# Fast Algorithms for Discrete and Continuous Wavelet Transforms

Olivier Rioul and Pierre Duhamel

*Abstract*—**Several algorithms are reviewed for computing various types of wavelet transforms: the Mallat algorithm, the "à trous" algorithm and their generalizations by Shensa. The goal is 1) to develop guidelines for implementing discrete and continuous wavelet transforms efficiently, 2) to compare the various algorithms obtained and give an idea of possible gains by providing operation counts. The computational structure of the algorithms rather than the mathematical relationship between transforms and algorithms, is focused upon. Most wavelet transform algorithms compute sampled coefficients of the continuous wavelet transform using the filter bank structure of the discrete wavelet transform. Although this general method is already efficient, it is shown that noticeable computational savings can be obtained by applying known fast convolution techniques (such as the FFT) in a suitable manner. The modified algorithms are termed "fast" because of their ability to reduce the computational complexity per computed coefficient from $L$ to $\log L$ (within a small constant factor) for large filter lengths $L$. For short filters, we obtain smaller gains: "fast running FIR filtering" techniques allow one to achieve typically 30% save in computations. This is still of practical interest when heavy computation of wavelet transforms is required, and the resulting algorithms remain easy to implement.**

*Index Terms*—**Discrete wavelet transform, continuous wavelet transform, octave-band filter banks, computational complexity, fast Fourier transform, fast FIR filtering algorithms.**

## I. Introduction

WAVELET transforms have become well known as useful tools for various signal processing applications. The continuous wavelet transform is best suited to signal analysis [2], [11], [13], [14], [17], [18], [34]. Its semi-discrete version (wavelet series) and its fully discrete one (the discrete wavelet transform) have been used for signal coding applications, including image compression [1], [21] and various tasks in computer vision [19], [20]. Wavelet transforms also find application in many other fields, too numerous to be listed here (see e.g., [34]).

Given a time-varying signal $x(t)$, wavelet transforms consist of computing coefficient that are inner products of the signal and a family of "wavelets." In a continuous wavelet transform, the wavelet corresponding to scale $a$ and time location $b$ is

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

where $\psi(t)$ is the wavelet "prototype," which can be thought of as a band-pass function. The factor $|a|^{-1/2}$ is used to ensure energy preservation [13], [14], [18], [34]. There are various ways of discretizing time-scale parameters $(b, a)$, each one yields a different type of wavelet transform. We adopt the following terminology, which parallels the classical one used for Fourier transforms.

The continuous wavelet transform (CWT) was originally introduced by Goupillaud, Grossmann, and Morlet [13]. Time $t$ and the time-scale parameters vary continuously:

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

(the asterisk stands for complex conjugate).

Wavelet series (WS) coefficients are sampled CWT coefficients. Time remains continuous but time-scale parameters $(b, a)$ are sampled on a so-called "dyadic" grid in the time-scale plane $(b, a)$ [4], [5], [15], [16], [19], [20], [22], [33], [34]. A common definition is

$$C_{j,k} = \mathrm{CWT}\{x(t); a = 2^j, b = k2^j\}, \qquad j, k \in \mathbb{Z}. \tag{3}$$

The wavelets are in this case

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

Wavelet series have been popularized under the form of a signal decomposition onto "orthogonal wavelets" by Meyer, Mallat, Daubechies, and other authors [4], [10], [19], [20], [22], [34]. However, we consider the general (nonorthogonal) case in this paper because nonorthogonal wavelet series are used in practical systems [1]. Computational issues for the orthogonal case are briefly discussed in section II.F.

The discrete wavelet transform (DWT) has been recognized as a natural wavelet transform for discrete-time signals by several authors (see e.g., [10], [26], [32], [33]). Both time and time-scale parameters are discrete. As far as the structure of computations is concerned, the DWT is in fact the same as an octave-band filter bank [10], [19], [26]–[29], [32], [33], depicted in Fig. 1. The filter bank has a regular structure; it is easily implemented by repeated application of identical cells. It is also computationally efficient [25], [29]. Therefore, if the computation of a wavelet transform can be reduced to a DWT, then the resulting implementation is likely to be efficient. Precise definitions and basic properties

Manuscript received February 1, 1991; revised September 20, 1991. This work was presented in part at the IEEE International Conference on Acoustics, Speech, and Signal Processing, Toronto, Canada, May 14–17, 1991.

The authors are with CNET, Centre Paris B, CNET/PAB/RPE/ETP, 38–40, rue du Général Leclerc, 92131 Issy-Les-Moulineaux, France.

IEEE Log Number 9104791.

0018-9448/92$03.00 © 1992 IEEE

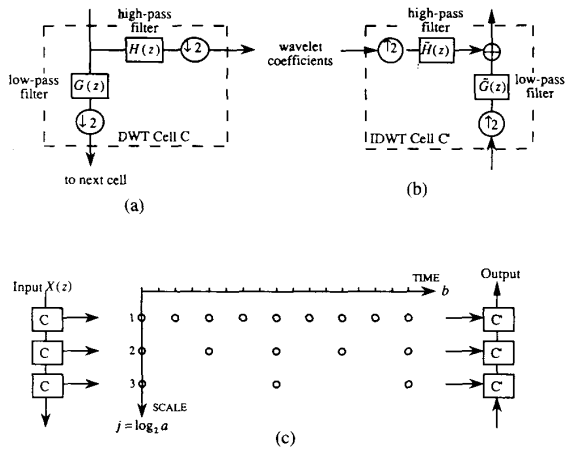


Fig. 1. Basic computational cell of (a) the DWT and (b) the inverse DWT. (c) Overall organization gives wavelet coefficients that correspond to a dyadic grid in the time-scale plane. Signal is reconstructed using the transposed scheme (b).

of various wavelet transforms are reviewed in Sections II-A and II-B.

Several discrete algorithms have been devised for computing wavelet coefficients. The Mallat algorithm [19], [20] and the "à trous" algorithm of Holschneider *et al.* [17] have been known for some time. Shensa was among the first to provide a unified approach [26] for the computation of discrete and continuous wavelet transforms. The material presented next has been derived independently and the overlap with the work of Shensa is pointed out throughout this paper. We also briefly outline other algorithms proposed by the Bertrands and Ovarlez [2] and by Gopinath and Burrus [12].

This paper is divided into two (related) parts. The first (Sections II and III) is devoted to the computation of Wavelet Series coefficients (3). The second (sections IV and V) uses the first to compute CWT coefficients on an arbitrary dense set of time-scale samples, in order to approach a full two-dimensional CWT representation in the time-scale plane $(b, a)$. The operation counts required by the various algorithms obtained are provided in tables.

Section II first reviews some basics of WS coefficients and the DWT. Then, the connection is made: WS coefficients are computed as a DWT applied to a prefiltered version of the signal. This topic is very close to Shensa's [26] but the spirit is slightly different: we focus deliberately on discrete-time implementation issues; the analog signal $x(t)$ is discretized from the start with a more general (hence more flexible) scheme than natural sampling, regardless of any parameter in the wavelet transform algorithm, and we work with discrete-time equivalents to Shensa's conditions [26] under which the DWT is exact for all signals. Finally, denser sampling in scale ($a = a_0^j$, $a_0 < 2$) and the inverse transform case are discussed.

Section III focuses on the computation of the DWT, for use either in the computation of WS coefficients or by itself. It is central in this paper. Known fast convolution techniques, including most recent ones, are applied to the computation of the (already efficient) DWT filter bank structure. This application is not as straightforward as it might seem. One has to take advantage of the special structure of the computations on a DWT. We derive two modified versions of the DWT, one uses the FFT and is efficient for long filters ($L \geq 16$), the other is most efficient for short ones. Reduction of complexity is achieved in any case of interest; complexity can be further reduced in the orthogonal case.

Sections IV and V apply sections II and III, respectively, to the computation of CWT coefficients on an arbitrary regular grid in the time-scale plane $(b, a)$, given by $a = a_0^j$ and $b = k$ (the sampling rate is assumed equal to one). Because the aim is here to approach a nearly continuous two-dimensional CWT representation in the time-scale plane, the resulting class of algorithms will be called "CWT algorithms." We start with a computation of WS coefficients for $a = 2^j$, $b = k2^j$. Then, additional grids are included in the time-scale plane, resulting in a computation of the wavelet coefficients for all integer values of $b$. Finally, additional scales, $a = a_0^j$, $a_0 < 2$, are included to obtain a denser sampling in scale. We use the same method as was derived independently by Shensa [26] that utilizes even and odd sequences at each stage in the algorithm. This can be understood as a modification of the basic "à trous" algorithm, to which fast filtering techniques are applied as in section III.

