Fast algorithms for wavelet transform computation

Olivier Rioul, Pierre Duhamel

Chapter 8

Fast Algorithms for Wavelet Transform Computation

Olivier Rioul, Pierre Duhamel

8.1. INTRODUCTION

Wavelet transforms have a wide range of applications, from signal analysis to image or data compression. Compared to the classical Fourier-based transformations, it can play either the role of the short time Fourier transform—or the Gabor transform—or that of a discrete Fourier transform, or even that of a discrete cosine transform. Therefore, it is not astonishing that the tool referred to as "wavelet transform" can take very different forms, depending on the application.

The continuous wavelet transform is best suited to signal analysis [1–3, 5–7]. Its semi-discrete version (wavelet series) and its fully discrete one (discrete wavelet transform) have been used for signal coding applications, including image compression [4–6] and various tasks in computer vision [8,9]. Wavelet transforms also find applications in many other fields, too numerous to be listed here (see e.g., [5]).

8.1.1 Classification of Wavelet Transforms

In a general sense, a wavelet transformation of a time-varying signal $x(t)$ consists of computing coefficients that are inner products of $x(t)$ against a family of "wavelets." These wavelets $\psi_{a,b}(t)$ are labeled by scale and time location parameters a and b. In a *continuous* wavelet transform, the wavelet corresponding to scale a and time location b is

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right) \tag{8-1}$$

where $\psi(t)$ is the wavelet "prototype," which can be thought of as a bandpass

function (the factor $|a|^{-1/2}$ is there to ensure energy preservation [2,5]). There are various ways of discretizing time-scale parameters, each one yielding a different type of wavelet transform. We adopt the following terminology, which parallels the one commonly used for Fourier transforms.

The continuous wavelet transform (CWT) was originally introduced by Goupillaud, Grossmann, and Morlet [2], and is given by

$$\mathrm{CWT}\{x(t);\, a, b\} = \int x(t)\psi^*_{a,b}(t)dt \tag{8-2}$$

where the asterisk stands for complex conjugation. Time t and the time-scale parameters (b, a) vary continuously.

Wavelet series (WS) coefficients are sampled CWT coefficients. Time remains continuous but time-scale parameters are sampled on a "dyadic" grid in the time-scale plane (b, a) [4,5,8–12]. A usual definition is

$$C_{j,k} = \mathrm{CWT}\{x(t);\quad a = 2^j,\quad b = k2^j\} \quad \text{for} \quad j, k \in Z \tag{8-3}$$

The wavelets are, in this case,

$$\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k) \tag{8-4}$$

and the original signal can be recovered through the following formula:

$$x(t) = \sum_{j\in Z}\sum_{k\in Z} C_{j,k}\tilde{\psi}_{j,k}(t) \tag{8-5}$$

where wavelets $\tilde{\psi}_{j,k}(t)$ are also of the form (8-4).

Wavelet series have been popularized under the form of a signal decomposition onto "orthogonal wavelets" by Meyer, Mallat, Daubechies, and other authors [5,8–11,13]. In the orthogonal case, the functions $\psi_{j,k}(t)$ and $\tilde{\psi}_{j,k}(t)$ are equal, and form an orthogonal basis. If, more generally, (8-3) and (8-5) hold exactly for $\psi_{j,k}(t)$ and $\tilde{\psi}_{j,k}(t)$ not necessarily equal, we are in the so-called "biorthogonal" case; the two sets of wavelet functions form two "mutually orthogonal" bases [4,12,14].

The discrete-time wavelet transform (DTWT) corresponds to the (continuous) Wavelet Transform of a sampled sequence $x_n = x(nT)$. Assuming sampling period T to be unity leads us to consider only integer time shifts in the analysis, resulting in

$$\mathrm{DTWT}\{x_n;\, a, m\} = \sum_n x_n\psi^*_{a,m}(n) \tag{8-6}$$

The discrete wavelet transform (DWT) (see e.g., [12,13,15]) applies to discrete-time signals—both time and time-scale parameters are discrete. A DWT output on J "octaves" consists of "wavelet coefficients" c_{jk} computed for $j = 1, \ldots, J$:

$$\mathrm{DWT}\{x(n);\, 2^j, k2^j\} = c_{j,k} = \sum_n x_n h^*_j(n - 2^j k) \tag{8-7}$$

and "residual coefficients" at octave J given by

$$r_{J,k} = \sum_n x_n g^*_J(n - 2^J k) \tag{8-8}$$

The $g_J(n - 2^J k)$ are the analysis *scaling sequences*: They are used to bring the input signal from the initial scale to scale 2^J. The $h_j(n - 2^j k)$ are the analysis wavelets, the discrete equivalent to the $2^{-j/2}\psi[2^{-j}(t - 2^j k)]$. The connection between both versions (discrete and continuous) is clarified later.

The reconstruction formula by which the inverse DWT reconstructs the signal from its coefficients is given by

$$x_n = \sum_{j=1}^{\infty} \sum_{k \in Z} c_{j,k} \tilde{h}_j(n - 2^j k) + \sum_{k \in Z} r_{J,k} \tilde{g}_J(n - 2^J k) \tag{8-9}$$

This formula is to be compared with (8-5). The main difference, apart from discretization, is the additional (low-pass) term: it is there to ensure perfect reconstruction, due to the finite iteration on the scale ($j = 1, \cdots, J$ in place of $j \in Z$). "Scaling functions" similar to the $g_J(n - 2^J k)$ can be defined for wavelet series [5,8–12,14–16] as shown in section 8.2.1.

8.1.2 Note on the Choice of the Wavelet

Orthogonality and biorthogonality properties, as defined earlier in the WS case, hold also for the DTWT and DWT, using appropriate (continuous or discrete) definitions of the inner product. The choice of particular orthogonal or biorthogonal wavelets is sometimes of importance in particular applications.

Here we focus on implementation issues, not on wavelet design. Therefore, even though design constraints on the shape of wavelets can sometimes be used to reduce the computational load, we do not take advantage of them so as to be as general as possible. Note, however, that orthogonality can bring slight computational gains, at the cost of a more involved implementation [17], and linear phase wavelets (possible only in the biorthogonal case) can be used to cut the number of multiplications by 2 in the straightforward implementation of the DWT described in section 8.4, by a simple use of symmetry in impulse responses.

We shall also restrict our focus to the (most frequent) case of wavelets with finite support. The issue of designing a wavelet with finite support is somewhat similar to a situation found in classical spectral analysis: when analyzing time-varying signals with Fourier-based tools, one cannot use the continuous Fourier transform directly, since it involves the whole signal of infinite support. Hence, the signal is restricted to a short segment around the instant of analysis by applying some window, and this windowed segment is then analyzed by a Fourier transform. Here, the design of a wavelet with finite support includes that of the window. This explains why one often chooses the wavelet in some library, just like the window for the short-time Fourier transform. The problem of designing some wavelet transform with specific properties is not addressed here. Note, however, that the spline implementation of wavelet transform offers much flexibility for this purpose [27, 28].

8.2. MULTIRESOLUTION AND TWO-SCALE EQUATIONS

If we stay with our previous definitions of wavelet transforms, the problem of choosing a wavelet is almost totally unconstrained, and full flexibility is possible—particularly in the case of the CWT. However, one of the main concepts of wavelet theory is the interpretation of wavelet transforms in terms of *multiresolution decomposition*. Of course, wavelets can exist without the multiresolution interpretation. However, this concept is so enlightening that we shall briefly outline its underlying concepts. As shown in the following sections, this is especially useful for fast wavelet algorithms and for the initial approximations that usually have to be performed on the signal and wavelets.

8.2.1 Multiresolution Spaces

A *multiresolution analysis* of $L^2(\mathbf{R})$ is a sequence $\{V_j\}$ $(j \in \mathbf{Z})$ of subspaces of $L^2(\mathbf{R})$, having the properties listed here (see [11] for mathematical details). The V_j's model spaces of signals having resolution at most 2^{-j}.

- Every signal lies in some V_j, and no signal—except the null signal—belongs to all V_j.
- V_j contains V_{j+1}.
- V_j is closed under time shifts $t \to t - k2^{-j}$, and $x(t) \in V_0$ is equivalent to $x(2^{-j}t) \in V_j$.
- There exists a function $\phi(t) \in V_0$ such that the set $\{\phi(t-k),\ \ k \in \mathbf{Z}\}$ forms a basis of V_0.

The function $\phi(t)$ is the *scaling function*. It is easily seen that the set of functions, defined in a "dyadic wavelet style" as

$$\phi_{j,k}(t) = 2^{-j/2}\phi(2^{-j}t - k) \tag{8-10}$$

forms a basis of V_j. Hence, all elements of V_j can be defined as linear combinations of $\phi_{j,k}(t)$.

Wavelet spaces W_j are orthogonal complements to V_j in V_{j-1}. They contain the necessary information to go from resolution 2^{-j} to $2^{-(j-1)}$. By construction, the subspaces $\{W_j\}$ are mutually orthogonal, and their direct sum spans the whole signal space $L^2(\mathbf{R})$.

One of the main results of the multiresolution theory is the existence of a function $\psi(t)$—the "mother wavelet"—constructed from the scaling function, and such that the set $\psi(t-k)$ is an orthonormal basis of W_0. Hence it follows from the definition of the W_j's that

$$W_j = \left\{ x(t) = \sum_{k \in Z} c_{j,k} 2^{-j/2} \psi(2^{-j}t - k), \quad c_{j,k} \in l_2(\mathbf{Z}^2) \right\} \tag{8-11}$$

and the set $2^{-j/2}\psi(2^{-j}t - k)$ forms an orthonormal basis of $L^2(\mathbf{R})$. This corresponds exactly to the definition of an orthonormal wavelet series: the coefficients of the WS at scale j are the coordinates of the signal in space W_j.

The biorthogonal case is slightly more complicated. It involves two sequences of multiresolution spaces, one (V_j) for the analysis, and the other $(\tilde{V}_j)$ for the synthesis.

8.2.2 Examples

Classical examples of multiresolution related with the topic of fast algorithms are

- Haar wavelet: the scaling function $\phi(t)$ is a rectangle of value 1 on the interval [0, 1), and V_0 is the space of the functions of $L^2(\mathbf{R})$, which are constant by parts.
- The dual situation in frequency: scaling functions $\phi(t)$ have a compact support spectrum e.g., on [−.5, .5]. A natural candidate for $\phi(t)$ is the sinc function, and a corresponding wavelet $\psi(t)$ can be obtained as a linear combination of sinc functions.
  This leads to an interpretation of Shannon's theorem in terms of multiresolution: Sampling corresponds to the projection into a multiresolution space, while the signal with the next coarser resolution has a spectrum twice as small. This will lead to an interesting interpretation of the initial approximation of a fast DWT algorithm as being similar to the half-band prefiltering made prior to sampling.
- Spaces of spline functions, built by parts, using polynomials with degree lower or equal to d. V_0 can be obtained through the use of the B-spline function $\beta^d(t)$ of order d, the dth iterative convolution of the unit rectangular pulse. These functions naturally lead to multiresolution spaces, since they are imbricated:

$$\beta^d(t/2) = \sum_{k \in Z} c_k^d \beta^d(t - k) \tag{8-12}$$

  These functions play an important role in many respects. The obtained spline wavelets have many useful properties, among them the possibility of convergence toward Gabor functions, which have an optimal mapping of the time-frequency plane. Second, they can easily be used to approximate a wavelet of any (time domain) shape, while building a multiresolution analysis, hence allowing the use of fast algorithms.

