2
    4     4
    8     8

>> approx = appcoef2(c1, s1, 'haar')
approx =
    260.0000

>> horizdet2 = detcoef2('h', c1, s1, 2)
horizdet2 =
    1.0e-013 *

         0   -0.2842
         0         0

>> newc1 = wthcoef2('h', c1, s1, 2);
>> newhorizdet2 = detcoef2('h', newc1, s1, 2)
newhorizdet2 =
    0     0
    0     0
```

Here, a three-level decomposition with respect to Haar wavelets is performed on an 8×8 magic square using the `wavedec2` function. The resulting coefficient vector, `c1`, is of size 1×64. Since `s1` is 5×2, we know that the coefficients of `c1` span $(N - 2) = (5 - 2) = 3$ decomposition levels. Thus, it concatenates the elements needed to populate $3N + 1 = 3(3) + 1 = 10$ approximation and detail coefficient submatrices. Based on `s1`, these submatrices include (a) a 1×1 approximation matrix and three 1×1 detail matrices for decomposition level 3 [see `s1(1, :)` and `s1(2, :)`], (b) three 2×2 detail matrices for level 2 [see `s1(3, :)`], and (c) three 4×4 detail matrices for level 1 [see `s1(4, :)`]. The fifth row of `s1` contains the size of the original image `f`.

Matrix `approx` = 260 is extracted from `c1` using toolbox function `appcoef2`, which has the following syntax:

```
a = appcoef2(c, s, wname)
```

Here, `wname` is a wavelet name from Table 7.1 and `a` is the returned approximation matrix. The horizontal detail coefficients at level 2 are retrieved using `detcoef2`, a function of similar syntax

```
d = detcoef2(o, c, s, n)
```

in which `o` is set to `'h'`, `'v'`, or `'d'` for the horizontal, vertical, and diagonal details and `n` is the desired decomposition level. In this example, 2×2 matrix

`horizdet2` is returned. The coefficients corresponding to `horizdet2` in `c1` are then zeroed using `wthcoef2`, a wavelet thresholding function of the form


```
nc = wthcoef2(type, c, s, n, t, sorh)
```

where `type` is set to `'a'` to threshold approximation coefficients and `'h'`, `'v'`, or `'d'` to threshold horizontal, vertical, or diagonal details, respectively. Input `n` is a vector of decomposition levels to be thresholded based on the corresponding thresholds in vector `t`, while `sorh` is set to `'s'` or `'h'` for soft or hard thresholding, respectively. If `t` is omitted, all coefficients meeting the `type` and `n` specifications are zeroed. Output `nc` is the modified (i.e., thresholded) decomposition vector. All three of the preceding Wavelet Toolbox functions have other syntaxes that can be examined using the MATLAB `help` command. ■

7.3.1 Editing Wavelet Decomposition Coefficients without the Wavelet Toolbox

Without the Wavelet Toolbox, bookkeeping matrix **S** is the key to accessing the individual approximation and detail coefficients of multiscale vector **c**. In this section, we use **S** to build a set of general-purpose routines for the manipulation of **c**. Function `wavework` is the foundation of the routines developed, which are based on the familiar cut-copy-paste metaphor of modern word processing applications.

wavework

```
function [varargout] = wavework(opcode, type, c, s, n, x)
%WAVEWORK is used to edit wavelet decomposition structures.
%   [VARARGOUT] = WAVEWORK(OPCODE, TYPE, C, S, N, X) gets the
%   coefficients specified by TYPE and N for access or modification
%   based on OPCODE.
%
%   INPUTS:
%     OPCODE      Operation to perform
%     ------------------------------------------------------------
%     'copy'      [varargout] = Y = requested (via TYPE and N)
%                 coefficient matrix
%     'cut'       [varargout] = [NC, Y] = New decomposition vector
%                 (with requested coefficient matrix zeroed) AND
%                 requested coefficient matrix
%     'paste'     [varargout] = [NC] = new decomposition vector with
%                 coefficient matrix replaced by X
%
%     TYPE        Coefficient category
%     ------------------------------------------------------------
%     'a'         Approximation coefficients
%     'h'         Horizontal details
%     'v'         Vertical details
%     'd'         Diagonal details
%
%     [C, S] is a wavelet toolbox decomposition structure.
```

```
%     N is a decomposition level (Ignored if TYPE = 'a').
%     X is a two-dimensional coefficient matrix for pasting.
%
%   See also WAVECUT, WAVECOPY, and WAVEPASTE.

error(nargchk(4, 6, nargin));

if (ndims(c) ~= 2) | (size(c, 1) ~= 1)
   error('C must be a row vector.');
end

if (ndims(s) ~= 2) | ~isreal(s) | ~isnumeric(s) | (size(s, 2) ~= 2)
   error('S must be a real, numeric two-column array.');
end

elements = prod(s, 2);           % Coefficient matrix elements.
if (length(c) < elements(end)) | ...
      ~(elements(1) + 3 * sum(elements(2:end - 1)) >= elements(end))
   error(['[C S] must form a standard wavelet decomposition ' ...
          'structure.']);
end

if strcmp(lower(opcode(1:3)), 'pas') & nargin < 6
   error('Not enough input arguments.');
end

if nargin < 5
   n = 1;         % Default level is 1.
end
nmax = size(s, 1) - 2;          % Maximum levels in [C, S].

aflag = (lower(type(1)) == 'a');
if ~aflag & (n > nmax)
   error('N exceeds the decompositions in [C, S].');
end

switch lower(type(1))           % Make pointers into C.
case 'a'
   nindex = 1;
   start = 1;    stop = elements(1);    ntst = nmax;
case {'h', 'v', 'd'}
   switch type
   case 'h', offset = 0;      % Offset to details.
   case 'v', offset = 1;
   case 'd', offset = 2;
   end
   nindex = size(s, 1) - n;      % Index to detail info.
   start = elements(1) + 3 * sum(elements(2:nmax - n + 1)) + ...
           offset * elements(nindex) + 1;
   stop = start + elements(nindex) - 1;
   ntst = n;
otherwise
   error('TYPE must begin with "a", "h", "v", or "d".');
end
```