A distinguishing feature of the topic of wavelets is that many results have been largely spread among researchers before publication, even as unpublished papers. As a result, it is often difficult to give the right credit to the right papers. We shall try to be as clear as possible when presenting results. Unless otherwise stated, the material of this paper is of tutorial nature since there is a large overlap with several other works.

## II. Using the DWT to Compute Wavelet Series Coefficients

### A. Review of the Wavelet Series Transform (WST)

Various data compression schemes, although fully discrete in nature, have been described using the wavelet series formalism. Two-dimensional versions were successfully used for image compression [1], [19]–[21]. Wavelet series are also closely related to octave-band filter banks used in split-band coding via the DWT filter bank structure [27]–[28].

A wavelet series [3]–[5], [15], [16], [19], [20], [22], [26], [34] decomposes a signal $x(t)$ onto a basis of continuous-time wavelets $\tilde{\psi}_{j,k}(t)$ as shown.

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

As in (4), these "synthesis wavelets" usually correspond to discretized parameters $a = 2^j$ ($j$ is called the "octave") and $b = k2^j$. But other choices are sometimes considered [5] (see section II-E). The WS coefficients are defined as

$$C_{j,k} = \int x(t) \psi_{j,k}^*(t)\, dt, \tag{6}$$

where the "analysis wavelets" $\psi_{j,k}(t)$ satisfy (4). Note that the WS scheme is a signal transformation and can be termed the "wavelet series transform (WST)." The direct transform is defined by (6), while the inverse transform, IWST$\{C_{j,k}\}$, is defined by (5).

The analysis and synthesis wavelet prototypes $\psi(t)$ and $\tilde{\psi}(t)$ are equal in the orthogonal case [4], [19], [20], [22]. The more general "biorthogonal" case [3], [32], [33] and "wavelet frames" [5], [16] are not restricted to $\psi(t) = \tilde{\psi}(t)$, however. Whenever the inverse transform is used in the following, we assume that $\psi(t)$ and $\tilde{\psi}(t)$ have been suitably designed so that (5), (6) hold exactly (as in the orthogonal or biorthogonal case), or maybe with sufficient accuracy [5], [16].

This paper is concerned with implementation issues, not with the wavelet design. Therefore, even though design constraints on the shape of wavelets (such as orthogonality) can sometimes be used to reduce the computational load, we do not take advantage of them so as to be as general as possible. However, we shall briefly address the orthogonal case in Section III-F.

### *B. Review of the Discrete Wavelet Transform (DWT)*

The DWT has been used implicitly or explicitly by many authors (see e.g., [1], [3], [4], [10], [12], [19], [20], [22], [32]–[34]). It mainly finds application in image compression [1], [19], [20] (in a two-dimensional form), but is also closely related to octave-band decompositions of filter banks that were used for some time in one-dimensional coding schemes [27], [29]. In section IV, we shall consider a generalization of the DWT which was used by Holschneider *et al.* [17] for analysis of sound signals. We restrict our description in this section to the "standard" DWT whose coefficients are sampled over the dyadic grid $a = 2^j$, $b = k2^j$ in the time-scale plane.

The DWT is in fact very close to wavelet series but in contrast applies to discrete-time signals $x[n]$, $n \in \mathbb{Z}$. It achieves a multiresolution decomposition of $x[n]$ on $J$ octaves labelled by $j = 1, \cdots, J$, given by

$$x[n] = \sum_{j=1}^{+\infty} \sum_{k \in \mathbb{Z}} c_{j,k} \tilde{h}_j[n - 2^j k] + \sum_{k \in \mathbb{Z}} b_{J,k} \tilde{g}_J[n - 2^J k]. \tag{7}$$

This equation is to be compared with (5). The $\tilde{h}_j[n - 2^j k]$ are the *synthesis wavelets*, the discrete equivalents to $2^{-j/2}\tilde{\psi}(2^{-j}(t - 2^j k))$. An additional (low-pass) term is used to ensure perfect reconstruction; the corresponding basis functions $\tilde{g}_J[n - 2^J k]$ are called (synthesis) "scaling sequences" ("scaling functions" can be defined in a similar fashion for wavelet series [1], [3], [4], [12], [19], [20], [22], [26], [33], [34]).

The DWT computes "wavelet coefficients" $c_{j,k}$ for $j = 1, \cdots, J$ and "scaling coefficients" $b_{J,k}$, given by

$$\text{DWT}\{x[n]; 2^j, k2^j\} = c_{j,k} = \sum_n x[n] h_j^*[n - 2^j k] \tag{8a}$$

and

$$b_{J,k} = \sum_n x[n] g_J^*[n - 2^J k], \tag{8b}$$

where the $h_j[n - 2^j k]$'s are the analysis discrete wavelets (compare (8a) with (6)) and the $g_J[n - 2^J k]$ are the analysis scaling sequences. The inverse DWT reconstructs the signal from its coefficients by (7).

The DWT is not fully described yet; wavelets and scaling sequences must be deduced from one octave to the next. Let us restrict ourselves to the analysis part for convenience—the treatment of synthesis "basis functions" is similar. Consider two filter impulse responses $g[n]$ and $h[n]$. (Here, $h$ stands for high-pass—or discrete wavelet, like in [5]—and $g$ stands for low-pass. This results in notations which differ from some previous ones [4], [26], [33].) The wavelets and scaling sequences are obtained iteratively as

$$g_1[n] = g[n], \; h_1[n] = h[n], \tag{9a}$$

$$g_{j+1}[n] = \sum_k g_j[k] g[n - 2k], \tag{9b}$$

$$h_{j+1}[n] = \sum_k h_j[k] g[n - 2k], \tag{9c}$$

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

$$f[n] \to \sum_k f[k] g[n - 2k], \tag{10}$$

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

In fact, it is well known that the structure of computations in a DWT is exactly an octave-band filter bank [10], [19], [26], [27], [29], [32], [33] as depicted in Fig. 1. The DWT corresponds to the analysis filter bank, whereas the IDWT corresponds to the synthesis one. The filters present in the filter bank are precisely $g[n]$, $h[n]$, $\tilde{g}[n]$, 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 (7), (8) hold exactly. That is, the filter bank of Fig. 1 allows perfect reconstruction. For more details on the design the reader is referred to [3], [4], [27], [28], [32], [33].

### *C. The Shensa Algorithm*

Among all types of wavelet transforms, the DWT is the only one that can be computed exactly (except for round-off errors) using a computer, since it deals with discrete-time signals and wavelets. Therefore, we deliberately take a discrete-time approach for the calculation of WS coefficients and avoid taking continuous-time properties into consideration as much as possible when describing the algorithm. We address the following problem: what conditions must the

parameters of the DWT algorithm satisfy so that the DWT computes WS coefficients (on $J$ octaves) exactly for any signal $x(t)$?

The analog input $x(t)$ is discretized from the start, and the whole computation is made in discrete-time. An obvious way to discretize the input is to sample it [17], [26].

$$x[n] = x(t = n),\ n \in \mathbb{Z} \tag{11}$$

(the sampling rate is assumed to be equal to one). This will be called "natural sampling." However, we now use a different discretization scheme, in which the original input $x(t)$ is related to the discrete sequence $x[n]$ (to be input to the algorithm) by a D/A converter as shown.

$$x(t) \approx \hat{x}(t) = \sum_n x[n]\chi(t - n). \tag{12}$$

We have chosen (12) instead of (11) because the resulting scheme is more flexible. Note that (12) includes natural sampling (11) as a special case when $\chi(n) = \delta_{0n}$. For example, if $x(t)$ is band-limited, then $\chi(t)$ is then a $(\sin t)/t$ function. More generally, the choice of $\chi(t)$ and the way $x[n]$ is computed is chosen so that $x(t)$ is well approximated by $\hat{x}(t)$. This requires some knowledge on $x(t)$. For example, the choice $\chi(t) = 1$ for $0 \le t < 1$ and $\chi(t) = 0$, otherwise (zero-order holder) amounts to a piecewise constant approximation to $x(t)$. It is important to note that this discretization is made prior to the algorithm and that the choice of $\chi(t)$ is completely independent of the wavelets or any other parameter in the algorithm.