8.2.3 Two-Scale Equations

By construction, $\phi(t/2) \in V_1 \subset V_0$, and $\psi(t/2) \in V_1 \subset V_0$. These functions can therefore be expressed as linear combinations of $\{\phi(t - k)\}$, the basis functions of V_0. We obtain *two-scale difference equations*:

$$\frac{1}{\sqrt{2}}\phi(t/2) = \sum_{k\in Z} g_k \phi(t-k) = g * \phi \tag{8-13}$$

$$\frac{1}{\sqrt{2}}\psi(t/2) = \sum_{k\in Z} h_k \phi(t-k) = h * \phi \tag{8-14}$$

It is known in multiresolution theory that the scaling function and the wavelet are fully characterized by the set of coefficients g_k and h_k. These coefficients are, in fact, impulse responses of filters used in the implementation of a DWT. They correspond, in our notation, to scaling sequences $g_1(n-k)$ and wavelets $h_1(n-k)$. It is therefore natural to consider the DWT as a natural implementation of WS, as explained in the next section.

8.3. THE INITIAL SIGNAL APPROXIMATION

Assume that an approximation of a CWT [defined as in (8-2)] has to be computed, and consider the following analogy with Fourier transforms. When implementing the short time Fourier transform of some continuous signal, one first *samples* the continuous signal. Information is not lost under the assumption that the signal has a finite spectrum, by Shannon's sampling theorem. This finite spectrum property is ensured by some prefiltering to avoid spectrum aliasing. It is well known that this corresponds to a projection of the initial signal onto the space of finite spectrum signals, which minimizes the mean square error of the frequency estimates.

This section is concerned with the same problem in the wavelet case [18]: Given some wavelet, which continuous signals can be represented by wavelet series without loss of information? Intuitively, this class of signals will be the only ones for which there will be no possibility of misinterpretation when exploiting the wavelet coefficients (think of the Fourier analogy: spectrum aliasing). Also, which procedure has to be applied in order to minimize the reconstruction error (formally equivalent to prefiltering in the Fourier case)?

Note that this problem can be stated in the context of a generalized sampling theory, in which the sampler no longer takes its ideal form $x_n = x(nT)$ (see [19]). But a direct use of such a generalized sampling theorem would require the knowledge (or worse, the design) of some precise sampling device. However, one usually knows only the samples of the signal, which are assumed to be sampled according to Shannon's theorem. Therefore, we follow here the approach of Abry and Flandrin [18], which is closely related to practical applications.

Assuming the continuous time is normalized such that $T = 1$, the continuous signal is related to its samples x_n by

$$x(t) = \sum_n x_n \mathrm{sinc}(t-n) \tag{8-15}$$

$$x_n = \int x(t)\mathrm{sinc}(t-n)dt \tag{8-16}$$

However, the initial signal, in a discrete implementation of a multiresolution procedure such as DWT or WS, is assumed to belong to V_0. Hence, the initialization should consist of projecting the signal $x(t)$ into V_0, as follows.

$$\hat{x}_k = \int x(t)\phi(t-k)dt \tag{8-17}$$

$$= \sum_n x_n \int \text{sinc}(t-n)\phi(t-k)dt$$

$$= \sum_n x_n \int \text{sinc}(u)\phi(u-k+n)du$$

$$= \sum_n x_n f_{k-n}, \qquad \text{with } f_k = \langle \text{sinc}, \phi(\cdot - k)\rangle \tag{8-18}$$

This initialization takes the form of a digital prefiltering, which has to be applied before any computation involving multiresolution. When this computation is possible, it will ensure the estimation of the wavelet coefficients with the least distortion. However, these coefficients f_k are obtained through an integral involving the (continuous) wavelet, which may be computationally intensive if several wavelets are to be used on the signal. In this case, a cheap approximation has been proposed in [18], which we now summarize.

Quite often, no approximation is made prior to the wavelet computation, i.e., one uses the implicit choice $\hat{x}_n = x_n$. It is then possible to show that the errors made on the approximations at the various scales and on the additional "details" come from the distance of the scale function ϕ to an ideal low-pass filter. This makes sense, since if the initial projection were an ideal sampler, the initial projection would be this ideal low-pass filter. The idea, explained in [18], is to make use of the fundamental low-pass character of ϕ. Since most of its energy lies in the frequency range $[-0.5, 0.5]$, the result of its convolution by the sinc function will not change much of its spectrum. Hence, a reasonable approximation is

$$f_k \approx \phi(-k) \tag{8-19}$$

$$\hat{x}_n = \sum_n x_n \phi(n-k) \tag{8-20}$$

Abry and Flandrin [18] provide convincing examples showing the necessity of the initialization. Note, however, that the initialization is not compulsory in the special case where the scaling function has $[-1, 1]$ as time support (e.g., Haar wavelet, splines of order 0 or 1), or when the input data are largely oversampled.

8.3.1 Remarks on Initialization and Sampling

Sampling a continuous signal consists of representing the whole information carried by this signal by means of a discrete sequence of numbers $\hat{x}_n$. In the case of Shannon sampling, there is the additional requirement that these numbers $\hat{x}_n = x(nT)$. It is well known that this operation is feasible if the input signal is ensured to have a finite spectrum by some prefiltering. This prefiltering is a projection of the original signal into the subspace of $L^2(\mathbf{R})$ of the finite spectrum functions.

This space can be generated by a linear combination of translated sinc functions (the interpolation formula: from the samples to the continuous function), thus forming a multiresolution, whose scale function generates an orthogonal basis.

If, however, one does not constrain the "samples" $\hat{x}_n$ to be values taken by the signal at regularly spaced instants, the projection of $x(t)$ into any multiresolution space V_0 takes the general form (8-18), which is also some kind of sampling procedure. In what follows, the prefiltering will be assumed to have been applied prior to the fast algorithm computation

8.4. THE DISCRETE WAVELET TRANSFORM (DWT)

Most fast algorithms for WT computation use the DWT as a basic building block [5,8–14,16]), hence its importance. As a transform of its own, the DWT mainly finds application in image compression [4–6,8,9] (in a two-dimensional form), but is also another description of octave-band filter banks that were used for some time in one-dimensional coding schemes [17,20].

The DWT is very much like a WS but applies to discrete-time signals x_n, $n \in Z$. More than a simple discretization of the DTWT to the dyadic grid, we assume that it achieves a multiresolution decomposition of x_n on J octaves labeled by $j = 1, \cdots, J$. It is precisely this requirement for a multiresolution—hence hierarchical—structure that makes fast computation possible. The requirement for a multiresolution computation can be stated as follows: Given some signal, at scale j, one decomposes it in a sum of details, at scale $j+1$ (the true wavelet coefficients), plus some residual, representing the signal at resolution $j+1$ (twice as coarse). A further analysis at coarser scales involves only the residual (think of the imbrication of subspaces in section 8.2). This requirement relies on the wavelet and on the signal: whether such a computation corresponds exactly to a sampling of the DTWT or not depends on properties of the wavelet (two-scale difference equation) and of the signal (initialization).

The efficient DWT computational structure can be obtained by observing that, due to the multiresolution requirement, wavelets and scaling sequences can be deduced from one octave to the next by some two-scale difference equation. Consider the analysis part (the treatment of synthesis "basis functions" is similar), and proceed by analogy with the multiresolution defined on WS in section 8.2. Consider two filter impulse responses $g(n)$ (corresponding to some low-pass interpolating filter—the scaling function) and $h(n)$ (corresponding to a high-pass filter—the discrete wavelet). The wavelets and scaling sequences are obtained iteratively as

$$g_1(n) = g(n) \qquad h_1(n) = h(n)$$

$$h_{j+1}(n) = \sum_k h_j(k) g(n - 2k) \tag{8-21}$$

$$g_{j+1}(n) = \sum_k g_j(k) g(n - 2k) \tag{8-22}$$

i.e., one goes from one octave j to the next $(j+1)$ by applying the interpolation operator

$$f(n) \rightarrow \sum_k f(k)g(n-2k) \tag{8-23}$$

which should be thought of as the discrete equivalent to the dilation $f(t) \rightarrow 2^{-1/2}f(t/2)$.

Consider, for example, the computation of $c_{j,k}$ as given by (8-7). For fixed j, $c_{j,k}$ is the result of filtering the input signal by $h_j(n)$ and then decimating the output by discarding one every 2^jth sample. Now the z-transform of filter $h_j(n)$ can be easily deduced from (8-21), which reads $H_{j+1}(z) = H_j(z^2)G(z)$ in z-transform notation. We obtain

$$H_{j+1}(z) = G(z)G(z^2)\cdots G(z^{2^{j-1}})H(z^{2^j}) \tag{8-24}$$

and, similarly for $g_j(n)$,

$$G_{j+1}(z) = G(z)G(z^2)\cdots G(z^{2^j}) \tag{8-25}$$

The computations of a DWT are now easily reorganized in the form of a binary tree, as shown in Fig. 8-1.

It is thus easily recognized that the structure of computations in a DWT is exactly an octave-band filter bank [8,12,13,15,17,20] as depicted in Fig. 8-1. The DWT corresponds to the analysis filter bank with filters $g(n)$ and $h(n)$, whereas the inverse DWT (IDWT) corresponds to the synthesis filter bank with filters $\tilde{g}(n)$ and $\tilde{h}(n)$.

Note that this filter bank is critically sampled: given N input samples, the DWT computes about $N/2 + N/4 + \cdots + N2^{-J} + N2^{-J} = N$ coefficients. In keeping with the critical sampling, the octave parameter j is restricted to $j \geq 1$ so that the sampling rate of wavelet coefficients is always less than that of the signal. Whenever the inverse DWT is used in the following, we assume that the filters $g(n)$, $h(n)$, $\tilde{g}(n)$, and $\tilde{h}(n)$ have been suitably designed so that (8-7) and (8-9) hold exactly. That is, the filter bank of Fig. 8-1 allows perfect reconstruction (this corresponds to the biorthogonal case). The reader is referred to [10,12,14,17,20] for more details on the design.

8.5. THE DWT FOR WS COMPUTATION

8.5.1 WS Computation: Mallat and Shensa Algorithm

It is well known since Mallat [8,9] that orthogonal wavelet series can be implemented using an orthogonal DWT, provided the discrete input is related to the original signal $x(t)$ by (8-17). The resulting algorithm, using filter banks, has been popularized as the Mallat algorithm. It was first derived using particular orthonormal wavelets.