```
switch lower(opcode)          % Do requested action.
case {'copy', 'cut'}
   y = repmat(0, s(nindex, :));
   y(:) = c(start:stop);          nc = c;
   if strcmp(lower(opcode(1:3)), 'cut')
      nc(start:stop) = 0; varargout = {nc, y};
   else
       varargout = {y};
   end
case 'paste'
   if prod(size(x)) ~= elements(end - ntst)
      error('X is not sized for the requested paste.');
   else
      nc = c;   nc(start:stop) = x(:);     varargout = {nc};
   end
otherwise
   error('Unrecognized OPCODE.');
end
```

As wavework checks its input arguments for reasonableness, the number of elements in each coefficient submatrix of c is computed via elements = prod(s, 2). Recall from Section 3.4.2 that MATLAB function Y = prod(X, DIM) computes the products of the elements of X along dimension DIM. The first switch statement then begins the computation of a pair of pointers to the coefficients associated with input parameters type and n. For the approximation case (i.e., case 'a'), the computation is trivial since the coefficients are always at the start of c (so pointer start is 1); the ending index, pointer stop, is the number of elements in the approximation matrix, which is elements(1). When a detail coefficient submatrix is requested, however, start is computed by summing the number of elements at all decomposition levels above n and adding offset * elements(nindex); where offset is 0, 1, or 2 for the horizontal, vertical, or diagonal coefficients, respectively, and nindex is a pointer to the row of s that corresponds to input parameter n.

The second switch statement in function wavework performs the operation requested by opcode. For the 'cut' and 'copy' cases, the coefficients of c between start and stop are copied into y, which has been preallocated as a two-dimensional matrix whose size is determined by s. This is done using y = repmat(0, s(nindex, :)), in which MATLAB's "replicate matrix" function, B = repmat(A, M, N), is used to create a large matrix B composed of M x N tiled copies of A. For the 'paste' case, the elements of x are copied into nc, a copy of input c, between start and stop. For both the 'cut' and 'paste' operations, a new decomposition vector nc is returned.

The following three functions—wavecut, wavecopy, and wavepaste—use wavework to manipulate c using a more intuitive syntax:

wavecut

```
function [nc, y] = wavecut(type, c, s, n)
%WAVECUT Zeroes coefficients in a wavelet decomposition structure.
%   [NC, Y] = WAVECUT(TYPE, C, S, N) returns a new decomposition
%   vector whose detail or approximation coefficients (based on TYPE
```

```
%   and N) have been zeroed. The coefficients that were zeroed are
%   returned in Y.
%
%   INPUTS:
%     TYPE       Coefficient category
%     ------------------------------------------------------------
%     'a'        Approximation coefficients
%     'h'        Horizontal details
%     'v'        Vertical details
%     'd'        Diagonal details
%
%     [C, S] is a wavelet data structure.
%     N specifies a decomposition level (ignored if TYPE = 'a').
%
%   See also WAVEWORK, WAVECOPY, and WAVEPASTE.

error(nargchk(3, 4, nargin));
if nargin == 4
   [nc, y] = wavework('cut', type, c, s, n);
else
   [nc, y] = wavework('cut', type, c, s);
end
```

wavecopy

```
function y = wavecopy(type, c, s, n)
%WAVECOPY Fetches coefficients of a wavelet decomposition structure.
%   Y = WAVECOPY(TYPE, C, S, N) returns a coefficient array based on
%   TYPE and N.
%
%   INPUTS:
%     TYPE       Coefficient category
%     ------------------------------------------------------------
%     'a'        Approximation coefficients
%     'h'        Horizontal details
%     'v'        Vertical details
%     'd'        Diagonal details
%
%   [C, S] is a wavelet data structure.
%   N specifies a decomposition level (ignored if TYPE = 'a').
%
% See also WAVEWORK, WAVECUT, and WAVEPASTE.

error(nargchk(3, 4, nargin));
if nargin == 4
    y = wavework('copy', type, c, s, n);
else
    y = wavework('copy', type, c, s);
end
```

wavepaste

```
function nc = wavepaste(type, c, s, n, x)
%WAVEPASTE Puts coefficients in a wavelet decomposition structure.
%   NC = WAVEPASTE(TYPE, C, S, N, X) returns the new decomposition
%   structure after pasting X into it based on TYPE and N.
```

```
%
%   INPUTS:
%     TYPE      Coefficient category
%     ------------------------------------------------------------
%     'a'       Approximation coefficients
%     'h'       Horizontal details
%     'v'       Vertical details
%     'd'       Diagonal details
%
%     [C, S] is a wavelet data structure.
%     N specifies a decomposition level (Ignored if TYPE = 'a').
%     X is a two-dimensional approximation or detail coefficient
%       matrix whose dimensions are appropriate for decomposition
%       level N.
%
%   See also WAVEWORK, WAVECUT, and WAVECOPY.

error(nargchk(5, 5, nargin))
nc = wavework('paste', type, c, s, n, x);
```

EXAMPLE 7.5: Manipulating **c** with wavecut and wavecopy.

■ Functions wavecopy and wavecut can be used to reproduce the Wavelet Toolbox based results of Example 7.4:

```
>> f = magic(8);
>> [c1, s1] = wavedec2(f, 3, 'haar');
>> approx = wavecopy('a', c1, s1)
approx =
    260.0000

>> horizdet2 = wavecopy('h', c1, s1, 2)
horizdet2 =
    1.0e-013 *

         0   -0.2842
         0         0

>> [newc1, horizdet2] = wavecut('h', c1, s1, 2);
>> newhorizdet2 = wavecopy('h', newc1, s1, 2)
newhorizdet2 =
    0    0
    0    0
```

Note that all extracted matrices are identical to those of the previous example. ■

7.3.2 Displaying Wavelet Decomposition Coefficients

As was indicated at the start of Section 7.3, the coefficients that are packed into one-dimensional wavelet decomposition vector **c** are, in reality, the coefficients of the two-dimensional output arrays from the filter bank in Fig. 7.2. For each iteration of the filter bank, four quarter-size coefficient arrays (neglecting any expansion that may result from the convolution process) are produced.

They can be arranged as a 2 × 2 array of submatrices that replace the two-dimensional input from which they are derived. Function wave2gray performs this subimage compositing—and both scales the coefficients to better reveal their differences and inserts borders to delineate the approximation and various horizontal, vertical, and diagonal detail matrices.

wave2gray