The algorithm itself, derived independently of Shensa [26], can be described as follows. Given the continuous-time wavelet $\psi(t)$, one first approximates by it $\hat{\psi}(t)$ in such a way that

$$2^{-j/2}\hat{\psi}(2^{-j}t) = \sum_n h_j[n]\phi(t - n), \qquad j = 1, \cdots, J, \tag{13}$$

where $h_j[n]$ are discrete wavelets present in a DWT (see Section II-B), and $\phi(t)$ is some interpolating function. The precise way these $J$ simultaneous approximations can be accomplished will be given in the next section.

The derivation of the algorithm is now straightforward. Substituting (12) and (13) into the equation defining the WS coefficients (6) gives

$$\begin{aligned}
\hat{C}_{j,k} &= \int \hat{x}(t)2^{-j/2}\hat{\psi}^*(2^{-j}t - k)\,dt \\
&= \int \left(\sum_m x[m]\chi(t - m)\right) \\
&\quad \cdot \left(\sum_n h_j[n]\phi(t - 2^j k - n)\right)^* dt \\
&= \sum_n \left(\sum_m x[m] \int \chi(t - m)\phi^*(t - n)\,dt\right) \\
&\quad \cdot h_j^*[n - 2^j k] \\
&= \mathrm{DWT}\{x'[n], 2^j, k2^j\}.
\end{aligned} \tag{14}$$

The last equality comes from (8a). The sequence $x'[n]$ is a prefiltered version of $x[n]$ given by

$$x'[n] = \sum_m x[m]f[n - m],$$

where

$$f[n] = \int \chi(t)\phi^*(t - n)\,dt. \tag{15}$$

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, the input of which is appropriately prefiltered. The accuracy of this algorithm is balanced by the approximations made for the input (12) and for the wavelets (13); the algorithm is exact only once the input and the wavelets have been replaced by their approximations. In the next section, the wavelet approximation is discussed in more detail.

Of course, we could have written

$$x'[n] = \int \hat{x}(t)\phi^*(t - n)\,dt, \tag{16}$$

instead of (15) since $x(t)$ and $x[n]$ are related by (12). But the discrete prefilter $f[n]$ (15) is easy to implement on a computer (its coefficients can be precomputed), whereas (16) is not since it involves analogue filtering. Equation (16) has mathematical significance, however. It hints at the fact that the approximations made in the algorithm consist of replacing the signal and wavelets by their (nonorthogonal) projections onto suitably defined "multiresolution spaces," as defined by Meyer and Mallat [19], [20], [22]. A complete mathematical treatment is out of the scope of this paper. We content ourselves with noting that (16) is just another way of saying that prefiltering (15) must be present.

Note that we have three different types of inputs at work: the original analog signal $x(t)$, its approximation $\hat{x}(t)$ defined by (12), of which the discrete-time equivalent is $x[n]$, and the filtered version $x'[n]$ defined by (15). They involve two successive approximations: the first one is the approximation $x(t) \to \hat{x}(t)$, which is made regardless of the parameters in the algorithm. The second one is the prefiltering (15), which depends on the parameters of the algorithm, and which amounts to a nonorthogonal projection of $\hat{x}(t)$. We shall briefly come back to the concept of nonorthogonal projection in Section II-F.

### *D. A Closer Look at the Wavelet Approximation*

Conditions (13) were introduced to simplify the derivation of the Shensa algorithm. In practice, it is appropriate to replace them by two more tractable conditions equivalent to (13): a first one on $\hat{\psi}(t)$ (in fact, condition (13) written for $j = 1$)

$$1/\sqrt{2}\,\hat{\psi}(t/2) = \sum_n h[n]\phi(t - n) \tag{17}$$

and the other reflecting the parallelism between the way discrete wavelets are defined using "discrete dilation" (10) and the way continuous-time wavelets are defined using the continuous-time dilation $\hat{\psi}(t) \to 2^{-j/2}\hat{\psi}(2^{-j}t)$. The latter is

derived by rewriting (13) at some octave $j$ in two equivalent forms:

$$2^{-(j+1)/2}\hat{\psi}(2^{-(j+1)}t) = \sqrt{2}\sum_n h_j[n]\phi(t/2 - n), = \sum_n \left(\sum_k h_j[k]\, g[n-2k]\right)\phi(t-n). \quad (18)$$

Thus, $\phi(t)$ must satisfy the following "two-scale difference equation"

$$\phi(t) = \sqrt{2}\sum_n g[n]\phi(2t - n). \quad (19)$$

Two-scale difference equations were studied in detail by Daubechies and Lagarias in [6]. They showed that given suitably normalized $g[n]$, there exists at most one integrable solution $\phi(t)$ to (19). But whether the resulting solution is a suitable interpolation function is another matter. For example, to find a necessary and sufficient condition on $g[n]$ such that a solution $\phi(t)$ exists and is $N$-times continuously differentiable is a difficult problem [6]. However, there exists standard choices for $g[n]$ and $\phi(t)$ satisfying (19), such as binomial filters and $B$-spline functions (see below).

The wavelet approximations made in the Shensa algorithm thus, reduce to conditions (17) and (19). These approximations are important because their accuracy determines that of the whole algorithm. There are two steps involved. First, determine a low-pass filter $g[n]$ and an interpolating function $\phi(t)$ satisfying (19). Second, approximate $\psi(t)$ by linear combinations of integer translates of $\phi(t)$ (17). This step determines the high-pass filter $h[n]$. Of course, it is crucial to choose a good interpolating function $\phi(t)$ satisfying (19) so that $\psi(t)$ can be accurately approximated. Note, however, that once $\psi(t)$ is accurately approximated by $\hat{\psi}(t)$ for which (17) and (18) hold, the $J$ approximations at all scales (13) 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*. In the following we briefly mention several "standard" choices for $\phi(t)$.

In a case of a band-limited wavelet $\psi(t)$ (e.g., whose frequency range is restricted to the interval $[-0.5, 0.5]$), a solution to (19) is given by $\phi(t) = \sin(\pi t)/\pi t$ and $g[n] = 1/\sqrt{2}\ \phi(n/2)$. For this choice, the discrete wavelets are precisely the samples of the continuous-time ones

$$h_j[n] = 2^{-j/2}\psi(2^{-j}n)$$

and (14) reduces to a simple discretization of the integral defining the WS coefficients (6). However, this choice is impractical because it involves an ideal low-pass filter $g[n]$, with slow decay as $n \to \infty$.

Another possibility is to choose an orthogonal family of functions $\phi(t-n)(n \in \mathbb{Z})$ satisfying (19), as in the Mallat algorithm [4], [19], [20], [26]. In fact, the computation of the WS coefficients using the DWT reduces exactly to the Mallat algorithm under several conditions: the wavelets and the $\phi(t-n)$'s (called the "scaling functions") are orthonormal, and one has $\chi(t) = \phi(t)$ in (12), which implies that prefiltering (15) is avoided. In the previous context, the latter condition is very unlikely since the A/D characteristic $\chi(t)$ is chosen independently of the parameters in the algorithm. In fact, the Mallat algorithm takes place within a different framework: Given an orthogonal basis of wavelet functions (with corresponding filters $g[n]$ and $h[n]$), and a discrete signal $x[n]$, one constructs an analog signal $x(t)$ satisfying (16) in order for the DWT of $x[n]$ to compute orthogonal WS coefficients exactly. Similarly, the synthesis Mallat algorithm is related to the inverse WST computation derived in Section II-F.

Yet another solution of (19) is the classical "basic spline" interpolating function of some degree $k$, whose Fourier transform is

$$\phi(f) = \left(\frac{\sin \pi f}{\pi f}\right)^{k+1}. \quad (21)$$

Rewriting (19) in the frequency domain and solving for $g[n]$, one finds, within a shift, the binomial filter

$$g[n] = \frac{1}{\sqrt{2}}\binom{k+1}{n}, \qquad n = 0, \cdots, k+1 \quad (22)$$

In this case, (17) reduces to a classical curve fitting problem. This allows a greater flexibility in the wavelet approximation than for the natural sampling case for which one must have [26] the extra condition $g[2n] = 0$ if $n \neq 0$. Therefore, one may obtain a better accuracy in the computation of WS coefficients compared to the popular "à trous" filters [9], [17], [26] for which $\phi(t)$ does not admit a simple closed form expression.