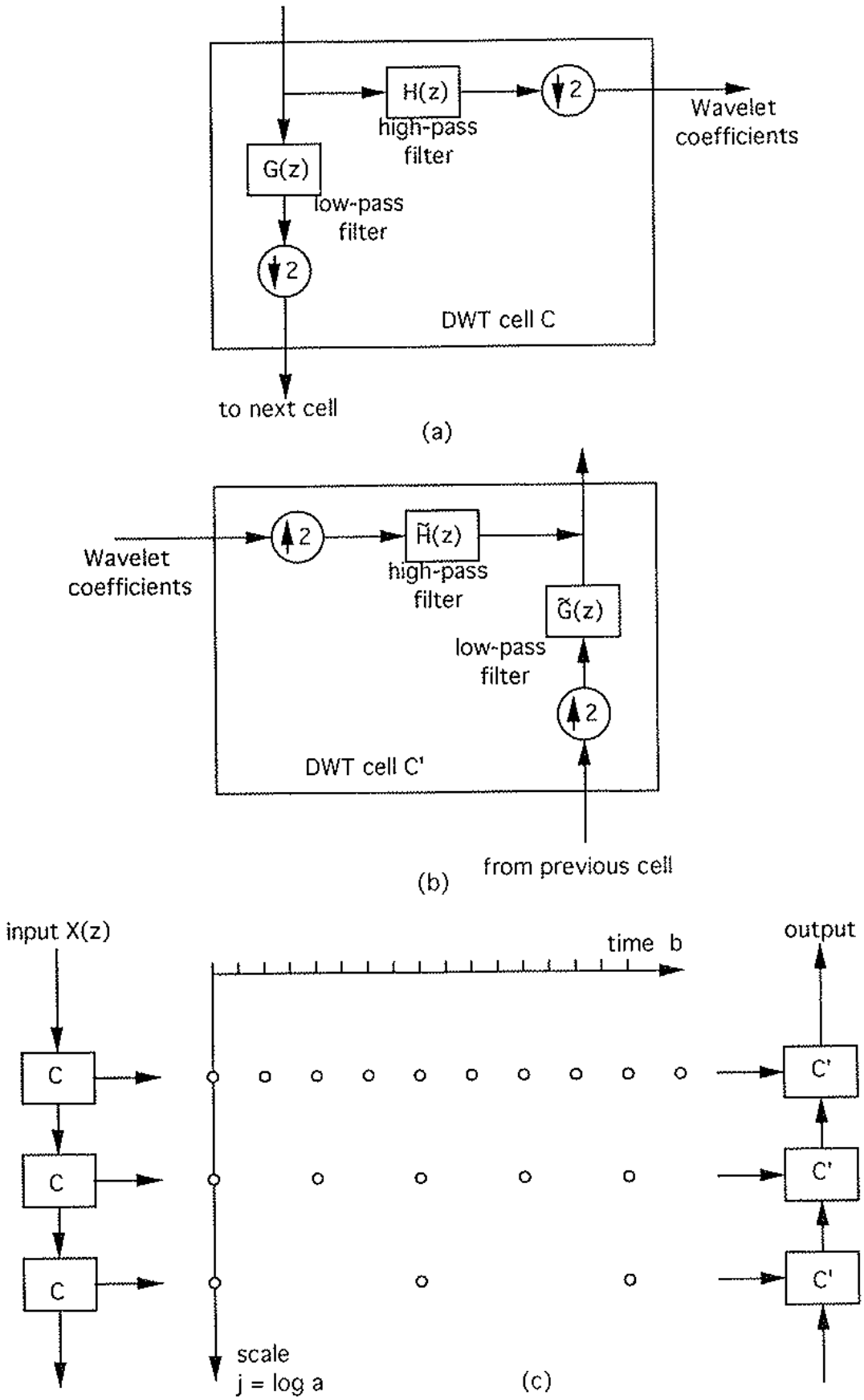

Figure 8-1 An octave-band filter bank. Basic computational cell of (a) the DWT and (b) the inverse DWT. (c) Overall organization takes the form of an octave-band filter bank. The analysis part gives wavelet coefficients that correspond to a dyadic grid in the time-scale plane. Signal is reconstructed using the transposed scheme (b) (synthesis filter bank).

The general algorithm of interest to us now, derived by Shensa [15], can be described as follows. Given the continuous-time wavelet $\psi(t)$, one first approximates it by $\hat{\psi}(t)$ in such a way that the following equation holds:

$$2^{-j/2}\hat{\psi}(2^{-j}t) = \sum_n h_j(n)\phi(t-n), \qquad j = 1, \cdots, J \tag{8-26}$$

where $h_j(n)$ are discrete wavelets present in a DWT, and $\phi(t)$ is some interpolating function (the scale function). The precise way these J simultaneous approximations can be accomplished is outlined in section 8.5.2.

The derivation of the algorithm is now straightforward. Substituting (8-26) into the equation defining the WS coefficients (8-3), and assuming that the initialization (8-17) has been done gives:

$$\hat{C}_{j,k} = \int \hat{x}(t)2^{-j/2}\hat{\psi}^*(2^{-j}t - k)dt \tag{8-27}$$

$$= \sum_n \hat{x}_n h_j^*(n - 2^j k) \tag{8-28}$$

$$= DWT\{\hat{x}_n, 2^j, k2^j\} \tag{8-29}$$

This ends the derivation of the Shensa algorithm: the WS coefficients with respect to the approximated wavelet $\hat{\psi}(t)$ are computed exactly for all signals using a DWT, provided that the input is appropriately prefiltered. The accuracy of this algorithm is balanced by the approximations made for the input (8-17) and for the wavelets (8-26); the algorithm is exact only when the input and the wavelets have been replaced by their approximations.

Note that we have three different types of inputs at work: the original analog signal, its approximation introduced by the original sampling, with discrete-time samples x_n, and the filtered version $\hat{x}_n$ defined by (8-18). They involve two successive approximations: the first one is made regardless of the parameters in the algorithm (initial sampling). The second one is the prefiltering, which depends on the parameters of the algorithm, and amounts to a nonorthogonal projection of $x(t)$.

8.5.2 The Wavelet Approximation

One may wonder how (8-26) can be computed. These approximations are important because their accuracy determines that of the whole algorithm. First, note that this whole set of equations is equivalent to assuming that Eqs. (8-14) [which is (8-26) rewritten for $j = 1$)], plus (8-13) hold. These two-scale difference equations were studied in detail by Daubechies and Lagarias in [21].

There are two steps involved. First, determine a low-pass filter $g(n)$ and an interpolating function $\phi(t)$ satisfying (8-13). Second, approximate $\psi(t)$ by linear combinations of integer translates of $\phi(t)$ (8-14). This step determines the high-pass filter $h(n)$. Of course, it is crucial to choose a good interpolating function $\phi(t)$ so that $\psi(t)$ can be accurately approximated. Note, however, that once $\psi(t)$ is accurately approximated by $\hat{\psi}(t)$ for which (8-13) and (8-14) hold, the J approximations at all scales (8-26) are satisfied automatically; for example, minimizing the error's

energy $\int |\psi(t) - \hat{\psi}(t)|^2 dt$ minimizes the maximum error $|C_{j,k} - \hat{C}_{j,k}|$ of the wavelet coefficients at all scales. Several "standard" choices for $\phi(t)$ were cited in section 8.2.2.

8.5.3 Using the Inverse DWT to Compute the Inverse WS (IWS)

We have seen that wavelet series coefficients (8-3) can be computed using a DWT (8-29). Similarly, its inverse transform (8-5) can be computed using an inverse DWT (8-9), under a condition similar to (8-26), but written for *synthesis* wavelets $\tilde{\psi}_{j,k}(t)$

$$2^{-j/2}\tilde{\psi}(2^{-j}t) = \sum_n \tilde{h}_j(n)\tilde{\phi}(t-n), \qquad j = 1, \cdots, J \tag{8-30}$$

Of course, this condition is, in practice, replaced by more tractable conditions as explained earlier. Substituting (8-30) for $2^{-j/2}\tilde{\psi}(2^{-j}t)$ in the formula defining the inverse WS (8-5) results in

$$\text{IWST}\{C_{j,k}\} = \sum_n y_n \tilde{\phi}(t-n) \tag{8-31}$$

where the $C_{j,k}$ are the WS coefficients (8-3), and y_n is defined by

$$y_n = \text{IDWT}\{C_{j,k}\} \tag{8-32}$$

Thus, the inverse DWT, followed by a D/A converter with characteristic $\tilde{\phi}(t)$, computes the IWS exactly.

The accuracy of the algorithm again depends on that of the signal and wavelet approximation. The resulting analysis/synthesis WS scheme is depicted in Fig. 8-2. First, the analog signal $x(t)$ is discretized according to (8-15). The discrete-time signal x_n is then prefiltered (8-18) and fed into the DWT algorithm. During synthesis, the signal is reconstructed using an inverse DWT, followed by the interpolation (or D/A conversion) (8-31).

Note that in this WS/IWS Shensa algorithm, the analysis and synthesis discrete wavelets do not necessarily form a perfect reconstruction filter bank pair. However, we now restrict the focus to the perfect reconstruction case to derive conditions under which the original signal $x(t)$ is recovered exactly.

When the DWT allows perfect reconstruction, one has $y_n = \hat{x}_n$. It can be shown that we are in fact in the "biorthogonal" case [12,14], and that one has

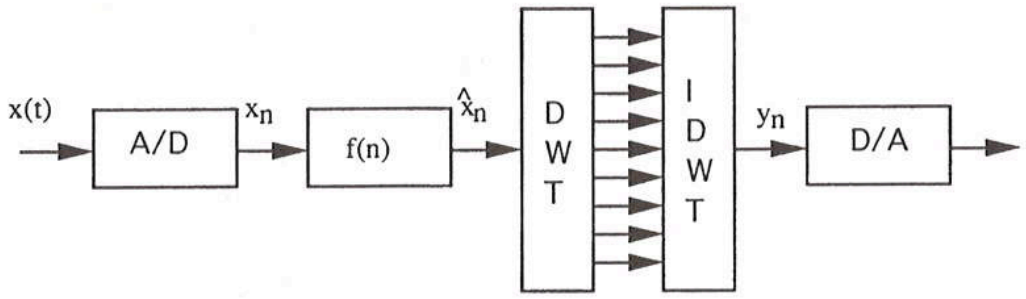

Figure 8-2 Full analysis/synthesis WS scheme. Exact reconstruction holds under certain conditions on $x(t)$ (see text).

$$\int \phi(t-n)\tilde{\phi}^*(t-m)dt = \delta_{m,n} \tag{8-33}$$

Since $y_n = \hat{x}_n$, we also have

$$\mathrm{IWS}\{\mathrm{WS}\{x(t)\}\} = \sum_n \left(\int \hat{x}(u)\phi^*(u-m)du \right) \tilde{\phi}(t-n) \tag{8-34}$$

The right-hand side of (8-34) is easily recognized to be a projection of $x(t)$ onto the subspace $\tilde{V}_0$ spanned by linear combinations of the $\tilde{\phi}(t-n)$: if $x(t)$ belongs to $\tilde{V}_0$, i.e., if $\hat{x}(u) = \sum_n c_k \tilde{\phi}(u-k)$, then using (8-33), Eq. (8-34) simplifies to $x(t)$. Therefore, only the projected approximation of $x(t)$ onto $\tilde{V}_0$ is recovered. However, since we recover $\hat{x}_n$, we may attempt to reconstruct $x(t)$ (or its projection onto V_0) directly from $\hat{x}_n$.