```
function w = wave2gray(c, s, scale, border)
%WAVE2GRAY Display wavelet decomposition coefficients.
%   W = WAVE2GRAY(C, S, SCALE, BORDER) displays and returns a
%   wavelet coefficient image.
%
%   EXAMPLES:
%     wave2gray(c, s);                      Display w/defaults.
%     foo = wave2gray(c, s);                Display and return.
%     foo = wave2gray(c, s, 4);             Magnify the details.
%     foo = wave2gray(c, s, -4);            Magnify absolute values.
%     foo = wave2gray(c, s, 1, 'append');   Keep border values.
%
%   INPUTS/OUTPUTS:
%     [C, S] is a wavelet decomposition vector and bookkeeping
%     matrix.
%
%     SCALE         Detail coefficient scaling
%     ------------------------------------------------------------
%     0 or 1        Maximum range (default)
%     2, 3...       Magnify default by the scale factor
%     -1, -2...     Magnify absolute values by abs(scale)
%
%     BORDER        Border between wavelet decompositions
%     ------------------------------------------------------------
%     'absorb'      Border replaces image (default)
%     'append'      Border increases width of image
%
%     Image W:   ------- ------ ------------- ------------------
%                |       |      |             |
%                | a(n)  | h(n) |             |
%                |       |      |             |
%                ------- ------      h(n-1)   |
%                |       |      |             |
%                | v(n)  | d(n) |             |        h(n-2)
%                |       |      |             |
%                ------- ------ -------------
%                |              |             |
%                |    v(n-1)    |    d(n-1)   |
%                |              |             |
%                -------------- -------------- ------------------
%                |                            |
%                |          v(n-2)            |        d(n-2)
%                |                            |
%
```

```
%     Here, n denotes the decomposition step scale and a, h, v, d are
%     approximation, horizontal, vertical, and diagonal detail
%     coefficients, respectively.

% Check input arguments for reasonableness.
error(nargchk(2, 4, nargin));

if (ndims(c) ~= 2) | (size(c, 1) ~= 1)
   error('C must be a row vector.'); end

if (ndims(s) ~= 2) | ~isreal(s) | ~isnumeric(s) | (size(s, 2) ~= 2)
   error('S must be a real, numeric two-column array.'); end

elements = prod(s, 2);
if (length(c) < elements(end)) | ...
      ~(elements(1) + 3 * sum(elements(2:end - 1)) >= elements(end))
   error(['[C S] must be a standard wavelet ' ...
         'decomposition structure.']);
end

if (nargin > 2) & (~isreal(scale) | ~isnumeric(scale))
   error('SCALE must be a real, numeric scalar.');
end

if (nargin > 3) & (~ischar(border))
   error('BORDER must be character string.');
end

if nargin == 2
   scale = 1;  % Default scale.
end

if nargin < 4
   border = 'absorb';  % Default border.
end

% Scale coefficients and determine pad fill.
absflag = scale < 0;
scale = abs(scale);
if scale == 0
   scale = 1;
end

[cd, w] = wavecut('a', c, s);   w = mat2gray(w);
cdx = max(abs(cd(:))) / scale;
if absflag
   cd = mat2gray(abs(cd), [0, cdx]); fill = 0;
else
   cd = mat2gray(cd, [-cdx, cdx]); fill = 0.5;
end

% Build gray image one decomposition at a time.
for i = size(s, 1) - 2:-1:1
   ws = size(w);

   h = wavecopy('h', cd, s, i);
   pad = ws - size(h);     frontporch = round(pad / 2);
   h = padarray(h, frontporch, fill, 'pre');
   h = padarray(h, pad - frontporch, fill, 'post');
```

```
    v = wavecopy('v', cd, s, i);
    pad = ws - size(v);      frontporch = round(pad / 2);
    v = padarray(v, frontporch, fill, 'pre');
    v = padarray(v, pad - frontporch, fill, 'post');
    d = wavecopy('d', cd, s, i);
    pad = ws - size(d);      frontporch = round(pad / 2);
    d = padarray(d, frontporch, fill, 'pre');
    d = padarray(d, pad - frontporch, fill, 'post');
    % Add 1 pixel white border.
    switch lower(border)
    case 'append'
       w = padarray(w, [1 1], 1, 'post');
       h = padarray(h, [1 0], 1, 'post');
       v = padarray(v, [0 1], 1, 'post');
    case 'absorb'
       w(:, end) = 1;     w(end, :) = 1;
       h(end, :) = 1;     v(:, end) = 1;
    otherwise
       error('Unrecognized BORDER parameter.');
    end
    w = [w h; v d];                    % Concatenate coefs.
end

if nargout == 0
    imshow(w);                         % Display result.
end
```

The "help text" or header section of `wave2gray` details the structure of generated output image `w`. The subimage in the upper left corner of `w`, for instance, is the approximation array that results from the final decomposition step. It is surrounded—in a clockwise manner—by the horizontal, diagonal, and vertical detail coefficients that were generated during the same decomposition. The resulting 2×2 array of subimages is then surrounded (again in a clockwise manner) by the detail coefficients of the previous decomposition step; and the pattern continues until all of the scales of decomposition vector `c` are appended to two-dimensional matrix `w`.

The compositing just described takes place within the only `for` loop in `wave2gray`. After checking the inputs for consistency, `wavecut` is called to remove the approximation coefficients from decomposition vector `c`. These coefficients are then scaled for later display using `mat2gray`. Modified decomposition vector `cd` (i.e., `c` without the approximation coefficients) is then similarly scaled. For positive values of input `scale`, the detail coefficients are scaled so that a coefficient value of 0 appears as middle gray; all necessary padding is performed with a `fill` value of 0.5 (mid-gray). If `scale` is negative, the absolute values of the detail coefficients are displayed with a value of 0 corresponding to black and the pad `fill` value is set to 0. After the approximation and detail coefficients have been scaled for display, the first iteration of the `for` loop extracts the last decomposition step's detail coefficients from `cd` and appends them to `w` (after padding to make the dimensions of the four subimages match and insertion of a

one-pixel white border) via the `w = [w h; v d]` statement. This process is then repeated for each scale in `c`. Note the use of `wavecopy` to extract the various detail coefficients needed to form `w`.

EXAMPLE 7.6: Transform coefficient display using `wave2gray`.

■ The following sequence of commands computes the two-scale DWT of the image in Fig. 7.4 with respect to fourth-order Daubechies' wavelets and displays the resulting coefficients:

```
>> f = imread('vase.tif');
>> [c, s] = wavefast(f, 2, 'db4');
>> wave2gray(c, s);
>> figure; wave2gray(c, s, 8);
>> figure; wave2gray(c, s, -8);
```

The images generated by the final three command lines are shown in Figs. 7.5(a) through (c), respectively. Without additional scaling, the detail coefficient differences in Fig. 7.5(a) are barely visible. In Fig. 7.5(b), the differences are accentuated by multiplying them by 8. Note the mid-gray

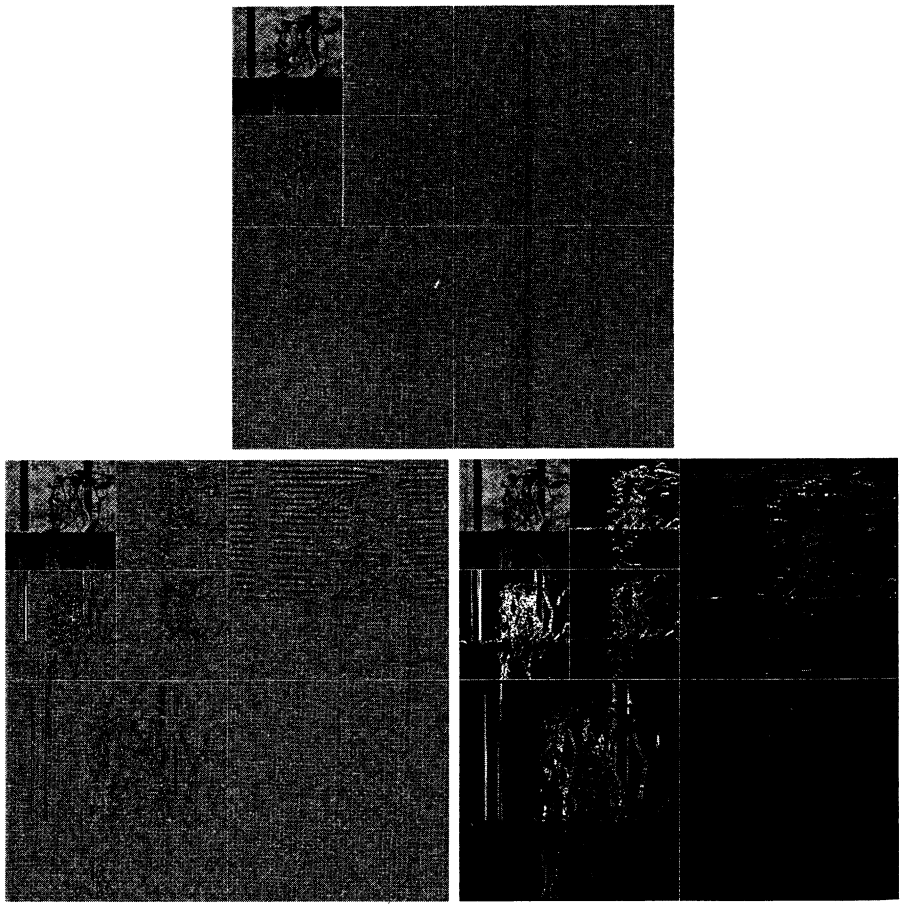


FIGURE 7.5 Displaying a two-scale wavelet transform of the image in Fig. 7.4: (a) Automatic scaling; (b) additional scaling by 8; and (c) absolute values scaled by 8.

padding along the borders of the level 1 coefficient subimages; it was inserted to reconcile dimensional variations between transform coefficient subimages. Figure 7.5(c) shows the effect of taking the absolute values of the details. Here, all padding is done in black. ■

7.4 The Inverse Fast Wavelet Transform

Like its forward counterpart, the *inverse fast wavelet transform* can be computed iteratively using digital filters. Figure 7.6 shows the required *synthesis* or *reconstruction filter bank*, which reverses the process of the analysis or decomposition filter bank of Fig. 7.2. At each iteration, four scale j approximation and detail subimages are *upsampled* (by inserting zeroes between every element) and convolved with two one-dimension filters—one operating on the subimages' columns and the other on its rows. Addition of the results yields the scale $j + 1$ approximation, and the process is repeated until the original image is reconstructed. The filters used in the convolutions are a function of the wavelets employed in the forward transform. Recall that they can be obtained from the `wfilters` and `wavefilter` functions of Section 7.2 with input parameter `type` set to `'r'` for "reconstruction."

When using the Wavelet Toolbox, function `waverec2` is employed to compute the inverse FWT of wavelet decomposition structure `[C, S]`. It is invoked using

```
g = waverec2(C, S, wname)
```

where `g` is the resulting reconstructed two-dimensional image (of class `double`). The required reconstruction filters can be alternately supplied via syntax

```
g = waverec2(C, S, Lo_R, Hi_R)
```

The following custom routine, which we call `waveback`, can be used when the Wavelet Toolbox is unavailable. It is the final function needed to complete our wavelet-based package for processing images in conjunction with IPT (and without the Wavelet Toolbox).

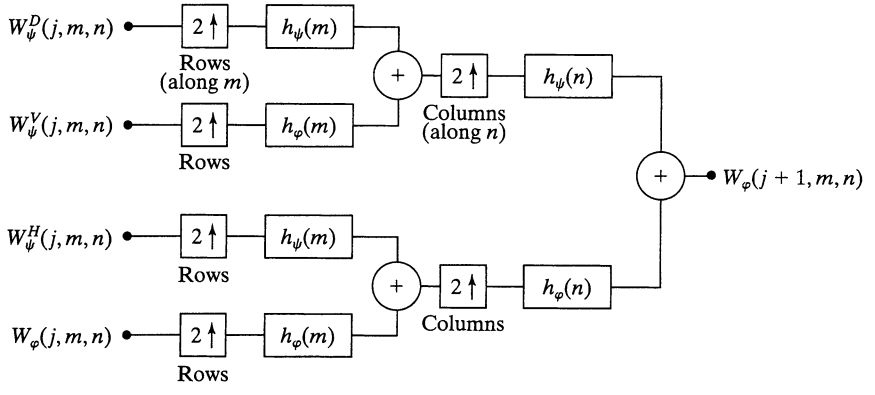


FIGURE 7.6 The 2-D FWT^{-1} filter bank. The boxes with the up arrows represent upsampling by inserting zeroes between every element.

waveback

```
function [varargout] = waveback(c, s, varargin)
%WAVEBACK Performs a multi-level two-dimensional inverse FWT.
%   [VARARGOUT] = WAVEBACK(C, S, VARARGIN) computes a 2D N-level
%   partial or complete wavelet reconstruction of decomposition
%   structure [C, S].
%
%   SYNTAX:
%     Y = WAVEBACK(C, S, 'WNAME');     Output inverse FWT matrix Y
%     Y = WAVEBACK(C, S, LR, HR);      using lowpass and highpass
%                                      reconstruction filters (LR and
%                                      HR) or filters obtained by
%                                      calling WAVEFILTER with 'WNAME'.
%
%     [NC, NS] = WAVEBACK(C, S, 'WNAME', N);   Output new wavelet
%     [NC, NS] = WAVEBACK(C, S, LR, HR, N);    decomposition structure
%                                              [NC, NS] after N step
%                                              reconstruction.
%
%   See also WAVEFAST and WAVEFILTER.