We have left out many details concerning the wavelet approximation. In particular, a precise determination of the error made in the algorithm remains a topic for future investigation. However, we conjecture that wavelet approximation using splines of second or third order suffices for most applications.

### *E. Finer Sampling in Scale*

Shensa's result (14) states that the DWT can be regarded as the basic structure for computing wavelet coefficients. We use this fact throughout this paper to derive various efficient wavelet transform algorithms, concentrating on the computation of the DWT on $J$ octaves. However, a set of points $(a, b)$, denser than the octave-by-octave grid of Fig. 1(c) may be required. It is, therefore, sometimes appropriate to generalize (14) to obtain more samples in the time-scale plane. This is especially useful for signal analysis, where one usually "oversamples" the discretization (3), to obtain "$M$ voices per octave" [5], [18], [34]. That is, $a = 2^j$ is replaced by

$$a = 2^{j+m/M}, \qquad m = 0, \cdots, M-1 \quad (23)$$

where $m$ is called the "voice."

The following simple method [17] allows one to compute WS coefficients on $M$ voices per octave, using only the standard "octave-by-octave" algorithm (14). For

each $m$, replace $\psi(t)$ by the slightly stretched wavelet $2^{-m/2M}\psi(2^{-m/M}t)$ in the expression of $\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k)$. The wavelets basis functions become

$$2^{-(j+m/M)/2}\psi\left(2^{-(j+m/M)}(t - k2^j)\right),$$
$$j, k \in \mathbb{Z},\ m = 0, \cdots, M-1. \quad (24)$$

The grid obtained in the time-scale plane $(b, a)$ is shown in Fig. 2. Now, a computation on $M$ voices per octave is done by applying the octave-by-octave algorithm (14) $M$ times, with $M$ different prototypes

$$2^{-m/2M}\psi(2^{-m/M}t), \qquad m = 0, \cdots, M-1. \quad (25)$$

Of course, the parameters of each octave-by-octave algorithm must be recomputed for each $m$ using the procedure described in Section II-D. Clearly, the whole algorithm requires about $M$ times the computational load of one octave-by-octave algorithm (14).

This method is perhaps not the best one for an "$M$ voices per octave" computation, because it does not take advantage of the fact that the various prototypes (25) 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 the Bertrands and Ovarlez in [2] is based on scale redundancy but is suited for another type of computation (see Section V-G).

### *F. Using the Inverse DWT to Compute the Inverse WST*

We have seen that the WST (6) can be computed using a DWT (8). Similarly, its inverse transform (5) can be computed using an inverse DWT, defined by (7), under a condition similar to (13), but written for *synthesis* wavelets $\tilde{\psi}_{j,k}(t)$:

$$2^{-j/2}\tilde{\psi}(2^{-j}t) = \sum_n \tilde{h}_j[n]\tilde{\phi}(t - n), \qquad j = 1, \cdots, J. \quad (26)$$

Of course, this condition is, in practice, replaced by more tractable conditions as explained in Section II-D. Substituting (26) for $2^{-j/2}\psi(2^{-j}t)$ in the formula defining the inverse WST (5) results in

$$\text{IWST}\{C_{j,k}\} = \sum_n y[n]\tilde{\phi}(t - n), \quad (27)$$

where the $C_{j,k}$ are the WS coefficients (6) and $y[n]$ is defined by

$$y[n] = \text{IDWT}\{C_{j,k}\}. \quad (28)$$

Thus, the IDWT, followed by a D/A converter with characteristic $\tilde{\phi}(t)$, computes the IWST exactly. The accuracy of the algorithm again depends on that of the signal and wavelet approximation. The resulting analysis/synthesis WST scheme is depicted in Fig. 3. First, the analog signal $x(t)$ is discretized according to (12). The discrete-time signal $x[n]$ is then prefiltered (15) and fed into the DWT algorithm.

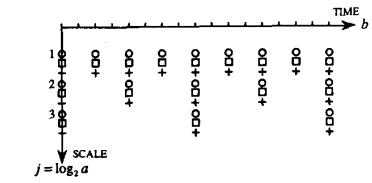


Fig. 2. Sampling of the time-scale plane corresponding to 3 voices per octave in a WST. The imbrication of the computation is shown using points labelled by circles, squares, and crosses, which can be computed separately using octave-by-octave DWT algorithms.

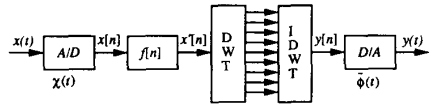


Fig. 3. Full analysis/synthesis WST scheme. Exact reconstruction holds under certain conditions on $x(t)$ (see text).

During synthesis, the signal is reconstructed using an IDWT, followed by the interpolation (or D/A conversion) (27).

Note that in this WST/IWST Shensa algorithm, the analysis and synthesis discrete wavelets do not necessarily form a perfect reconstruction filter bank pair. However, we now restrict 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 $y[n] = x'[n]$. It can be shown that we are in fact in the "biorthogonal" case [3], [32], [33], and that one has

$$\int \phi(t - n)\tilde{\phi}^*(t - m)\, dt = \delta_{m,n}. \quad (29)$$

Since $y[n] = x'[n]$, we also have

$$\text{IWST}\{\text{WST}\{x(t)\}\} = \sum_n \left(\int \hat{x}(u)\phi^*(u - m)\, du\right)\tilde{\phi}(t - n). \quad (30)$$

The right-hand side of (30) is easily recognized to be a projection of $\hat{x}(t)$ onto the subspace $V$ spanned by linear combinations of the $\tilde{\phi}(t - n)$: if $\hat{x}(t)$ belongs to $V$, i.e., if $\hat{x}(u) = \sum_n c_k\tilde{\phi}(t - k)$, then using (29), equation (30) simplifies to $\hat{x}(t)$. Therefore, unless $\hat{x}(t)$ belongs to the appropriate subspace $V$, it cannot be recovered at the output of a WST/IWST Shensa algorithm. Only its projected approximation onto $V$ is recovered.

We, therefore, have two types of loss of information at the reconstruction of a WST/IWST: One is, of course, due to the approximation $x(t) \to \hat{x}(t)$, which may not be invertible. The other is due to the projection of $\hat{x}(t)$ onto the subspace $V$. This can be seen as the fact that the discrete equivalent to $\hat{x}(t)$, $x[n]$, may not be recovered from its filtered version $x'[n]$ that is obtained at the output of the IDWT. The condition that $\hat{x}(t)$ belongs to $V$ clearly depends on the parameters of the algorithm. Therefore, the output error in the algorithm is produced by the discretization scheme, and the way the continuous-time signal and wavelets are recovered from their discrete-time counterparts.

## III. 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.

### A. Preliminaries

In this section, we specify the framework in which the algorithms will be derived. We also briefly motivate the need for further reduction of complexity in a DWT.

We assume real data and filters (of finite length), but the results extend easily to the complex-valued case. It can be shown that the FFT-based algorithms described next require about twice as many multiplications in the complex case than in the real case, a property shared by FFT algorithms [8], [24]. However, a straightforward filter bank implementation of the DWT (Fig. 1), or the "short-length" algorithms described in Section III-E 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].

In our derivations, we do not take advantage of possible constraints (such as orthogonality), even though these can be used to further reduce the complexity. The resulting algorithms therefore apply in general. (See Section III-F for a brief discussion of the orthogonal case.)

The derivation of fast algorithms 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 (see Section II-B). 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).

Due to the lack of space, we shall not derive algorithms explicitly for the inverse DWT. However, a IDWT algorithm is easily deduced 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 IDWT. It can be shown that this implies that the DWT and IDWT require *exactly the same number of operations* (multiplications and additions) per point.

The filters involved in the computation of the DWT (cf. Fig. 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 used in [9], [17] uses a short low-pass filter $g[n]$ and a long high-pass filter $h[n]$, we restrict in this section to the case of equal filter lengths for simplicity. If lengths differ, one can pad the filter coefficients with zeros. Section III-G discusses the case when filters are of very different lengths.