8.6. THE DWT FOR CWT COMPUTATION

The DWT, as well as WS, are nonredundant transforms. However, it may be useful to obtain samples of the CWT at denser places of the time-scale plane than the dyadic grid. It is, therefore, sometimes appropriate to generalize (8-29) in order to obtain more samples in the time-scale plane. This is especially useful for signal analysis, where one usually "oversamples" the discretization (8-3), in two ways: First, one may want to evaluate the scale output at any time sample, whatever the scale (see section 8.6.2), instead of a coarser sampling when increasing the scale. Then it is often useful to have a finer sampling in scale, in order to obtain, e.g., "M voices per octave" [5] (see section 8.6.1). Finally, one could wish a time-scale parameter sampling as follows:

$$a = a_0^j \tag{8-35}$$

$$b = k \tag{8-36}$$

where $1 < a_0 \leq 2$. Note that a is restricted to positive values. This implicitly assumes that the signal and wavelets are either both real-valued or both complex analytic (i.e., their Fourier transforms vanish for negative frequencies). One interest of (8-35) is the possibility to approximate a nearly continuous CWT representation in the time-scale plane for analysis purposes.

The full discretization previously defined is addressed separately, by working with one parameter sampled as in the WS transform, while the other one takes denser, regular sampling values. The general case is obtained by combining both techniques.

8.6.1 Finer Sampling in Scale

Here, we stay with $b = k2^j$, while the scale parameters are sampled according to

$$a = 2^{j+m/M}, \quad m = 0, \cdots, M-1 \tag{8-37}$$

where m is called the "voice." In other words, a_0 in (8-35) is chosen as an Mth root of 2.

The following simple method [3] allows one to compute WS coefficients on M voices per octave, using the standard "octave-by-octave" algorithm (8-29) as a building block. For each m, replace $\psi(t)$ by the slightly stretched wavelet $2^{-m/2M}\psi(2^{-j+m/M}t)$ in the expression of $\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k)$. The wavelet basis functions become

$$2^{-(j+m/M)/2}\psi(2^{-(j+m/M)}(t - k2^j)), \qquad j, \mathrm{k} \in Z, \qquad m = 0, \cdots, M-1 \tag{8-38}$$

The grid obtained in the time-scale plane (b, a) is shown in Fig. 8-3. Now, a computation on M voices per octave is done by applying the octave-by-octave algorithm M times, with M different prototypes.

Of course, the parameters of each octave-by-octave algorithm must be recomputed for each m using the procedure previously described. Clearly, the whole algorithm requires about M times the computational load of one octave-by-octave algorithm.

This method is certainly not the best one for an "M voices per octave" computation if M is large, because it does not take advantage of the fact that the various prototypes (8-38) are related in a simple manner. It would be more appropriate to devise a method that takes advantage of both time redundancy and scale redundancy (with more scales than in the octave-by-octave case). The algorithm devised by Bertrand et al. in [1] is based on scale redundancy but is suited for another type of computation (see section 8.9.3).

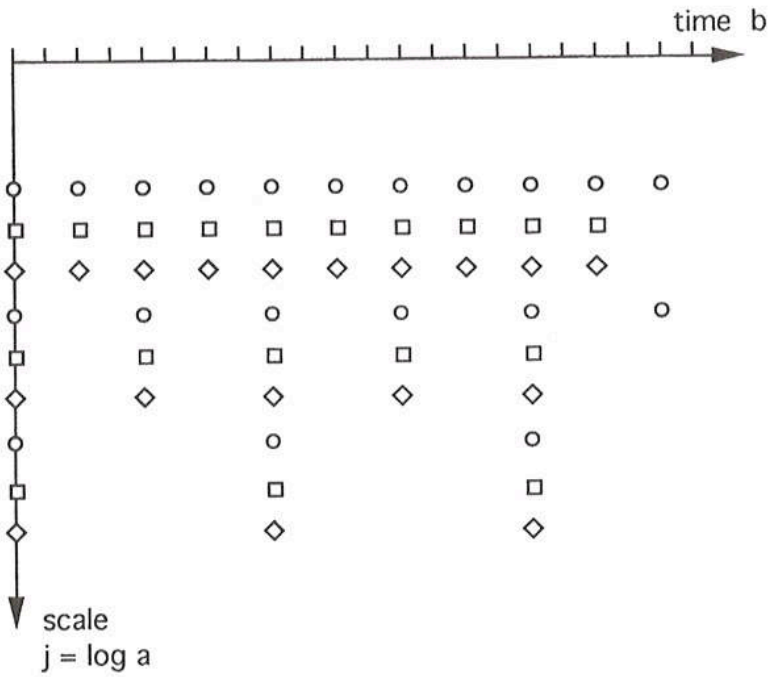

Figure 8-3 Sampling of the time-scale plane corresponding to three voices per octave in a WS. The imbrication of the computation is shown using points labeled by circles, squares, and crosses, which can be computed separately using octave-by-octave DWT algorithms.

8.6.2 Finer Sampling in Time: Modified Shensa and "à trous" Algorithms

Here, we restrict our study to an octave-by-octave computation, i.e., $a = 2^j$, while considering all possible values for the time parameter $b = k$. First, note that the computation of the WS coefficients treated in section 8.5.1 is nothing but part of the computation required here, since

$$C_{j,k} = \mathrm{CWT}\{x(t); 2^j, k2^j\} \tag{8-39}$$

Now, the Shensa algorithm for the WS coefficients can be readily extended to the required computation of $\mathrm{CWT}\{x(t); 2^j, k\}$ [15]. We have a result similar to (8-29), namely,

$$\mathrm{CWT}\{x(t); 2^j, k\} = \mathrm{DWT}\{\hat{x}_n; 2^j, k\} \tag{8-40}$$

where $\hat{x}_n$ is a prefiltered discrete input defined by (8-17), more easily computed using (8-18). The only difference is, of course, that the DWT is computed for all integer values of b, instead of $b = k2^j$, as in the standard description of the DWT. Equation (8-40) indicates that CWT coefficients sampled on an arbitrary grid in the time-scale plane can be computed using a filter bank structure derived from the initial DWT. This fact was mentioned by Gopinath and Burrus in [16] and subsequently discussed in detail by Shensa in [15]: The resulting CWT algorithm was recognized to be identical with the "à trous" algorithm of Holschneider et al. [3,5].

This "à trous" structure is pictured in Fig. 8-4(a). It can be easily derived as follows. For fixed j, the result of (8-40) is simply the discrete input filtered by $h_j(n)$, whose transfer function is given by (8-24). The difference with section 8.4 is the absence of decimation. Now, reorganize the computation in a hierarchical way as follows. The input is iteratively filtered by $G(z)$, $G(z^2)$, and so on. At the jth step, it is enough to filter by $H(z^{2^{j-1}})$ in order to obtain the expected coefficients (8-40), as shown in Fig. 8-4(a). The term "à trous"—with holes—was coined by Holschneider et al. in reference to the fact that only one every 2^{j-1} coefficients is nonzero in the filter impulse responses at the jth octave.

8.6.3 A Slightly Different Building Block

We now consider another variation of filter bank implementation of the CWT—which was also derived by Shensa in [15]—because it is more suited to further reduction of complexity using fast filtering techniques than the one using DWT. Consider the filter bank structure of Fig. 8-4(c), where the elementary cell is depicted in Fig. 8-4(b). This filter bank structure is easily deduced from the one of Fig. 8-4(a) [15].

The advantage of this slightly different structure is easily understood as follows: Consider the computation performed at the first octave ($j = 1$) of Fig. 8-4 and compare it to Fig. 8-1(a). In the latter structure, half the wavelet coefficients required for the CWT at this octave are computed: the missing ones are the outputs of $H(z)$ that are discarded by the decimation process. It is sufficient to remove the subsampling on $H(z)$ to obtain the required wavelet coefficients of the first octave, as shown

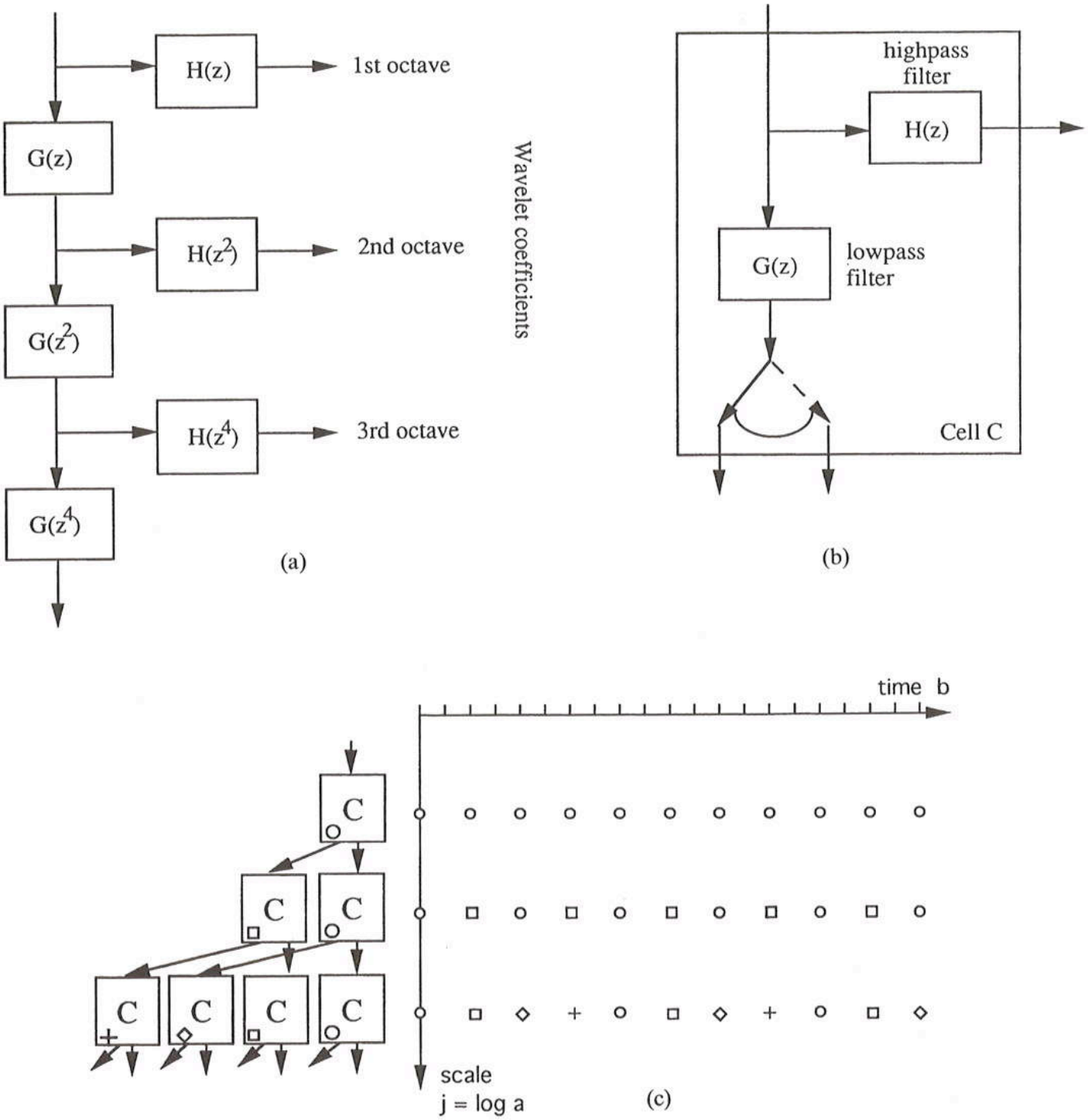

Figure 8-4 (a) "A trous" structure as derived by Holschneider et al; (b) basic computational cell used for computing CWT coefficients octave by octave; (c) connection of the cells used in this paper and corresponding location of the wavelet coefficients in the time-scale plane.

in Fig. 8-4(a). Also, in Fig. 8-1(a), the output of the filter $G(z)$ is used to compute the wavelet coefficients for the next stage ($j = 2$) for even values of the time-shift parameter b. The missing sequence, which allows one to obtain the coefficients with odd values of b is nothing but the discarded subsampled sequence; it is recovered in Fig. 8-4(a).