% Check the input and output arguments for reasonableness.
error(nargchk(3, 5, nargin));
error(nargchk(1, 2, nargout));

if (ndims(c) ~= 2) | (size(c, 1) ~= 1)
   error('C must be a row vector.');
end

if (ndims(s) ~= 2) | ~isreal(s) | ~isnumeric(s) | (size(s, 2) ~= 2)
   error('S must be a real, numeric two-column array.');
end

elements = prod(s, 2);
if (length(c) < elements(end)) | ...
      ~(elements(1) + 3 * sum(elements(2:end - 1)) >= elements(end))
   error(['[C S] must be a standard wavelet ' ...
         'decomposition structure.']);
end

% Maximum levels in [C, S].
nmax = size(s, 1) - 2;
% Get third input parameter and init check flags.
wname = varargin{1};   filterchk = 0;   nchk = 0;

switch nargin
case 3
   if ischar(wname)
      [lp, hp] = wavefilter(wname, 'r');   n = nmax;
   else
      error('Undefined filter.');
   end
   if nargout ~= 1
```

```
      error('Wrong number of output arguments.');
   end
case 4
   if ischar(wname)
      [lp, hp] = wavefilter(wname, 'r');
      n = varargin{2};  nchk = 1;
   else
      lp = varargin{1}; hp = varargin{2};
      filterchk = 1;    n = nmax;
      if nargout ~= 1
         error('Wrong number of output arguments.');
      end
   end
case 5
   lp = varargin{1};    hp = varargin{2};    filterchk = 1;
   n = varargin{3};     nchk = 1;
otherwise
   error('Improper number of input arguments.');
end

fl = length(lp);
if filterchk                                        % Check filters.
   if (ndims(lp) ~= 2) | ~isreal(lp) | ~isnumeric(lp) ...
         | (ndims(hp) ~= 2) | ~isreal(hp) | ~isnumeric(hp) ...
         | (fl ~= length(hp)) | rem(fl, 2) ~= 0
      error(['LP and HP must be even and equal length real, ' ...
            'numeric filter vectors.']);
   end
end

if nchk & (~isnumeric(n) | ~isreal(n))        % Check scale N.
   error('N must be a real numeric.');
end
if (n > nmax) | (n < 1)
    error('Invalid number (N) of reconstructions requested.');
end
if (n ~= nmax) & (nargout ~= 2)
    error('Not enough output arguments.');
end

nc = c;    ns = s;    nnmax = nmax;             % Init decomposition.
for i = 1:n
   % Compute a new approximation.
   a = symconvup(wavecopy('a', nc, ns), lp, lp, fl, ns(3, :)) + ...
       symconvup(wavecopy('h', nc, ns, nnmax), ...
                 hp, lp, fl, ns(3, :)) + ...
       symconvup(wavecopy('v', nc, ns, nnmax), ...
                 lp, hp, fl, ns(3, :)) + ...
       symconvup(wavecopy('d', nc, ns, nnmax), ...
                 hp, hp, fl, ns(3, :));
```

```
   % Update decomposition.
   nc = nc(4 * prod(ns(1, :)) + 1:end);      nc = [a(:)' nc];
   ns = ns(3:end, :);                        ns = [ns(1, :); ns];
   nnmax = size(ns, 1) - 2;
end

% For complete reconstructions, reformat output as 2-D.
if nargout == 1
   a = nc;    nc = repmat(0, ns(1, :));    nc(:) = a;
end

varargout{1} = nc;
if nargout == 2
   varargout{2} = ns;
end

%-------------------------------------------------------------------%
function z = symconvup(x, f1, f2, fln, keep)
% Upsample rows and convolve columns with f1; upsample columns and
% convolve rows with f2; then extract center assuming symmetrical
% extension.

y = zeros([2 1] .* size(x));      y(1:2:end, :) = x;
y = conv2(y, f1');
z = zeros([1 2] .* size(y));      z(:, 1:2:end) = y;
z = conv2(z, f2);
z = z(fln - 1:fln + keep(1) - 2, fln - 1:fln + keep(2) - 2);
```

The main routine of function `waveback` is a simple `for` loop that iterates through the requested number of decomposition levels (i.e., scales) in the desired reconstruction. As can be seen, each loop calls internal function `symconvup` four times and sums the returned matrices. Decomposition vector `nc`, which is initially set to `c`, is iteratively updated by replacing the four coefficient matrices passed to `symconvup` by the newly created approximation `a`. Bookkeeping matrix `ns` is then modified accordingly—there is now one less scale in decomposition structure `[nc, ns]`. This sequence of operations is slightly different than the ones outlined in Fig. 7.6, in which the top two inputs are combined to yield

$$[W_\psi^D(j, m, n)\uparrow^{2m} * h_\psi(m) + W_\psi^V(j, m, n)\uparrow^{2m} * h_\varphi(m)]\uparrow^{2n} * h_\psi(n)$$

where $\uparrow^{2m}$ and $\uparrow^{2n}$ denote upsampling along m and n, respectively. Function `waveback` uses the equivalent computation

$$[W_\psi^D(j, m, n)\uparrow^{2m} * h_\psi(m)]\uparrow^{2n} * h_\psi(n) + [W_\psi^V(j, m, n)\uparrow^{2m} * h_\varphi(m)]\uparrow^{2n} * h_\psi(n)$$

Function `symconvup` performs the convolutions and upsampling required to compute the contribution of one input of Fig. 7.6 to output $W_\varphi(j + 1, m, n)$ in accordance with the proceding equation. Input `x` is first upsampled in the row direction to yield `y`, which is convolved columnwise with filter `f1`. The resulting output, which replaces `y`, is then upsampled in the column direction and convolved row by row with `f2` to produce `z`. Finally, the center `keep` elements of `z` (the final convolution) are returned as input `x`'s contribution to the new approximation.

EXAMPLE 7.7: Comparing the execution times of waveback and waverec2.

■ The following test routine compares the execution times of Wavelet Toolbox function waverec2 and custom function waveback using a simple modification of the test function in Example 7.3:

```
function [ratio, maxdiff] = ifwtcompare(f, n, wname)
%IFWTCOMPARE Compare waverec2 and waveback.
%   [RATIO, MAXDIFF] = IFWTCOMPARE(F, N, WNAME) compares the
%   operation of Wavelet Toolbox function WAVEREC2 and custom function
%   WAVEBACK.
%
%   INPUTS:
%     F         Image to transform and inverse transform.
%     N         Number of scales to compute.
%     WNAME     Wavelet to use.
%
%   OUTPUTS:
%     RATIO     Execution time ratio (custom/toolbox).
%     MAXDIFF   Maximum generated image difference.