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 Saramäki in the context of octave-band filter banks [25]. What makes the DWT "fast" is the decomposition of the computation into elementary cells and the subsampling operations (called decimations), which occur at each stage. More precisely, the operations required by one elementary cell at the $j$th octave (Fig. 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

$$L \text{ mults/point/cell and } L-1 \text{ adds/point/cell.} \quad (31)$$

Since the cell at the $j$th ocatve has input subsampled by $2^{j-1}$, the total complexity required by a filter bank implementation of the DWT on $J$ octaves is $(1 + 1/2 + 1/4 + \cdots + 1/2^{J-1}) = 2(1 - 2^{-J})$ times the complexity (31). That is

$$2L(1-2^{-J}) \text{ mults/point and}$$

$$2(L-1)(1-2^{-J}) \text{ adds/point.} \quad (32)$$

The DWT is therefore roughly equivalent, in terms of complexity, to one filter of length $2L$. A remarkable fact is that the complexity remains bounded as the number of octaves, $J$, increases [25].

We remark, in passing, that a naive computation of the DWT would implement (8) exactly as written, with precomputed discrete wavelets $h_j[n]$. This does not take advantage of the dilation property of wavelets (9), and therefore is not effective. Since the length of $h_j[n]$ is $(L-1)(2^j-1)+1$, one would have, at the $j$th 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

$$J(L-1)+1 \text{ mults/point and } J(L-1) \text{ adds/point.} \quad (33)$$

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."

The aim of the following sections is to further reduce the computational load of the DWT. We briefly motivate this with a brief analogy to fast filtering. FFT's 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 $\log L$, hence the term "fast." For short filters, however, the FFT is no longer efficient and other fast filtering techniques are used [23], [24], [31]; 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 FFT's, we shall be able to reduce the complexity of the DWT from $2L$ to $4 \log L$, when the filter length $L$ is large. However, DWT's have been mostly used with short filters so far (although nothing ensures that this will last forever). For them, (using different techniques) we shall obtain smaller gains, typically 30% save in computations, which can still be desirable. The algorithms derived in the following are therefore called "fast DWT algorithms," even though it can be argued that this can be confused with the already fast straightforward implementation of the DWT.

### B. Reorganization of the Computations

From the operation counts above (32), it is clear that if all of the 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. To further reduce the computational load of the DWT, it, therefore, suffices to apply fast convolution techniques to only one elementary cell. We propose two classes of fast algorithms: one based on the FFT [8], [24] and the other on short-length FIR filtering algorithms [23], [31].

The basic DWT elementary cell, depicted in Fig. 1(a), contains two filters. But these do not appear alone, since 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 bringing together the computation of several successive outputs [24]. Since the filter outputs are decimated in Fig. 1(a), it is necessary to reorganize the computations in such a way that "true" filters appear. To do this, we apply a classical reorganization of filter banks building blocks [29], [32] based on "biphase decomposition," which consists of separating into even and odd sequences. At this point, it is convenient to use the $z$-transform notation. The biphase decomposition expresses the $z$-transform of the input sequence $x[n]$,

$$X(z) = \sum_n x[n] z^{-n} \tag{34}$$

in the form

$$X(z) = X_0(z^2) + z^{-1} X_1(z^2)$$

where

$$X_0(z) = \sum_n x[2n] z^{-n} \text{ and } X_1(z) = \sum_n x[2n+1] z^{-n}. \tag{35}$$

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. Since we have

$$\begin{aligned} G(z)X(z) &= \left(G_0(z^2) + z^{-1}G_1(z^2)\right)\left(X_0(z^2) + z^{-1}X_1(z^2)\right) \\ &= G_0(z^2)X_0(z^2) + z^{-2}G_1(z^2)X_1(z^2) + \text{odd terms}, \end{aligned} \tag{36}$$

picking out the even part of $G(z)X(z)$ results in

$$Y(z) = G_0(z)X_0(z) + z^{-1}G_1(z)X_1(z). \tag{37}$$

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. 4 (the corresponding IDWT cell is simply obtained by flow graph transposition). Compare with Fig. 1(a): there are now four "true" filters of length $L/2$, whose impulse responses are the decimated initial filters $G(z)$ and $H(z)$.

### C. An FFT-Based DWT Algorithm

This method consists of computing the four $L/2$-tap filters of Fig. 4 using the FFT. More precisely, we use the "split-radix" FFT algorithm [8] which, among all practical FFT algorithms, has the best known complexity for lengths

$$N = 2^{\boldsymbol{n}} \tag{38}$$

($\boldsymbol{n} = \log_2 N$ is typed boldface to avoid confusion with the samples index $n$). For real data, the split-radix FFT (or inverse FFT) requires exactly

$$\begin{aligned} &2^{\boldsymbol{n}-1}(\boldsymbol{n} - 3) + 2 \text{ (real) mults} \\ &2^{\boldsymbol{n}-1}(3\boldsymbol{n} - 5) + 4 \text{ (real) adds}. \end{aligned} \tag{39}$$

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 FFT's 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. It is well known, however, that this does not give a true filter convolution, but a *cyclic* convolution [24]. There-

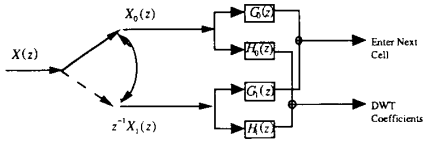


Fig. 4. Rearrangement of the DWT cell of Fig. 1(a) that avoids subsampling and, hence, allows the application of fast filtering techniques.

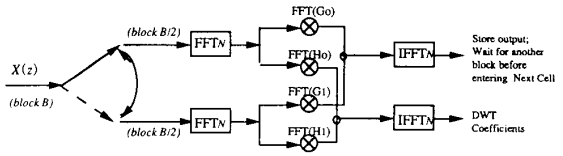


Fig. 5. FFT-based implementation of the DWT cell of Fig. 4. Overlap-add (or overlap-save) procedure is not explicitly shown.

fore some time processing must be done in order to avoid wrap-around effects. There are two well-known methods for this, called the overlap-add and overlap-save methods [24]. 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 $B/2$, wrap-around effects are avoided if the FFT-length $N$ satisfies the inequality

$$N \geq L/2 + B/2 - 1. \tag{40}$$

The block length is therefore determined by

$$B = 2N - (L - 2). \tag{41}$$

The DWT algorithm is thus modified as follows. As before, each elementary cell has the same structure, pictured in Fig. 5. 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 IFFT's are applied. Assuming that a complex multiplication is done with three real multiplications and three real additions [24], this gives

$$2\,\mathrm{FFT}_N + 4.3.N/2 \text{ mults} + 4.3.N/2 \text{ adds} + 2N/2 \text{ adds} + 2\,\mathrm{IFFT}_N$$

per cell, for $B$ inputs. That is

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

$$\frac{(3n-1)2^{n+1} + 16}{2^{n+1} - (L-2)} \text{ adds/point/cell}. \tag{42}$$

Note that $L$ has typically the same order of magnitude as $N$, hence, the number of operations in (49) is roughly proportional to $n \approx \log L$. More precisely, for a given length $L$ there is an optimal value of $B = 2N - (L-2)$, i.e., an optimal value of $N = 2^n$ that minimizes (42). Table I and II show the resulting minimized complexities for different lengths $L$ in comparison with the direct method (31) (straightforward filter bank implementation). The comparison is evidently in favor of the FFT version of the DWT algorithm for medium to large filter lengths ($L \geq 16$).

A more precise comparison with (31) can be done for large filter lengths by minimizing the criterion (mults + adds) of (42)

$$C(N) = \frac{(4\log_2 N - 1)N + 12}{N - (L/2 - 1)}, \tag{43}$$

with respect to $N$. The minimal value of $C(N)$ attained when $N = N^*$ is such that the first derivative of $C(N)$ vanishes. One has

$$C(N^*) = \min_N C(N) = 4\log_2 N^* + (4/\ln 2 - 1) \tag{44}$$

where $N^*$ satisfies the relation $N^* = (L/2 - 1)(\ln N^* + 1 - \ln 2/4) + 3\ln 2$. For large filter lengths $L$ this gives $\ln N^* = \ln L + O(\ln\ln L)$, hence,

$$\min_N C(N) = 4\log_2 L + O(\log\log L). \tag{45}$$

This is to be compared with (31), for which the value of the criterion (mults + adds) equals $2L - 1$. The FFT-based DWT algorithm significantly improves the direct method for large lengths $L$. The gain is about $L/(2\log_2 L)$. However, as seen in Table I, the FFT implementation of the DWT is not effective for short filters.