At the next octave, $j = 2$, both inputs are processed separately using identical cells. One provides the same coefficients as in the WS computation [round dots in

Fig. 8-4(c)], while the other allows one to start a new computation of the same type, shifted in time, and beginning at the next scale [squared dots in Fig. 8.4(c)]. The whole process is iterated as shown in Fig. 8-4(c).

In the overall organization, all outputs of both filters have to be computed, those of $G(z)$ being used to build two interleaved sequences, while those of $H(z)$ are simply the desired samples of the CWT at the given scale. This is in contrast to the basic computational cells of the fast DWT algorithms. Hence the reorganization of the computations described in section 8.7.2 should not be used in this case.

8.6.4 Inner Product Implementation of the CWT

Consider the filter bank implementation of Fig. 8-4(c), and assume that both filters $g(n)$ and $h(n)$ are finite impulse response (FIR) filters and have same length L. When the filters are directly implemented as inner products, the octave-by-octave CWT algorithm requires

$$2L \text{ mults/input point/cell} \qquad 2(L-1) \text{ adds/input point/cell} \tag{8-41}$$

Note that there are 2^{j-1} elementary cells at the jth octave in Fig. 8-4(c), which are identical but "work" at a different rate: a cell at the jth octave is fed by an input which is subsampled by 2^{j-1} compared to the original input $x(t)$. Therefore, the total complexity required by an octave-by-octave CWT algorithm on J octaves, is exactly J times the complexity of one cell. Thus the complexity of any filter bank implementation of a CWT grows linearly with the number of octaves. This results then, for a CWT on J octaves, in

$$2LJ \text{ mults/input point} \qquad 2(L-1)J \text{ adds/input point} \tag{8-42}$$

As mentioned in [3], this is a significant improvement compared to the naive method that would consist of directly implementing the CWT and would not take advantage of the fact that wavelets are easily related by dilation (this direct implementation would require a complexity exponentially increasing with J). Since the whole CWT algorithm requires J times the complexity of one cell, the latter is the total complexity of the CWT per input point and per octave. Hence the complexity of one cell is also the total complexity of the CWT per output point, i.e., per computed wavelet coefficient.

Since the elementary cell contains filters, its arithmetic complexity can be reduced using any fast filtering technique. This is explained in the following section.

8.7. EFFICIENT IMPLEMENTATIONS OF THE DWT

In the following, we derive efficient implementations of the DWT, which can be used to compute WS coefficients using the Shensa algorithm. Hence most of the content of this section also applies to the implementation of tree-structured two-band filter banks iterated on the low-pass filter.

8.7.1 Preliminaries

It is important to note that the standard DWT algorithm, implemented directly as a filter bank, is already "fast." This fact was mentioned by Ramstad and Saramaki in the context of octave-band filter banks [22]. What makes the DWT "fast" is the decomposition of the computation into elementary cells and the subsampling operations (decimations), which occur at each stage. More precisely, the operations required by one elementary cell at the jth octave [Fig. 8-1(a)] are counted as follows. There are two filters of equal length L involved. The "wavelet filtering" by $h(n)$ directly provides the wavelet coefficients at the considered octave, while filtering by $g(n)$ and decimating is used to enter the next cell. A direct implementation of the filters $g(n)$ and $h(n)$ followed by decimation requires $2L$ multiplications and $2(L-1)$ additions for every set of two inputs. That is, the complexity per input point for each elementary cell is

$$L \text{ mults/point/cell} \quad \text{and} \quad L-1 \text{ adds/point/cell} \tag{8-43}$$

Since the cell at the jth octave has input subsampled by 2^{j-1}, the total complexity required by a filter bank implementation of the DWT on J octaves is $(1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^{J-1}}) = 2(1-2^{-J})$ times the complexity (8-43). That is,

$$2L(l-2^{-J})\text{mults/point} \quad \text{and} \quad 2(L-1)(1-2^{-J})\text{adds/point} \tag{8-44}$$

The DWT is therefore roughly equivalent, in terms of complexity, to one filter of length $2L$. Note that the complexity remains bounded as the number of octaves, J, increases [22].

In contrast, a naive computation of the DWT, which would implement (8-7) exactly as written, with precomputed discrete wavelets $h_j(n)$, would be very costly. This lack of efficiency is due to the fact that (8-7) does not take advantage of the dilation property of wavelets, summarized by the two-scale difference equation: Since the length of $h_j(n)$ is $(L-1)(2^J-1)+1$, one would have, at the jth octave, $(L-1)(2^j-1)+1$ real multiplications and $(L-1)(2^j-1)$ real additions for each set of 2^j inputs. For a computation on J octaves ($j=1,\cdots,J$), this gives

$$J(L-1)+1 \text{ mults/point} \quad \text{and} \quad J(L-1) \text{ adds/point} \tag{8-45}$$

This complexity increases linearly with J, while that of the "filter bank" DWT algorithm is bounded as J increases. The use of the filter bank structure in the DWT computation thus reduces the complexity from JL to L. This is a huge gain; the DWT already deserves the term "fast."

8.7.2 Reorganization of the Computations

The derivation of faster algorithms described in section 8.8 is primarily based on the reduction of computational complexity. Here, "complexity" means the number of real multiplications and real additions required by the algorithm, per input point. In the DWT case, this is also the complexity per output point since the DWT is critically sampled. Of course, complexity is not the only relevant criterion. For example, regular computational structures (i.e., repeated application of identical

computational cells) are also important for implementation issues. However, since most algorithms considered in this paper have regular structures, a criterion based on complexity is fairly instructive for comparing the various DWT algorithms. We have chosen the total number of operations (multiplications + additions) as the criterion. With today's technology, this criterion is generally more useful than the sole number of multiplications [23], at least for general-purpose computers (another choice would have been to count the number of multiplication-accumulations, for implementation on digital signal processors).

From the operation counts given earlier (8-44), it is clear that if all elementary cells require the same complexity, then a filter bank implementation of the DWT requires $2(1-2^J)$ times the complexity of one cell. Therefore, any fast convolution technique applied to the elementary cell will further reduce the computational load of the DWT. Section 8.8 proposes two classes of fast algorithms: one based on the fast Fourier transform (FFT) [24] and the other on short-length FIR filtering algorithms [23].

The basic DWT elementary cell, depicted in Fig. 8-1(a), contains two filters. However, they are always followed by subsampling (or decimation), which discards every other output. It is well known that reducing the arithmetic complexity of an FIR filter implementation is obtained by gathering the computations of several successive outputs [24]. Since the filter outputs are decimated in Fig. 8-1(a), it is necessary to reorganize the computations in such a way that "true" filters appear. To do this, we apply a biphase decomposition, [17] to all signals involved, which consists of separating them into even- and odd-indexed sequences. The biphase decomposition expresses the z-transform of the input sequence x_n as:

$$X(z) = \sum_n x_n z^{-n} \tag{8-46}$$

$$= X_0(z^2) + z^{-1} X_1(z^2) \tag{8-47}$$

where $X_0(z) = \sum_n x_{2n} z^{-n}$ and $X_1(z) = \sum_n x_{2n+1} z^{-n}$.

Similarly, apply the biphase decomposition to the L-tap filters $G(z)$ and $H(z)$ involved in the computation. The cell output $Y(z)$ that enters the next stage is obtained by first filtering by $G(z)$, then subsampling. Picking out the even part of $G(z)X(z)$ results in

$$Y(z) = G_0(z)X_0(z) + z^{-1}G_1(z)X_1(z) \tag{8-48}$$

Now that this rearrangement has been made, the output $Y(z)$ is obtained differently: First the even- and odd-indexed input samples $X_0(z)$ and $z^{-1}X_1(z)$ are extracted as they flow by (hence, the delay factor z^{-1} for odd-indexed samples). Then, $L/2$-tap filters $G_0(z)$ and $G_1(z)$ are applied to the even and odd sequences, respectively. Finally, the results are added together. The other output of the elementary cell (the one corresponding to the filter $H(z)$) is obtained similarly using $H_0(z)$ and $H_1(z)$.

The resulting flow graph is depicted in Fig. 8-5 (the corresponding IDWT cell is simply obtained by flow graph transposition). Compare with Fig. 8-1(a): there are now four "true" filters of length $L/2$, whose impulse responses are the decimated

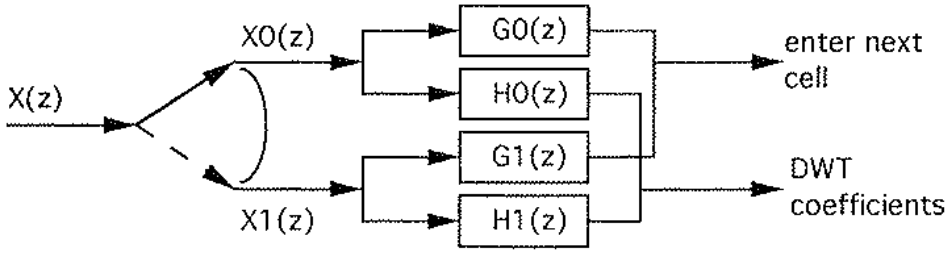

Figure 8-5 Rearrangement of the DWT cell of Fig. 8-1(a) that avoids subsampling, hence allows the application of fast filtering techniques.

initial filters $G(z)$ and $H(z)$. The complexity has not changed, but the resulting structure is easily improved by the use of classical fast filtering algorithms, as shown in the next section.

8.8. FASTER DWT ALGORITHMS

The aim of this section is to further reduce the computational load of the DWT. We briefly motivate this with a brief analogy to fast filtering. FFTs are used for implementing long filters (typically $L \geq 64$) because they greatly reduce the complexity: Compared to a direct implementation of the filter, the number of operations per input point is reduced from L to $\log_2 L$, hence the term "fast." For short filters, however, the FFT is no longer efficient and other fast filtering techniques are used [23,24]; the resulting gain is fairly modest, but still interesting when heavy computation of short filters is required, provided that the accelerated algorithm does not require a much more involved computation compared to the initial one. The situation of the DWT is identical: using FFTs, the complexity of the DWT can be reduced from $2L$ to $4 \log_2 L$, when the filter length L is large. However, DWTs have been mostly used with short filters so far (although nothing ensures that this will last forever). For them, using different techniques, smaller gains are obtained, typically a 30% saving in the number of computations, which can still be useful.