% Compute the transform and get output and computation time for
% waverec2.
[c1, s1] = wavedec2(f, n, wname);
tic;
g1 = waverec2(c1, s1, wname);
reftime = toc;

% Compute the transform and get output and computation time for
% waveback.
[c2, s2] = wavefast(f, n, wname);
tic;
g2 = waveback(c2, s2, wname);
t2 = toc;

% Compare the results.
ratio = t2 / (reftime + eps);
maxdiff = abs(max(max(g1 - g2)));
```

For a five scale transform of the 512 × 512 image in Fig. 7.4 with respect to 4th-order Daubechies' wavelets, we get

```
>> f = imread('Vase', 'tif');
>> [ratio, maxdifference] = ifwtcompare(f, 5, 'db4')

ratio =
    1.0000

maxdifference =
    3.6948e-013
```

Note that the inverse transformation times of the two functions are equivalent (i.e., the ratio is 1) and that the largest output difference is 3.6948×10^{-13}. For all practical purposes, they generate identical results in identical times. ■

7.5 Wavelets in Image Processing

As in the Fourier domain (see Section 4.3.2), the basic approach to wavelet-based image processing is to

1. Compute the two-dimensional wavelet transform of an image.
2. Alter the transform coefficients.
3. Compute the inverse transform.

Because scale in the wavelet domain is analogous to frequency in the Fourier domain, most of the Fourier-based filtering techniques of Chapter 4 have an equivalent "wavelet domain" counterpart. In this section, we use the preceding three-step procedure to give several examples of the use of wavelets in image processing. Attention is restricted to the routines developed earlier in the chapter; the Wavelet Toolbox is not needed to implement the examples given here—nor the examples in Chapter 7 of *Digital Image Processing* (Gonzalez and Woods [2002]).

EXAMPLE 7.8: Wavelet directionality and edge detection.

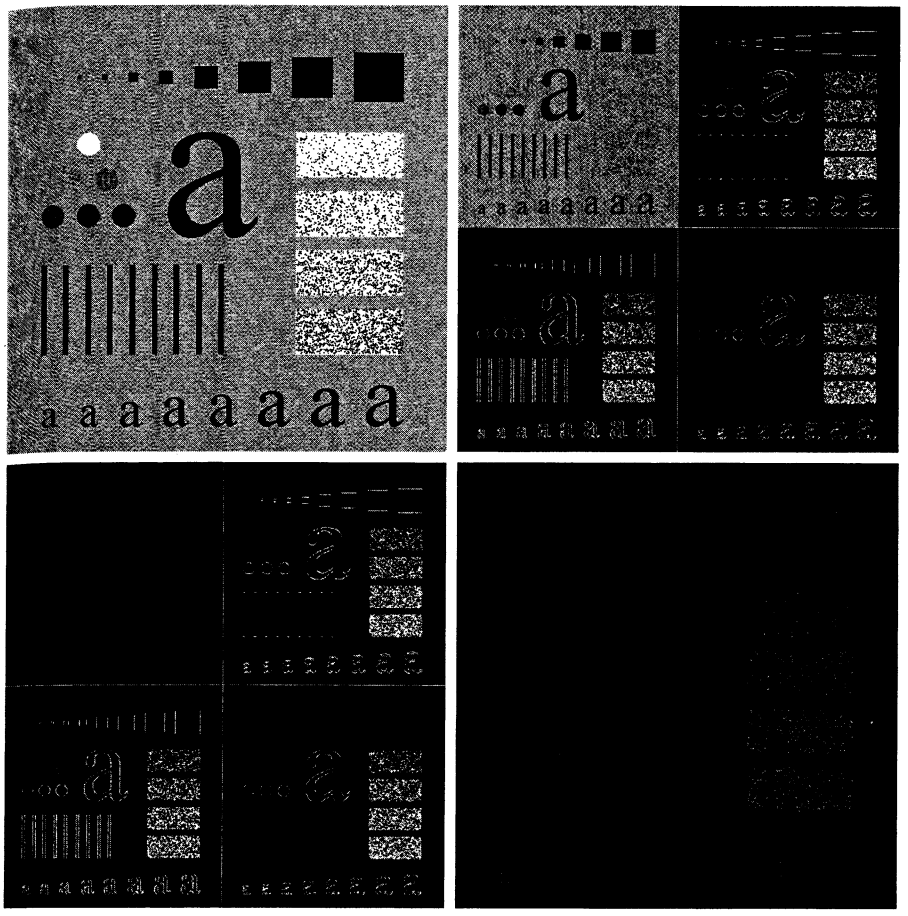
■ Consider the 500 × 500 test image in Fig. 7.7(a). This image was used in Chapter 4 to illustrate smoothing and sharpening with Fourier transforms. Here, we use it to demonstrate the directional sensitivity of the 2-D wavelet transform and its usefulness in edge detection:

```
>> f = imread('A.tif');
>> imshow(f);
>> [c, s] = wavefast(f, 1, 'sym4');
>> figure; wave2gray(c, s, -6);
>> [nc, y] = wavecut('a', c, s);
>> figure; wave2gray(nc, s, -6);
>> edges = abs(waveback(nc, s, 'sym4'));
>> figure; imshow(mat2gray(edges));
```

The horizontal, vertical, and diagonal directionality of the single-scale wavelet transform of Fig. 7.7(a) with respect to `'sym4'` wavelets is clearly visible in Fig. 7.7(b). Note, for example, that the horizontal edges of the original image are present in the horizontal detail coefficients of the upper-right quadrant of Fig. 7.7(b). The vertical edges of the image can be similarly identified in the vertical detail coefficients of the lower-left quadrant. To combine this information into a single edge image, we simply zero the approximation coefficients of the generated transform, compute its inverse, and take the absolute value. The modified transform and resulting edge image are shown in Figs. 7.7(c) and (d), respectively. A similar procedure can be used to isolate the vertical or horizontal edges alone. ■

EXAMPLE 7.9: Wavelet-based image smoothing or blurring.

■ Wavelets, like their Fourier counterparts, are effective instruments for smoothing or blurring images. Consider again the test image of Fig. 7.7(a), which is repeated in Fig. 7.8(a). Its wavelet transform with respect to fourth-



a b
c d

FIGURE 7.7 Wavelets in edge detection: (a) A simple test image; (b) its wavelet transform; (c) the transform modified by zeroing all approximation coefficients; and (d) the edge image resulting from computing the absolute value of the inverse transform.

order symlets is shown in Fig. 7.8(b), where it is clear that a four-scale decomposition has been performed. To streamline the smoothing process, we employ the following utility function:

wavezero

```
function [nc, g8] = wavezero(c, s, l, wname)
%WAVEZERO Zeroes wavelet transform detail coefficients.
%   [NC, G8] = WAVEZERO(C, S, L, WNAME) zeroes the level L detail
%   coefficients in wavelet decomposition structure [C, S] and
%   computes the resulting inverse transform with respect to WNAME
%   wavelets.