There is a subtlety to keep in mind when wrap-around effects at the cell output are eliminated in the time-domain. One could immediately take the output blocks (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, as shown in Section III-A, $2(1 - 2^J)$ times the complexity of one cell.

### *D. A Generalization: The Vetterli Algorithm*

The FFT-based DWT algorithm described above can be improved by gathering $J_0$ consecutive stages, using a method due to Vetterli (originally in the filter bank context [30], and then applied to the computation of the DWT [32]). The idea is to avoid subsequent IFFT's and FFT's by performing the subsampling operation in the frequency domain. This is done

TABLE I
FFT-BASED DWT ALGORITHMS: ARITHMETIC COMPLEXITY PER POINT AND PER OCTAVE*

| Filter Length $L$ | Straightforward Filter Bank (Section III-A) | FFT-Based Algorithm (Section III-B) | Vetterli's Alg. (2 octaves merged) (Section III-C) | Vetterli's Alg. (3 octaves merged) (Section III-C) | Vetterli's Alg. (4 octaves merged) (Section III-C) |
|---|---|---|---|---|---|
| 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 + 15.53 (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 corresponding initial FFT length. Complexities should be multiplied by $2(1 - 2^{-J})$ for a computation of the DWT on $J$ octaves.

TABLE II
ARITHMETIC COMPLEXITY PER POINT AND PER CELL* FOR VARIOUS DWT ALGORITHMS

| Filter Length $L$ | Straightforward Filter Bank (Section III-A) | FFT-Based Algorithm (Section III-B) | Short-Length Algorithm (Section III-C) |
|---|---|---|---|
| 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 (see text). Complexities should be multiplied by $2(1 - 2^{-J})$ for a computation of the DWT on $J$ octaves.

by inverting the last stage of a decimation-in-time radix-2 FFT algorithm [30], [32]. 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 have two major limitations. First, the structure of computations is less regular than for the simple FFT algorithm of the preceding section because FFT's have different lengths. Second, the relative efficiency of an FFT scheme per computed point decreases at each stage. The difficulties brought by this method are easily understood even by evaluating its arithmetic complexity. One finds, assuming the DWT is computed on a multiple of $J_0$ octaves, an average of

$$\frac{2^{J_0-1}}{2^{J_0}-1}\,\frac{2^n(2n+5-10.2^{-J_0})+2(J_0+3)}{2^{n+1}-(2^{J_0}-1)(L-2)}\cdot \text{mults/point/cell}$$

$$\frac{2^{J_0-1}}{2^{J_0}-1}\,\frac{2^n(6n+5-14.2^{-J_0})+4(J_0+3)}{2^{n+1}-(2^{J_0}-1)(L-2)}\cdot \text{adds/point/cell} \quad (46)$$

per elementary cell (this complexity was calculated such that the total complexity of the DWT algorithm is exactly $2(1 - 2^{-J})$ times the average complexity per cell (46), so as to permit a precise comparison with (42)). Note that (46) reduces to (42) for $J_0 = 1$.

Table I 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 ($J_0 = 1$), but only for long filters ($L \geq 32$) and small $J_0$. Efficiency is lost in any case when $J_0$ is greater than 3.

### *E. 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. In this section, we apply short-length "fast running FIR" algorithms [23], [31] to the computation of the DWT. The class of "fast running FIR algorithms" is interesting because the multiply/accumulate structure of computations is partially retained. These algorithms are in fact very easily implemented [23], [31].

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 give the exact decimated filter outputs.

Fig. 6 provides an example for $R = 2$, $A_i = 2$, $M = 3$, and $A_0 = 2$. We have also applied other algorithms derived in [23], corresponding to the values $R = 3$, $A_i = 4$, $M = 6$, $A_o = 6$, and $R = 5$, $A_i = 14$, $M = 12$, $A_o = 26$.

This computation can be repeated: the subfilters of length $L/R$ are still amenable to further decomposition. For example, 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 "$3 \times 5$ algorithm," which first applies the procedure with $R = 3$, then again decompose the subfilters (of length 5) using the procedure associated with $R = 5$. Alternatively, a "$5 \times 3$ algorithm" can be used. Each of these algorithms yield different complexities, which are discussed in detail in [23]. We restrict ourselves here to at most two nested applications of fast running FIR algorithms (as in the previous example), so that the resulting computation remains simple, even though this is at the cost of a slight loss of efficiency.

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. 4). Here, since two pairs of filters share the same input, all pre-additions ($A_i$) can be combined together on a single input.

Table II 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 (another choice would have been to minimize the number of multiplication-accumulations). Table II shows that short-length DWT algorithms are more efficient than the FFT-based DWT algorithms for lengths up to $L = 18$.

Since, in practice, DWT's are generally computed using short filters [1], [19], [20], the short-length algorithms probably give the best practical alternative. Compared to the straightforward filter bank implementation, they do provide noticeable savings: 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. Such a gain is interesting when heavy DWT computation is required.

### F. The Orthogonal Case

In our derivations, we did not take advantage of orthogonality constraints [4], [10], [19], [20], [22], [34] 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

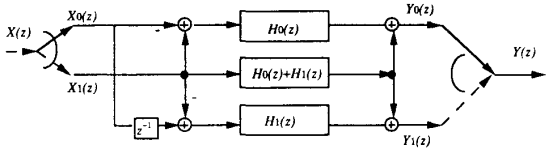


Fig. 6. Simple example of fast running FIR filtering algorithm with decimation ratio $R = 2$ [23]. Subscripts 0 and 1 indicate biphase decomposition.

of the DWT filter bank cell of Fig. 1(a), Vaidyanathan and Hoang have shown in [28], [29] that the complexity can be reduced by a factor of 50% in the orthogonal case.

Preliminary work on this subject using the relation

$$h[n] = (-1)^n g[L - 1 - n], \tag{47}$$

which holds for orthogonal wavelets (see e.g., [4], [19]), shows that 25% reduction of computational complexity is attainable (with techniques similar to the ones previously stated), while preserving the classical FIR filtering structure. This indicates that this 25% reduction will be preserved in accelerated algorithms based on the above techniques. Whether or not 50% reduction can be attained while preserving the inner products (unlike the lattice structure implementation) is an open problem. A complete treatment remains a topic for future investigation. In any case, Tables I and II do not provide a fair and detailed comparison between various algorithms in the orthogonal case.

### G. Unequal Filter Lengths

In the previous derivations, we have restricted ourselves to filters of equal lengths for simplicity. However, as pointed out to us by one of the reviewers, 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 [9], [17], 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 (31) then becomes $(L_g + L_h)/2$ mults and $(L_g + L_h - 1)/2$ adds. (Padding the short filter coefficients with zeros as previously suggested would require $L_h$ mults and $L_h$ adds instead.)

However, the equal length assumption is still valid for the derivation of fast (FFT-based) DWT algorithms for two reasons.

1) If one of the lengths is large ($L_h \gg 16$), then clearly the most suitable algorithm for improving efficiency compared to a straightforward filter bank implementation is an FFT-based DWT algorithm. One can use the one described in Section III-C, although the fact that one filter is of small length is then not taken advantage of. Alternatively, this algorithm can be modified to compute the short filter directly as inner products and use the FFT only for the long one. But then, as is clear from Fig. 5, four FFT's are still required for the inputs

and outputs just as in the initial FFT-based algorithm, and saving is only obtained for the frequency domain multiplications (2 instead of 4).

For a wavelet of length $L_h = 64$ and interpolation filters of length $L_g$ of 3, 7, and 11, the modified "unequal lengths" FFT-based algorithm give gains over a standard DWT of 49.9%, 47.1% and 44.5%, respectively. However, for the same lengths the initial FFT-based algorithm of Section III-C give respective gains over a standard DWT of 46.9%, 49.9% and 52.6%. More generally, we found that the modified FFT-based algorithm is more efficient than the initial FFT-based one only for very short interpolation filters ($L_g \leq 4$), and in any case the complexities of both methods are about the same.