We assume real data and filters (of finite length), but the results extend easily (if necessary) to the complex-valued case. A quick evaluation of the corresponding number of operations can be obtained from the results provided in the following real-valued case: the FFT-based algorithms described next require about twice as many multiplications in the complex case as in the real case, a property shared by FFT algorithms [24]. However, a straightforward filter bank implementation of the DWT (Fig. 8-1), or the "short-length" algorithms described in section 8.8.3, require about three times as many multiplications in the complex case, assuming that a complex multiplication is carried out with three real multiplications and additions [24].

We shall not derive algorithms explicitly for the inverse DWT. However, an inverse DWT algorithm is easily obtained from a DWT algorithm as follows: If the wavelets form an orthogonal basis, the exact inverse algorithm is obtained by taking the Hermitian transpose of the DWT flowgraph. Otherwise, only the structure of the inverse algorithm is found that way, the filter coefficients $g(n), h(n)$ have to be

replaced by $\tilde{g}(n), \tilde{h}(n)$, respectively. In both cases, any DWT algorithm, once transposed, can be used to implement an inverse DWT. It can be shown that this implies that the DWT and inverse DWT require exactly the same number of operations (multiplications and additions) per point.

The filters involved in the computation of the DWT (cf. Fig. 8-1) usually have equal length L. This is true in the orthogonal case, while in the biorthogonal case the filter lengths may differ by a few samples only. Although an implementation of "Morlet-type" wavelets given in [3,5] uses a short low-pass filter $g(n)$ and a long high-pass filter $h(n)$, we restrict our focus in this section to the case of equal filter lengths for simplicity. If lengths differ, one can pad the filter coefficients with zeros.

8.8.1 An FFT-Based DWT Algorithm

This method consists of computing the four $L/2$-tap filters of Fig. 8-5 using the overlap-add or overlap-save FFT. Operation counts are done using the "split radix" FFT algorithm which, among all practical FFT algorithms, has the best known complexity for lengths that are powers of 2: $N = 2^n$ ($n = \log_2 N$ should not be confused here with the sample index n). For real data, the split radix FFT (or inverse FFT) requires exactly

$$2^{n-1}(n-3)+2 \quad \text{(real) mults} \tag{8-49}$$

$$2^{n-1}(3n-5)+4 \quad \text{(real) adds} \tag{8-50}$$

We now briefly recall the standard method for computing filters using the FFT. The input of the DWT cell is blocked B samples by B samples (the decimated sequences input to the filters therefore flow as blocks of length B/2). Each discrete filter is performed by computing the inverse FFT (IFFT) of the product of the FFTs of the input and filter. Since the latter FFT can be precomputed once and for all, only one IFFT and one FFT are required per block for one filter. However, this results in a cyclic convolution [24], and the overlap-add and overlap-save methods [24] can be used in order to avoid wraparound effects. One is the transposed form of the other and both require exactly the same complexity. For one filter of length $L/2$, with input block length $B/2$, wraparound effects are avoided if the FFT length N satisfies $N \geq L/2 + B/2 - 1$. Here, we assume $B = 2N - (L-2)$.

Assume that each elementary cell has the same structure, pictured in Fig. 8-6. The input is first split into even- and odd-indexed sequences. Then, a length-N FFT is performed on each decimated input, and four frequency-domain convolutions are performed by multiplying the (Hermitian symmetric) FFT of the input by the (Hermitian symmetric) FFT of the filter. This requires $4N/2$ complex multiplications for the four filters. Finally, two blocks are added ($2N/2$ additions) and two IFFTs are applied. Assuming that a complex multiplication is done with three real multiplications and three real additions [24], this gives a total of

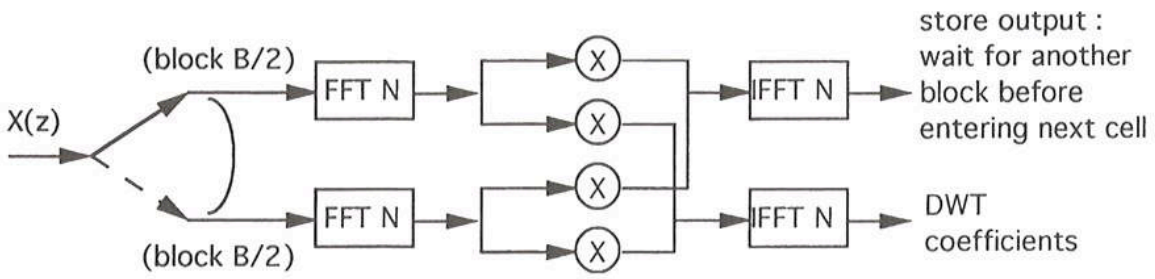

Figure 8-6 FFT-based implementation of the DWT cell of Fig. 8-4. Overlap-add (or overlap-save) procedure is not explicitly shown.

$$\frac{n2^{n+1}+8}{2^{n+1}-(L-2)} \quad \text{mults/point/cell} \tag{8-51}$$

$$\frac{(3n-1)2^{n+1}+16}{2^{n+1}-(L-2)} \quad \text{adds/point/cell} \tag{8-52}$$

Note that for a given length L, there is an optimal value of N that minimizes the complexity. Tables 8-1 and 8-2 show the resulting minimized complexities for different lengths L in comparison with the inner product implementation of the filter bank. The comparison is clearly in favor of the FFT version of the DWT algorithm for medium to large filter lengths ($L \geq 16$). The asymptotic gain brought by the FFT-based DWT algorithm is about $L/(2\log_2 L)$. However, as seen in Table 8-1, the FFT implementation of the DWT is not effective for short filters.

There is a subtlety to keep in mind when wraparound effects at the cell output are eliminated in the time-domain. One could immediately take the output blocks

TABLE 8-1: FFT-Based DWT Algorithms: Arithmetic Complexity Per Point and Per Octave

Filter Length	Inner Product Filter Bank	FFT-Based Algorithm	Vetterli, 2 Octaves Merged	Vetterli, 3 Octaves Merged	Vetterli, 4 Octaves Merged
2	2 + 1	3 + 6 (2)	3.17 + 5.83 (2)	3.07 + 6.07 (4)	3.17 + 6.17 (4)
4	4 + 3	4 + 9.33 (4)	4.56 + 10.97 (16)	5.17 + 12.43 (32)	5.58 + 14.00 (128)
8	8 + 7	5.23 + 14.15 (16)	5.68 + 14.67 (64)	6.10 + 15.33 (128)	6.61 + 16.90 (256)
16	16 + 15	6.56 + 18.24 (32)	6.61 + 17.41 (128)	6.88 + 18.10 (512)	7.25 + 19.06 (1024)
32	32 + 31	7.92 + 22.37 (64)	7.50 + 20.05 (256)	7.56 + 20.14 (1024)	7.90 + 21.01 (2048)
64	64 + 63	9.12 + 26.20 (256)	8.25 + 22.55 (1024)	8.23 + 22.13 (2048)	8.54 + 22.90 (4096)
128	128 + 127	10.27 + 29.67 (512)	9 + 24.79 (2048)	8.89 + 24.10 (4096)	9.16 + 24.76 (8192)

Each entry gives the number of operations per input or output point in the form *mults* + *adds*, and the initial FFT length. Complexities should be multiplied by $2(1-2^{-J})$ for a computation of the DWT on J octaves.

TABLE 8-2: Arithmetic Complexity Per Point and Per Cell: DWT Algorithms

Filter Length L	Straightforward Filter Bank	FFT-Based Algorithm	Short Length Algorithm
4	$4+3$	$4+9.33$ (4)	$3+4$ (2)
6	$6+5$	$4.67+12$ (8)	$4+6.3$ (3)
8	$8+7$	$5.23+14.15$ (16)	$4.5+8.5$ (2×2)
10	$10+9$	$5.67+15.33$ (16)	$4.8+14.2$ (5)
12	$12+11$	$6.18+16.73$ (16)	$6+12$ (2×3)
16	$16+15$	$6.56+18.24$ (32)	$9+13$ (2×2)
18	$18+17$	$6.83+19$ (32)	$8+17$ (3×3)
20	$20+19$	$7.13+19.83$ (32)	$7.2+21.4$ (5×2)
24	$24+23$	$7.32+20.68$ (64)	$12+18$ (2×3)
30	$30+29$	$7.76+21.92$ (64)	$9.6+27$ (5×3)
32	$32+31$	$7.92+22.37$ (64)	$18+22$ (2×2)

Each entry gives the number of operations per input or output point in the form *mults* + *adds*, and either the FFT length or the type of fast-running FIR algorithm used. Complexities should be multiplied by $2(1-2^{-J})$ for a computation of the DWT on J octaves.

(now of length $B/2$ instead of B) as inputs to the next cell, but this would halve the block length at each stage. This method is not effective eventually because the FFT is most efficient for an optimized value of the block length B (at fixed filter length L). It is therefore advisable to work with the same optimized degree of efficiency at each cell, by waiting for another block before entering the next cell, so that each cell has the same input block length B and FFT length N. This method involves strictly identical cells: they not only have the same computational structure, but they also process blocks of equal length. As usual, the resulting total complexity of the DWT is $2(1-2^{J})$ times the complexity of one cell, as shown in section 8.7.1.

8.8.2 A Generalization: The Vetterli Algorithm

The FFT-based DWT algorithm just described can be improved by gathering J_0 consecutive stages, using a method due to Vetterli (originally in the filter bank context [25], and then applied to the computation of the DWT [12]). The idea is to avoid subsequent IFFT's and FFT's by performing the subsampling operation in the frequency domain. This is done by inverting the last stage of a decimation-in-

time radix-2 FFT algorithm. The FFT length is then necessarily halved at each DWT stage, whereas the filter lengths remain constant, equal to $L/2$.

Unfortunately, this class of algorithms has two major limitations. First, the structure of computations is less regular than for the simple FFT algorithm of the preceding section because FFTs have different lengths. Second, the relative efficiency of an FFT scheme per computed point decreases at each stage.

Table 8-1 lists the resulting complexities for $J_0 = 2, 3$, and 4, minimized against $N = 2^n$. Vetterli algorithms are more efficient than the initial FFT-based computation of the DWT ($J_0 = 1$) only for long filters ($L \geq 32$) and small J_0. Efficiency is lost in any case when J_0 is greater than 3.

8.8.3 DWT Algorithms for Short Filters

We have seen that for small filter lengths ($L < 16$), FFT-based algorithms do not constitute an improvement compared to the initial filter bank computation. Therefore, it is appropriate to design a specific class of fast algorithms for short filters. Here, "fast running FIR" algorithms [23] are applied to the DWT computation. The class of "fast running FIR algorithms" is interesting because the multiply/accumulate structure of computations is partially retained, hence these algorithms are very efficiently implemented [23].