[nc, foo] = wavecut('h', c, s, l);
[nc, foo] = wavecut('v', nc, s, l);
[nc, foo] = wavecut('d', nc, s, l);
i = waveback(nc, s, wname);
g8 = im2uint8(mat2gray(i));
figure; imshow(g8);
```

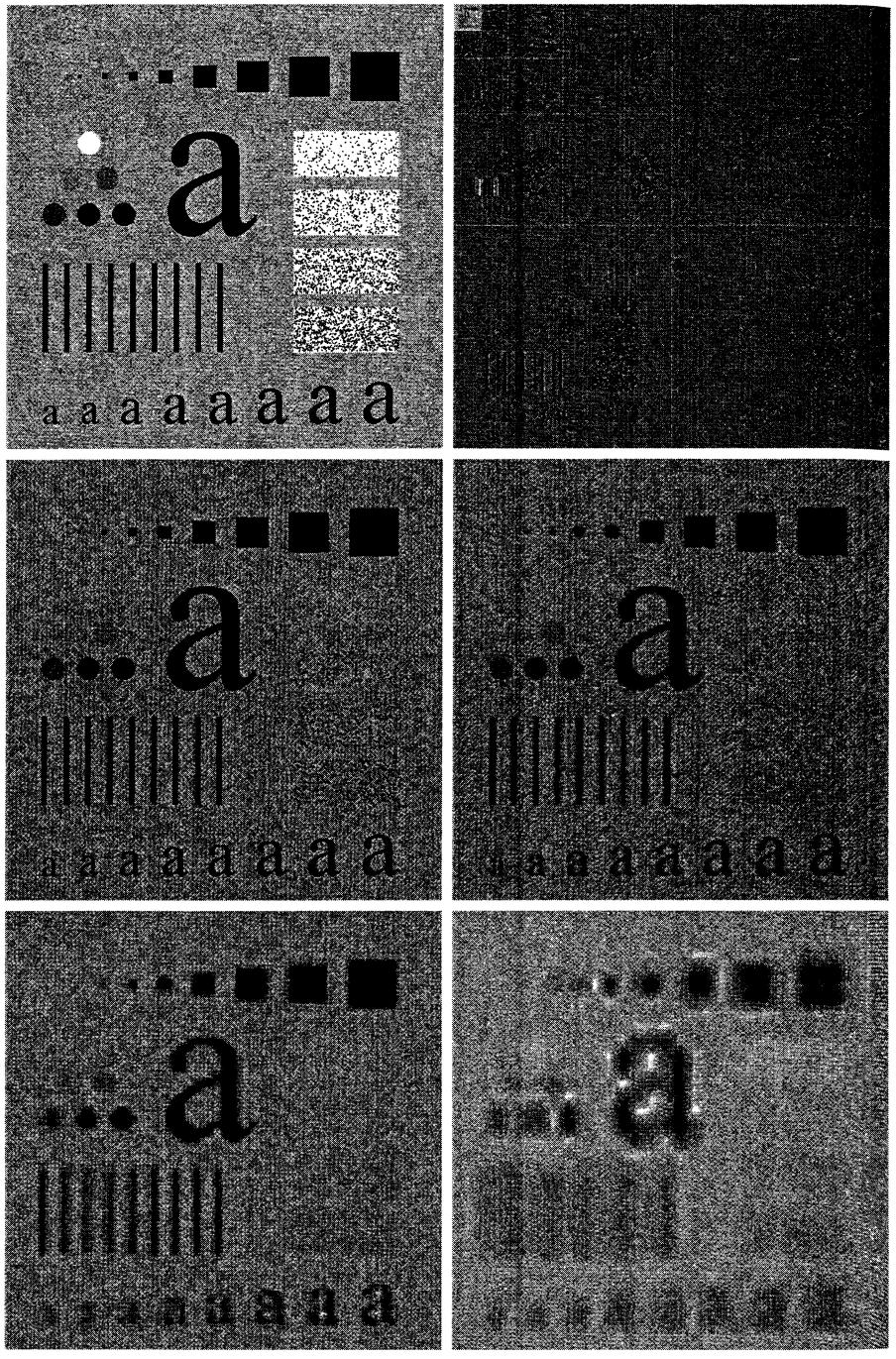


FIGURE 7.8
Wavelet-based image smoothing: (a) A test image; (b) its wavelet transform; (c) the inverse transform after zeroing the first-level detail coefficients; and (d) through (f) similar results after zeroing the second-, third-, and fourth-level details.

Using `wavezero`, a series of increasingly smoothed versions of Fig. 7.8(a) can be generated with the following sequence of commands:

```
>> f = imread('A.tif');
>> [c, s] = wavefast(f, 4, 'sym4');
>> wave2gray(c, s, 20);
>> [c, g8] = wavezero(c, s, 1, 'sym4');
>> [c, g8] = wavezero(c, s, 2, 'sym4');
>> [c, g8] = wavezero(c, s, 3, 'sym4');
>> [c, g8] = wavezero(c, s, 4, 'sym4');
```

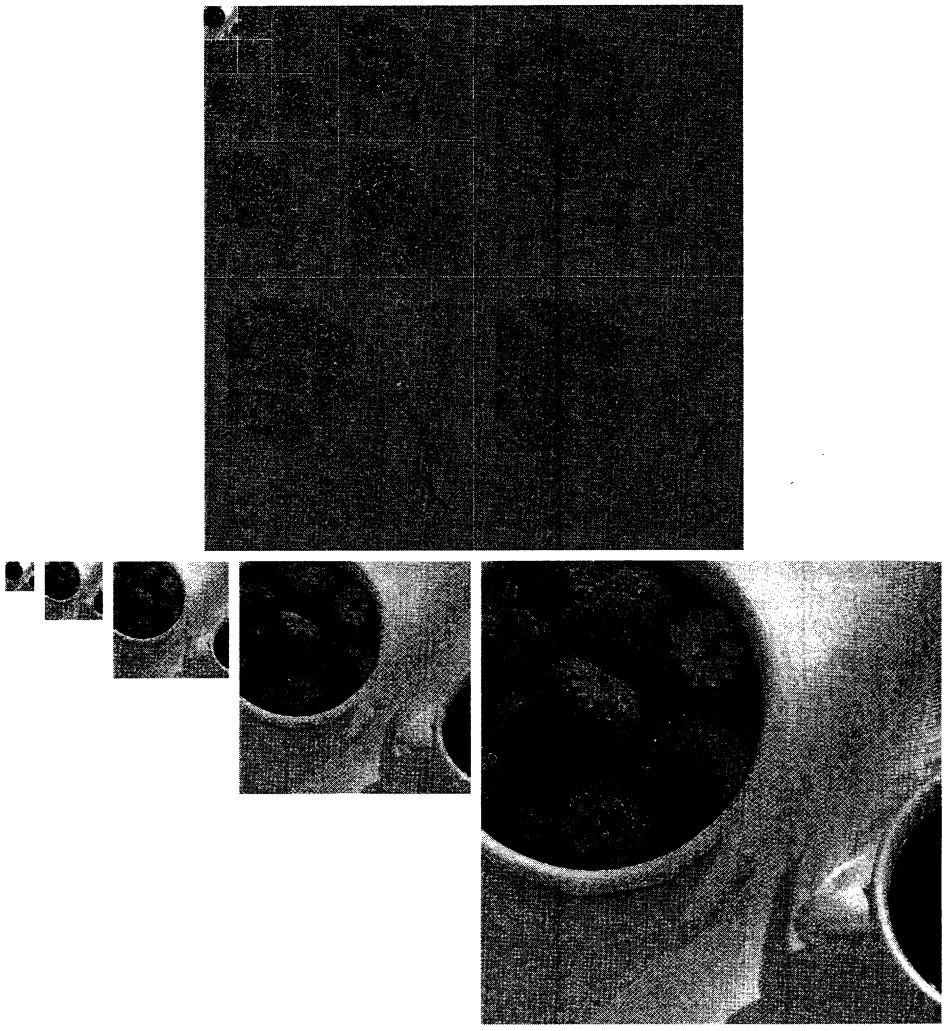
Note that the smoothed image in Fig. 7.8(c) is only slightly blurred, as it was obtained by zeroing only the first-level detail coefficients of the original image's wavelet transform (and computing the modified tranform's inverse). Additional blurring is present in the second result—Fig. 7.8(d)—which shows the effect of zeroing the second level detail coefficients as well. The coefficient zeroing process continues in Fig. 7.8(e), where the third level of details is zeroed, and concludes with Fig. 7.8(f), where all the detail coefficients have been eliminated. The gradual increase in blurring from Figs. 7.8(c) to (f) is reminiscent of similar results with Fourier transforms. It illustrates the intimate relationship between scale in the wavelet domain and frequency in the Fourier domain. ■

EXAMPLE 7.10: Progressive reconstruction.

■ Consider next the transmission and reconstruction of the four-scale wavelet transform in Fig. 7.9(a) within the context of browsing a remote image database for a specific image. Here, we deviate from the three-step procedure described at the beginning of this section and consider an application without a Fourier domain counterpart. Each image in the database is stored as a multiscale wavelet decomposition. This structure is well suited to progressive reconstruction applications, particularly when the 1-D decomposition vector used to store the transform's coefficients assumes the general format of Section 7.3. For the four-scale transform of this example, the decomposition vector is

$$[\mathbf{A}_4(:)' \quad \mathbf{H}_4(:)' \quad \cdots \quad \mathbf{H}_i(:)' \quad \mathbf{V}_i(:)' \quad \mathbf{D}_i(:)' \quad \cdots \quad \mathbf{V}_1(:)' \quad \mathbf{D}_1(:)']$$

where $\mathbf{A}_4$ is the approximation coefficient matrix of the fourth decomposition level and $\mathbf{H}_i$, $\mathbf{V}_j$, and $\mathbf{D}_i$ for $i = 1, 2, 3, 4$ are the horizontal, vertical, and diagonal transform coefficient matrices for level i. If we transmit this vector in a left-to-right manner, a remote display device can gradually build higher resolution approximations of the final high-resolution image (based on the user's needs) as the data arrives at the viewing station. For instance, when the $\mathbf{A}_4$ coefficients have been received, a low-resolution version of the image can be made available for viewing [Fig. 7.9(b)]. When $\mathbf{H}_4$, $\mathbf{V}_4$, and $\mathbf{D}_4$ have been received, a higher-resolution approximation [Fig. 7.9(c)] can be constructed, and



a
b c d e f

FIGURE 7.9 Progressive reconstruction: (a) A four-scale wavelet transform; (b) the fourth-level approximation image from the upper-left corner; (c) a refined approximation incorporating the fourth-level details; (d) through (f) further resolution improvements incorporating higher-level details.

so on. Figures 7.9(d) through (f) provide three additional reconstructions of increasing resolution. This progressive reconstruction process is easily simulated using the following MATLAB command sequence:

```
>> f = imread('Strawberries.tif');          % Generate transform
>> [c, s] = wavefast(f, 4, 'jpeg9.7');
>> wave2gray(c, s, 8);
```

```
>>
>> f = wavecopy('a', c, s);                    % Approximation 1
>> figure; imshow(mat2gray(f));
>>
>> [c, s] = waveback(c, s, 'jpeg9.7', 1);      % Approximation 2
>> f = wavecopy('a', c, s);
>> figure; imshow(mat2gray(f));
>> [c, s] = waveback(c, s, 'jpeg9.7', 1);      % Approximation 3
>> f = wavecopy('a', c, s);
>> figure; imshow(mat2gray(f));
>> [c, s] = waveback(c, s, 'jpeg9.7', 1);      % Approximation 4
>> f = wavecopy('a', c, s);
>> figure; imshow(mat2gray(f));
>> [c, s] = waveback(c, s, 'jpeg9.7', 1);      % Final image
>> f = wavecopy('a', c, s);
>> figure; imshow(mat2gray(f));
```

Note that the final four approximations use `waveback` to perform single level reconstructions. ■

Summary

The material in this chapter introduces the wavelet transform and its use in image processing. Like the Fourier transform, wavelet transforms can be used in tasks ranging from edge detection to image smoothing, both of which are considered in the material that is covered. Because they provide significant insight into both an image's spatial and frequency characteristics, wavelets can also be used in applications in which Fourier methods are not well suited, like progressive image reconstruction (see Example 7.10). Because the Image Processing Toolbox does not include routines for computing or using wavelet transforms, a significant portion of this chapter is devoted to the development of a set of functions that extend the Image Processing Toolbox to wavelet-based imaging. The functions developed were designed to be fully compatible with MATLAB's Wavelet Toolbox, which is introduced in this chapter but is not a part of the Image Processing Toolbox. In the next chapter, wavelets will be used for image compression, an area in which they have received considerable attention in the literature.