Therefore, for very different lengths $L_g \ll L_h$, some efficiency of FFT-based algorithms is lost compared to a direct implementation based on the pessimistic assumption $L_g = L_h$ (because the direct implementation is then significantly more efficient), but the FFT-based algorithm of Section III-C, which is based on the very same pessimistic assumption, cannot be greatly improved. Note that in such a case it is still more interesting to use an FFT-based algorithm that yields a substantial gain over a standard, straightforward filter bank implementation of the DWT.

2) One can, and indeed should, reduce the difference between $L_g$ and $L_h$ as much as possible in the wavelet approximation procedure, since the complexity can then be significantly reduced using one of the accelerated algorithms described earlier. For the Morlet example, it is possible to use higher order splines as suggested in Section II-D to obtain discrete filters of moderate lengths $L_g$ and $L_h$ with a comparable approximation error, thereby resulting in a much more efficient "equal length" implementation, that is, one of the accelerated algorithms given above, as applicable.

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. Obviously we cannot describe all specific cases in detail (this would require much more space). Nevertheless, the linear phase case can be treated just as easily as the previous general case, and one still obtains reduction of complexity in any case of interest using appropriate methods.

## IV. Continuous Wavelet Transform Algorithms

### A. CWT Coefficients Sampled on Arbitrary Grids and the "à Trous" Algorithm

In this section, we review how the Shensa algorithm [26] reviewed in Section II-C, is applied to the computation of CWT coefficients sampled on arbitrary grids in the time-scale plane. As in the WS case, the DWT can be used as an intermediate step. The time-scale parameters are discretized as shown:

$$\begin{aligned} a &= a_0^j \\ b &= k, \end{aligned} \tag{48}$$

where $1 < a_0 \leq 2$. Notice that $a$ is restricted to positives 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 (48) is the possibility to approximate a nearly continuous CWT representation in the time-scale plane for analysis purposes. For this reason, the algorithms described in the following will be called "CWT algorithms."

Let us first restrict to an octave-by-octave computation, i.e., $a = 2^j$. In the next section, we shall consider the computation if the CWT on a fine grid in the time-scale plane. We remark that the computation of the WS coefficients treated in Sections II and III is nothing but part of the computation required here: One has

$$C_{j,k} = \mathrm{CWT}\left\{x(t); 2^j, k2^j\right\}. \tag{49}$$

Now, the Shensa algorithm for the WS coefficients (cf., Section II-C) can be readily extended to the required computation of CWT $\{x(t); 2^j, k\}$ [26]. We have a result similar to (14), namely,

$$\mathrm{CWT}\left\{x(t); 2^j, k\right\} = \mathrm{DWT}\left\{x'[n]; 2^j, k\right\}, \tag{50}$$

where $x'[n]$ is a prefiltered discrete input defined by (15). The difference with (14) is, of course, that the DWT is computed for all integer values of $b$. In contrast, in the standard description of the DWT (Section II-B), the wavelet coefficients are only output every $2^j$ samples ($b = k2^j$). Equation (50) 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 [12] and subsequently discussed in detail by Shensa in [26]: The resulting CWT algorithm was recognized to be identical with the "à trous" algorithm of Holschneider *et al.* [9], [17].

In fact, the framework of Section II-C is more general than that found in [17], in which a sampled version of the integral in (6) is the starting point. This corresponds, in our case, to the extra condition that perfect sampling is used for both the input (11) and the wavelets (20). This implies that the low-pass filter $g[n]$ (cf. Section II-D) in [17] should fulfill the condition

$$g[2n] = \delta_{n,0}, \tag{51}$$

which comes from $g[n] = (1/\sqrt{2})\phi(n/2)$ in (19). This forbids the "binomial" choice (22) for spline orders greater than one. In [17], Holschneider *et al.* use instead a second order Lagrangian filter for which $\phi(t)$ cannot be written in closed form (such $\phi(t)$'s were studied in detail by Deslauriers and Dubuc [7]). Therefore, the curve fitting problem (13) is more tedious to solve in the "perfect sampling" case. This is a possible disadvantage considering that the accuracy of the algorithm is governed by the solution to (13). Unlike [17], [26], the more flexible framework of equations (12), (13) has allowed us to use a classical curve fitting problem using splines in the wavelet approximation (see Section II-D).

### B. Finer Sampling in Scale

In signal analysis, an octave-by-octave computation of the CWT (i.e., with $a = 2^j$) is generally not enough. It is desirable to obtain more wavelet coefficients, with finer sampling in the scale parameter $a$, namely, $a = 2^{j/M}$, where $M$ is the number of "voices per octave" [5], [18], [34].

To do that we apply the same trick [17] as in section II-E. An "$M$ voices per octave" CWT computation results from $M$ successive applications of the octave-by-octave algorithm, each one corresponding to a different basic wavelet prototype:

$$2^{-m/M}\psi(2^{-m/M}t),\ m = 0, \cdots, M-1. \qquad (52)$$

The approximation (17) must be satisfied for each of these slightly stretched wavelet prototypes (52), and the whole algorithm requires about $M$ times the computational load of an octave-by-octave algorithm.

### C. Computation of the Inverse Continuous Wavelet Transform

It is well known that the CWT brings a lot of redundancy into the representation of the signal (a one-dimensional signal is mapped to a two-dimensional plane). As a result, there are several possibilities to reconstruct the signal $x(t)$ from its CWT coefficients (2).

One can use the classical (but computationally expensive) inversion formula [13], [14]

$$x(t) = c\iint \mathrm{CWT}\{x(t); a, b\}\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)\frac{da\,db}{a^2}, \qquad (53)$$

where $c$ is a constant depending only on $\psi(t)$.

More efficient is to use an inverse WST on the coefficients $C_{j,k} = \mathrm{CWT}\{x(t); 2^j, k2^j\}$, when $\psi(t)$ is carefully chosen. Reference [5] contains a detail treatment of questions of accuracy if perfect reconstruction is not ensured by the choice of the wavelet $\psi(t)$. The algorithms presented in Section III apply to this method. We have mentioned in Section III-A that a DWT and an inverse DWT algorithm have identical complexities if the latter is obtained by flow-graph transposing the former. Thus, the complexity is easy to estimate in this case.

Still another way is to use Morlet's formula [13], [14], [17]

$$x(b) = c'\int \mathrm{CWT}\{x(t); a, b\}\frac{da}{a^{3/2}}, \qquad (54)$$

which requires a single itegration. This computation is performed from available CWT coefficients, which should be known for a sufficient number values of $a$ in order to give an accurate reconstruction. The issue of convergence remains open.

## V. Efficient Implementations of the Continuous Wavelet Transform

In the following, we derive efficient implementations of the CWT, or, more properly, of CWT coefficients sampled on regular grids in the time-scale plane $(b, a)$, restricting to an octave-by-octave computation, $a = 2^j$, $b = k$ (see Section IV-B for a generalization to $a = a_0^j$). We assume that the discrete input has been prefiltered in a suitable manner as explained in Sections II-C and IV-A, and concentrate on the computation of $\mathrm{DWT}\{x[n]; 2^j, k\}$.

### A. Reorganization of the "à Trous" Computational Structure

There are several ways of deriving a filter bank implementation of the CWT. In [17], Holschneider *et al.* proposed an "à trous" structure pictured in Fig. 7(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 $j$th octave.) From (50) it is also possible to derive a CWT algorithm by combining several DWT algorithms with $a = 2^j$, $b = k2^j + k_0$, $k_0 = 0, \cdots, 2^j - 1$ in a suitable manner. In this case, the algorithms of Section III could be readily used, although we found that the resulting CWT algorithms would not be the most efficient ones. All of these variations (including the one used next) require exactly the same complexity when filters are implemented directly as inner products (and multiplication by zero is avoided for the "à trous" structure).

However, we now use another variation of filter bank implementation of the CWT, which was also derived independently by Shensa in [26], because it is more suited to further reduction of complexity using fast filtering techniques. Consider the filter bank structure of Fig. 7(c), where the elementary cell is depicted in Fig. 7(b). This filter bank structure is easily deduced from the one of Fig. 7(a) [26].

Since the accelerated algorithms of Section III apply to the computation of the DWT filter bank structure of Fig. 1, it is important to relate it to the new filter bank structure of Fig. 7(c). Consider, more specifically, the computation performed at the first octave ($j = 1$) of Fig. 7 and compare it to Fig. 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 in Fig. 7(a).