A detailed description of fast running FIR algorithms can be found in [23]. Basically, a filter of length L is implemented as follows. The involved sequences (input, output, and filters) are separated into subsequences, decimated with some integer ratio R. Assuming L is a multiple of R, filtering is done in three steps:

1. The input is decimated and the resulting R sequences are suitably combined, requiring A_i additions per point, to provide M subsampled sequences.
2. The resulting sequences serve as inputs to M decimated subfilters of length L/R.
3. The outputs are recombined, with A_o additions per point, to provide the exact decimated filter outputs.

Fig. 8-7 provides an example for $R = 2$, $A_i = 2$, $M = 3$, and $A_o = 2$. Other algorithms derived in [23] were also applied, corresponding to $R = 3$ and $R = 5$.

This computation can be repeated: the subfilters of length L/R are still amenable to further decomposition. For example, in order to implement a 15-tap filter, one can either use a fast running FIR algorithm for $R = 3$ or $R = 5$, or decompose this filter by a "3×5 algorithm," which first applies the procedure with $R = 3$, then again decomposes the subfilters using the procedure associated with $R = 5$. Alternatively, a "5×3 algorithm" can be used. Each of these algorithms yields different complexities, which are discussed in detail in [23]. The short-length DWT algorithm is derived as follows. One applies fast running FIR algorithms to the four filters of length $L/2$ in the elementary cell of the DWT (Fig. 8-5). Here, since two

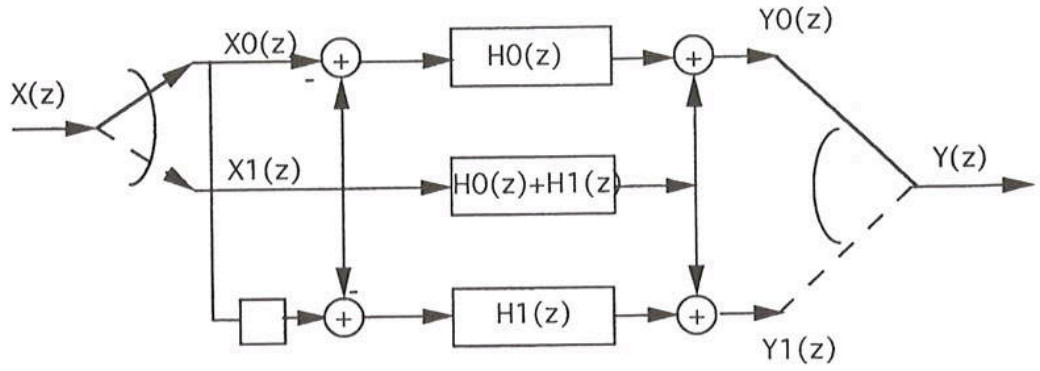

Figure 8-7 Simple example of fast-running FIR filtering algorithm with decimation ratio $R = 2$ [22]. Subscripts 0 and 1 indicate biphase decomposition.

pairs of filters share the same input, all preadditions can be combined together on a single input.

Table 8-2 lists the resulting complexities, using the fast running FIR algorithm that minimizes the criterion (multiplications + additions). When two different decompositions yield the same total number of operations, we have chosen the one that minimizes the number of multiplications. Table 8-2 shows that short-length DWT algorithms are more efficient than the FFT-based DWT algorithms for lengths up to $L = 18$.

Since, in practice, DWTs are generally computed using short filters [8,9], the short-length algorithms probably give the best practical alternative when heavy DWT computation is required. As an example, for $L = 18$, the short-length algorithm requires a total of 25 operations per point instead of 35 for the direct method.

8.8.4 Other Considerations

- *The Orthogonal Case:* In our derivations, we did not take advantage of orthogonality constraints [5,8–11,13] so as to be as general as possible. However, orthogonality is worthy of consideration because of its simplicity: the analysis and synthesis filters coincide (within time reversal and complex conjugation). Furthermore, it allows one to further reduce the complexity of the DWT: Using a lattice implementation of the DWT filter bank cell of Fig. 8-1(a), Vaidyanathan has shown [17] that the complexity can be reduced by a factor of 50% in the orthogonal case.
  Whether or not this reduction can be attained while preserving the inner products (unlike the lattice structure implementation) is an open problem. In any case, Tables 8-1 and 8-2 do not provide a fair and detailed comparison between various algorithms in the orthogonal case.
- *Unequal Filter Lengths:* In the previous derivations, we have restricted ourselves to filters of equal lengths for simplicity. However, it may happen that one uses a low-pass interpolation filter $g(n)$ of small length ($L_g \ll 16$) and a very long high-pass filter $h(n)$ of length $L_h \gg 16$. This is the case in [3,5],

where one typically uses a first-order interpolation filter $g(n)$ ($L_g = 3$) to approximate the "Morlet wavelet," a modulated Gaussian.
Obviously, for a direct implementation of the DWT filter bank, it is in this case absurd to assume equal filter lengths since the complexity then becomes $(L_g + L_h)/2$ mults and $(L_g + L_h - 1)/2$ adds.
However, FFT-based DWT algorithms are still efficient when one of the filters is very long. In this case, some efficiency of FFT-based algorithms is lost, but they still yield a substantial gain over a standard, straightforward filter bank implementation of the DWT. As an example, for a wavelet of length $L_h = 64$ and interpolation filters of length $L_g = 3, 7$, and 11, the FFT-based algorithms give respective gains over a standard DWT of 46.9%, 49.9% and 52.6%.

- *Linear phase:* In the previous discussion, we did not take other properties of filters into account, such as the linear phase property, which holds for the Morlet wavelet. In this case, rather than using involved fast algorithms, we recommend straightforward use of the symmetry in the inner product implementation of the algorithms, which cuts by 2 the number of multiplications.

8.8.5 Faster CWT Algorithms

The same fast convolution tools can be applied on the CWT, slightly modified building block described in section 8.6.2. The main difference is that the filters involved are comparatively twice as long as in the WS case, due to the absence of decimation. This increases the efficiency of the "faster" algorithms. Being applications of the same techniques, they are not described, but the arithmetic complexities are given in Table 8-3, in order to allow the reader to evaluate their potential compared to straightforward inner product implementation.

8.9. OTHER ALGORITHMS FOR CWT COMPUTATION

Several algorithms for computing CWT coefficients, which differ notably from those already described, have been proposed recently (see e.g., [1,16,26–28]). Several of them are outlined in this section.

8.9.1 Reproducing Kernels

Gopinath and Burrus [16] proposed a method that also uses DWTs. The signal is assumed to be completely determined from its WS coefficients. Therefore, these alone can be used to compute all CWT coefficients by some reproducing kernel equation. The introduction of an auxiliary wavelet moreover allows one to precompute the kernel and to obtain a method particularly suited to the computation of CWT coefficients with respect to several wavelets. However, the kernel expansion in [16] seems to be computationally expensive.

TABLE 8-3: Arithmetic Complexity Per Computed Point for Various CWT Algorithms

Filter Length L	Straightforward Filter bank	FFT-Based Algorithm	FFT-Based (2 Octaves Merged)	Short Length Algorithm
2	4 + 2	4 + 10 (4)	4.8 + 12 (16)	3 + 3 (2)
3	6 + 4	5 + 14 (8)	5.8 + 15.2 (32)	4 + 5.3 (3)
4	8 + 6	6 + 16.8 (8)	6.5 + 17.2 (32)	4.5 + 7.5 (2 × 2)
5	10 + 8	6.5 + 19 (16)	6.9 + 18.7 (64)	4.8 + 13.2 (5)
6	12 + 10	7.1 + 20.7 (16)	7.3 + 19.8 (64)	6 + 11 (2 × 3)
8	16 + 14	7.9 + 23.5 (32)	7.8 + 21.6 (128)	9 + 12 (2 × 2)
9	18 + 16	8.2 + 24.5 (32)	8.1 + 22.3 (128)	8 + 16 (3 × 3)
10	20 + 18	8.6 + 25.6 (32)	8.3 + 22.9 (128)	7.2 + 20.4 (5 × 2)
12	24 + 22	9.2 + 27.4 (64)	8.6 + 24.2 (256)	12 + 17 (2 × 3)
15	30 + 28	9.7 + 29 (64)	9 + 25.2 (256)	9.6 + 26 (5 × 3)
16	32 + 30	9.9 + 29.6 (64)	9.1 + 25.5 (256)	18 + 21 (2 × 2)
18	36 + 34	10.3 + 30.9 (64)	9.4 + 26.3 (256)	16 + 24 (3 × 3)
20	40 + 38	10.6 + 31.8 (128)	9.6 + 27 (512)	14.4 + 27.6 (5 × 2)
24	48 + 46	11 + 33 (128)	9.8 + 27.8 (512)	24 + 29 (2 × 3)
25	50 + 48	11.1 + 33.3 (128)	9.9 + 27.9 (512)	11.5 + 44.9 (5 × 5)
27	54 + 52	11.3 + 34 (128)	10 + 28.3 (512)	24 + 32 (3 × 3)
30	60 + 58	11.7 + 35 (128)	10.2 + 28.9 (512)	19.2 + 35.6 (5 × 3)
32	64 + 62	11.9 + 35.7 (128)	10.4 + 29.4 (512)	36 + 39 (2 × 2)
64	128 + 126	13.7 + 41.1 (512)	11.6 + 33.1 (2048)	72 + 75 (2 × 2)
128	256 + 254	15.4 + 46.2 (1024)	12.7 + 36.4 (4096)	144 + 147 (2 × 2)

Each entry gives the number of operations per computed coefficient (i.e., per input point per octave) in the form *mults* + *adds*, and either the FFT length or the type of fast-running FIR algorithm used.

8.9.2 Algorithms Using Splines

We have already emphasized the importance of splines for WT computation in section 8.2.2. In fact, there is another remarkable property which makes them useful

for CWT computation: B-splines of degree d follow a generalized two-scale difference equation (8-12) (generalized to an m-scale equation), valid for any $m > 0$ if d is odd, and m odd only if d is even:

$$\beta^d(t/m) = \sum_{k \in Z} c_k^{m,d} \beta^d(t-k). \tag{8-53}$$

with $c_k^{m,d}$ defined (by identification) as

$$\sum c_k^{m,d} z^{-k} = \frac{z^{(d+1)(m+1)/2}}{m^d} \left(\sum_{k=0}^{m-1} z^{-k} \right)^{d+1} \tag{8-54}$$

This has led Unser *et al.* [28], following a generalization of Shensa's algorithm, to use B-splines in order to compute a DTWT in which the scale parameter a can take any integer value. While this property is the key to an increased flexibility, the fast algorithm is obtained following the same steps as in the Shensa algorithm:

- First, approximate the input signal as its spline approximations of degree d_1:

$$x(t) = \sum_{k \in \mathbf{Z}} x_k \beta^{d_1}(t-k) \tag{8-55}$$

- Then, specify the wavelet by its B-spline expansion of degree d_2:

$$\psi(t) = \sum_{k \in Z} p_k \beta^{d_2}(t-k) \tag{8-56}$$

  Thus due to the generalized two-scale difference equation, the wavelet, when expanded by a factor m, can be expressed as

$$\psi(t/m) = \sum_{k \in \mathbf{Z}} (\{[p]_{\uparrow m}\} * \{c^{m,d_2}\})(k) \beta^{d_2}(t-k) \tag{8-57}$$

  where $(\{[p]_{\uparrow m}\} * \{c^{m,d_2}\})(k)$ denotes the kth term of the convolution of sequences p_l, as defined in (8-56), upsampled by a factor m, and of sequence c_l^{m,d_2}, as defined in (8-54).
- Finally, the CWT of $x(t)$ at scale m is given by

$$\mathrm{CWT}\{x(t), m, b\} = \sum_{k \in \mathbf{Z}} (\{[p]_{\uparrow m}\} * \{c^{m,d_2}\} * \{x_k\})(k) \beta^{d_1+d_2+1}(b-k) \tag{8-58}$$

  which, when evaluated at integer time samples, simplifies to:

$$\mathrm{CWT}\{x(t), m, k\} = (\{[p]_{\uparrow m}\} * \{c^{m,d_2}\} * \{b^{d_1+d_2+1}\} * \{x_k\})(k) \tag{8-59}$$

  where $\{b^{d_1+d_2+1}\}$ is the discrete B-spline of order $d_1 + d_2 + 1$.

The filter bank at work in the algorithm has very simple low-pass filters owing to the special structure of B-splines. As seen from (8-54), they are iterated discrete convolutions of moving sums, and therefore can be computed without any multiplication. This remarkable feature thus results in very efficient algorithms.

8.9.3 Mellin-Transform–Based Algorithms

Another beautiful CWT algorithm, which uses the scaling property of wavelets $\psi(t) \to a^{-1/2}\psi(t/a)$ rather than the convolutional form of (8-1), (8-2) has been proposed by Bertrand et al. [1]. This algorithm makes use of some redundancy between the computations of the various scales of a signal around some time location, while the previously described algorithms make use of redundancy between the computations of several successive outputs of the same scale.

This algorithm is briefly outlined here. Write (8-2) in the frequency domain, assuming that the signal $x(t)$ and wavelet $\psi(t)$ are complex analytic. This gives

$$\mathrm{CWT}\{x(t);\, a, b\} = \int_0^\infty X(f) e^{2i\pi fb} \sqrt{a}\psi^*(af) df \tag{8-60}$$

where $X(f) = \int x(t) e^{-2i\pi ft} dt$ and $\psi(f)$ are the Fourier transforms of $x(t)$ and $\psi(t)$, respectively. Then perform the changes of variable $\phi = \ln f$. A correlation form in $\alpha = \ln a$ appears in the integral.

$$\mathrm{CWT}\{x(t);\, a, b\} = \int_{\mathbf{R}} X(e^\phi) e^{\phi/2} e^{2i\pi e^\alpha b} \psi(e^{\alpha+\phi}) e^{\frac{\alpha+\phi}{2}} d\phi \tag{8-61}$$

After suitable discretization, this correlation can be performed using an FFT algorithm. As stated in [1], the Mellin transform, $M_x(\beta)$ of $x(t)$, plays a central role, since it turns out to be exactly the inverse Fourier transform of $\sqrt{f}X(f)$ in the variable $\phi = \ln f$:

$$M_x(\beta) = \int_{f>0} X(f) f^{-1/2+2i\pi\beta} df \tag{8-62}$$

$$= \int e^{\phi/2} X(e^\phi) e^{2i\pi\beta\phi} d\phi \tag{8-63}$$

As a result, the FFTs involved in the computation of (8-61) are "discrete Mellin transforms," as defined in [1].

This algorithm requires the precomputation of the whole Fourier transform of $x(t)$, which makes a running implementation (in case of infinite duration signals) cumbersome. To overcome this difficulty, we propose a variation on the Bertrands–Ovarlez algorithm, based on the time domain rather than on the frequency domain. Assume that the signal and wavelets are causal (i.e., supported by $t \geq 0$), and make the change of variable $\tau = \ln t$ in (8-2). One obtains a convolution in $\alpha = \ln a$:

$$\mathrm{CWT}\{x(t);\, a, b\} = \int e^{\tau/2} x(e^\tau + b) e^{(\tau-\alpha)/2} \psi^*(e^{\tau-\alpha}) d\tau \tag{8-64}$$

The CWT coefficients are obtained, for a given b, by discretizing the convolution (8-64), resulting in a discrete filtering operation that can be implemented for running data.

Both algorithms (8-61), (8-64) have common characteristics. Some of them can be considered as drawbacks: First, they involve a geometric sampling of either $X(f)$ or $x(t)$. Second, the approximation error made by discretizing (8-61) or (8-64) is difficult to estimate. Finally, in contrast to the octave-by-octave CWT implementa-

tion previously described, the time shift structure of b has completely disappeared, and the input has to be recomputed for each value of b. As a result, the complexity of such algorithms (about two FFTs of length $2JM$ per input point, where J is the number of octaves and M is the number of voices per octave) is found higher than the one obtained for the more classical algorithms described earlier.

However, a nice property of the Mellin-based algorithms is that the CWT coefficients are computed for all desired values of $\ln a$ at the same time (for given value of b), while the efficiency of the classical algorithms requires the computation of long signals. It makes the Bertrands-Ovarlez algorithms very useful when a "zoom," or a refinement, of the wavelet analysis in a short extent around some time location b is desired.

8.10. CONCLUSION

This chapter has reviewed several methods for efficiently implementing various kinds of wavelet transforms, from the fully discrete version to the fully continuous one, and for any type of wavelet.

Emphasis has been put on the various approximations required for the algorithms to be efficient, and on their link with multiresolution analysis. As a result, prefiltering the signal allows one to use the DWT as an intermediate computation for any type of wavelet transform. Guidelines were given for the design of the appropriate prefilter.

Fast DWT algorithms were derived for computing WS coefficients and were modified to compute wavelet coefficients with oversampling in the time-scale plane ("CWT algorithms").

While the inner product implementation of these transforms is already efficient, a further improvement has been obtained by using fast convolution algorithms, adapted to the situation. The availability of both FFT-based and fast-running-FIR-based algorithms allows one to reduce the complexity of the existing algorithms in any case of interest. Tables are provided for the reader to evaluate whether the decrease in computation is worth the complexity of the implementation.

Other fast algorithms were also outlined, either using splines, or using discrete Mellin transforms, each one offering specific advantages: The splines-based algorithms can easily approximate some given wavelet, while still allowing a fast implementation. Mellin-based transforms are more suited to the situation where one is able to sample the signal in a geometric manner (either in the time or in the frequency domain), in which case the redundancy between all scales can efficiently be exploited.

REFERENCES

[1] J. Bertrand, P. Bertrand, and J. P. Ovarlez, "Discrete Mellin transform for signal analysis," in *Proc. 1990 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Albuquerque, NM, April 3–6, 1990, pp. 1603–1606.

[2] P. Goupillaud, A. Grossmann, and J. Morlet, "Cycle-octave and related transforms in seismic signal analysis," *Geoexploration*, vol. 23, pp. 85–102, 1984/85.

[3] M. Holschneider, R. Kronland-Martinet, J. Morlet, and Ph. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in [5], pp. 286–297.

[4] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*, Englewood Cliffs, NJ: Prentice Hall, 1995.

[5] J. M. Combes, A. Grossmann, and Ph. Tchamitchian, Eds., *Wavelets, Time-Frequency Methods and Phase Space*, Berlin: Springer, IPTI, 1989.

[6] Y. Meyer Ed., *Wavelets and Applications*, Paris: Masson/Berlin: Springer Verlag, 1992.

[7] O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Trans. Inform. Thoery*, vol. 38, pp. 569–586, March 1992.

[8] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 674–693, July 1989.

[9] S. Mallat, "Multifrequency channel decompositions of images and wavelet models," *IEEE Trans. Acoust., Speech, Signal Process*, vol. 37, pp. 2091–2110, December 1989.

[10] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Comm. Pure Applied Math.*, vol. 41, no. 7, pp. 909–996, 1988.

[11] Y. Meyer, *Ondelettes et Operateurs*, Tome 1. Paris: Herrmann, 1990.

[12] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," *IEEE Trans. Acoust., Speech, Signal Process*, vol. SP-40, pp. 2207–2232, 1992.

[13] G. Evangelista, "Orthogonal wavelet transforms and filter banks," presented at Proc. 23rd Asilomar Conf., IEEE, November 1989.

[14] A. Cohen, I. Daubechies, and J. C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Comm. Pure Applied Math.*, vol 45, pp. 485–560, 1992.

[15] M. J. Shensa, "Affine wavelets: Wedding the Atrous and Mallat algorithms," *IEEE Trans. Signal Proc.*, vol. 40, pp. 2464–2482, October 1992.

[16] R. A. Gopinath and C. S. Burrus, "Efficient computation of the wavelet transforms," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Albuquerque, NM, April 3–6, 1990, pp. 1599–1601 .

[17] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, NJ: Prentice Hall, 1993.

[18] P. Abry and P. Flandrin, "On the initialization of the discrete wavelet transform algorithm," *IEEE Sig. Proc. Letters*, vol. 1, pp. 32–34, February 1994.

[19] M. Unser and A. Aldroubi, "A general sampling theory for non ideal acquisition devices," *IEEE Trans. Signal Proc.*, vol. 42, pp. 2915–2925, November 1994.

[20] M. J. T. Smith and T. P. Barnwell, "Exact reconstruction for tree-structured subband coders," *IEEE Trans. Acoust., Speech, Signal Process*, vol. ASSP-34, pp. 434–441, June 1986.

[21] I. Daubechies and J. C. Lagarias, "Two-scale difference equations 1. Existence and global regularity of solutions," *SIAM J. Math. Anal.*, vol. 22, no. 5, pp. 1388–1410, September 1991.

[22] T. A. Ramstad and T. Saramah, "Efficient multirate realization for narrow transition-band FIR filters," in *IEEE 1988 Int. Symp. Circ. Syst.*, 1988, pp. 2019–2022.

[23] Z. J. Mou and P. Duhamel, "Short length FIR filters and their use in fast nonrecursive filtering," *IEEE Trans. Signal Proc.*, vol. 39, pp, 1322–1332, June 1991.

[24] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*. Berlin: Springer, 1981.

[25] M. Vetterli, "Analyse, Synthese et Complexité de Calcul de Bancs de Filtres Numériques," Ph.D. thesis, Ecole Polytechique Federale de Lausanne, 1986.

[26] D. L. Jones and R. G. Baraniuk, "Efficient computation of densely sampled wavelet transforms," in *Advanced Signal-Processing Algorithms, Architectures, and Implementations II*, F. T. Luk (ed.), *Proc. SPIE 1566*, San Diego, CA, July 1991.

[27] M. Unser, "Fast Gabor-like windowed Fourier and continuous wavelet transforms," *IEEE Signal Proc. Letters*, vol. 1, pp.76–79, May 1994.

[28] M. Unser, A. Aldroubi, and S.J. Schiff, "Fast implementation of the continuous wavelet transform with integer scales," *IEEE Trans. Signal Proc.*, vol. 42, pp. 3519–3523, December 1994.