Also, in Fig. 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 to obtain the coefficients with *odd* values of $b$ is nothing but the discarded subsampled sequence; it is recovered in Fig. 7(a).

At the next octave $j = 2$, both inputs are processed separately using identical cells. One provides the same points as in the WST computation (round dots in Fig. 7(c)), while the other allows to start a new computation of the same type, shifted in time, and beginning at the next scale (squared dots in Fig. 7(c)). The whole process is iterated as shown in Fig. 7(c).

In the overall organization, the outputs of both filters have to be computed, those of $G(z)$ are used to build two interleaved sequences, while those of $H(z)$ are simply the wavelet

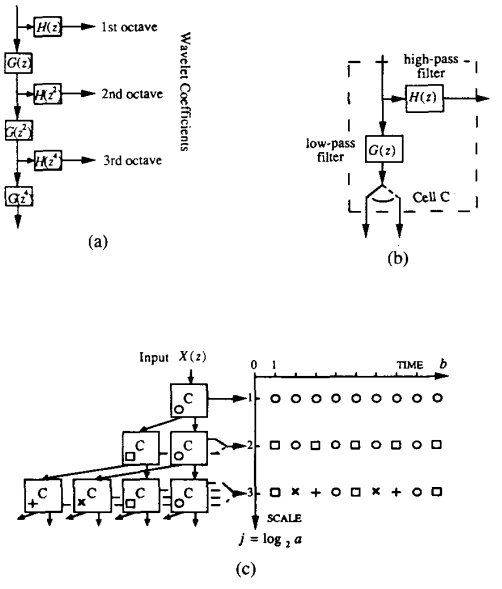


Fig. 7. "À trous" structure as derived by Holschneider *et al.* (b) Basic computational cell used for computing CWT coefficients octave and octave. (c) Connection of the cells used in this paper and corresponding location of the wavelet coefficients in the time-scale plane.

coefficients. Note that the basic computational cells of the fast DWT algorithms were specifically designed in Section III for decimated outputs. Their structure is therefore not adapted to the new situation, and the operation counts have to be reworked. This is done next.

### *B. Straightforward Filter Bank Implementation of the CWT*

Consider the filter bank implementation of Fig. 7(c), and assume, as in Section III, that both filters $g[n]$ and $h[n]$ are FIR filters and have same length $L$. (Section V-F discusses the case of unequal filter lengths). When the filters are directly implemented as inner products, the octave-by-octave CWT algorithm requires

$$2L \text{ mults/input point/cell}$$

$$2(L-1) \text{ adds/input point/cell}. \tag{55}$$

Note that there are $2^{j-1}$ elementary cells at the $j$th octave in Fig. 7. These cells are identical but "work" at a different rate: a cell at the $j$th 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 $\sum_{j=1}^{J} 2^{j-1}/2^{j-1} = J$ times the complexity of one cell. Thus, the complexity of any filter bank implementation of a CWT grows linearly with the number of octaves.

In case of a direct implementation (55), the total complexity required by a CWT on $J$ octaves is simply

$$2LJ \text{ mults/input point}$$

$$2(L-1)J \text{ adds/input point}. \tag{56}$$

As mentioned in [17], this is a significant improvement compared to the "naive method" that would consist in directly implementing the CWT and would not take advantage of the fact that wavelets are easily related by dilation (this would reqquire 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 the techniques described previously in section III for WS coefficients. Furthermore, in the CWT case filter lengths are comparatively twice as long as in the WST case. This will increase the efficiency of the accelerated algorithms described below.

### *C. An FFT-Based CWT Algorithm*

As in Section III.C, the FFT can be used to accelerate the two filter computations in the elementary cell of the filter bank of Fig. 7(b), (c). In the CWT case, wrap-around effects are avoided if the FFT-length $N$ is such that

$$N \geq L + B - 1 \tag{57}$$

where $B$ is the input block length, and where $L$ is the filter length. The block length is, therefore, chosen as $B = N - (L-1)$. Each elementary cell is computed by first performing an FFT of length $N = 2^n$ on the input, then performing two frequency-domain convolutions by multiplying (Hermitian symmetric) length-$N$ FFT's of $g[n]$ and $h[n]$, and finally applying two inverse FFT's on the results. This requires $2\text{FFT}_N + 2N/2$ complex mults + $\text{IFFT}_N$ per cell (for $B$ input points). Under the same conditions as in Section III-C, we obtain

$$\frac{3.2^{n-1}(n-1)+6}{2^n - L + 1} \text{ mults/input point/cell}$$

$$\frac{9.2^{n-1}(n-1)+12}{2^n - L + 1} \text{ adds/input point/cell} \tag{58}$$

for each elementary cell. Once a cell is computed, wrap-around effects are cancelled in the time domain and one waits for one block before entering the next stage, so that each cell has the same input block length $B$ and the same FFT length $N$. Table III lists the obtained complexities (58), minimized for an optimal value of $N$, for different values of the filter length $L$. Since all cells are computed with FFT's of the same length $N$, once $N$ is optimal for one cell, it is optimal for the whole algorithm.

Table III shows that compared to the direct implementation (55), the FFT computation is more efficient than the direct method for $L \geq 9$, in terms of total number of operations (multiplications + additions). By deriving this criterion with respect to $N$, one finds that the optimal FFT length satisfies the relation $N = (0.69n + 0.31)(L-1)$. For large $L$, the

TABLE III
ARITHMETIC COMPLEXITY PER COMPUTED POINT* FOR VARIOUS CWT ALGORITHMS

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

* Each entry gives the number of operations per computed wavelet 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 (see text).

corresponding minimized total number of operations per computed point is

$$6\log_2 L + O(\log\log L).$$

This is a significant improvement compared to (55), for which the total number of operations per input point is about $4L$. The gain of the FFT computation is therefore asymptotically $2L/3\log_2 L$ for large filter lengths $L$. This gain is larger than in the WST case (Section III-C).

### D. A Generalization Using the Vetterli Algorithm

The method shortly presented in Section III-D, that Vetterli [30], [32] derived in the DWT case, can also be used for the octave-by-octave CWT computation. The discussion of Section III-D could have been made here as well.

We provide an example, when two octaves are gathered together. Three elementary cells of Fig. 7(c) are merged into one 1-input, 7-output cell that covers two octaves. Here the FFT length $N = 2^n$ must be greater than or equal to $B + 3(L-1)$ to avoid wrap-around effects (compare with (57)). This results in an average of

$$\frac{2^{n-1}(2n-1)+6}{2^n - 3L + 3} \text{ mults/input point}$$
$$\frac{6.2^{n-1}(n-1)+12}{2^n - 3L + 3} \text{ adds/input point} \qquad (60)$$

*per octave (more precisely, twice this complexity per cell). Table III shows that the resulting complexities, when minimized against $N$, are significantly lower than* (58) *for large lengths only, although they slightly reduce the complexity as soon as $L \geq 8$. The price to pay is a more involved implementation, with much larger FFT lengths.*

### E. CWT Algorithms for Short Filters

Fast running FIR algorithms with decimation ratios $R = 2, 3, 5$, described in Section III-E, can easily be applied to the CWT case. Furthermore, in one elementary cell of Fig.

7(b), both filters share the same input and pre-additions can therefore be combined on the single input.

Table III lists the resulting complexities, using the fast running FIR decomposition that minimizes the total number of operations (multiplications + additions). When two different decompositions yield the same number of operations, we have chosen the one that minimizes the number of multiplications. As in Section III-E, we have restricted ourselves to at most two nested fast running FIR algorithms. Otherwise the resulting implementation becomes more involved.

Table III shows that short-length CWT algorithms are more efficient than the FFT-based algorithm of Section V-C for lengths up to 20. It even remains more efficient than the generalized algorithm of Section V-D (which gathers two octaves) for lengths up to 12. If the CWT is computed with medium filter lengths so as to maintain the complexity at a reasonable level, short-length algorithms may be a good trade-off both in terms of structure and complexity.

### F. Unequal Filter Lengths

The situation here is almost the same as the one discussed earlier (see Section III-G for more details). Assume that the length of $h[n]$ is much larger than the length of $g[n]$ ($L_h \gg L_g$). If the FFT-based algorithm described in Section V-C is modified to compute the short filter directly as inner products and use the FFT only for the long one, then one obtains a more efficient FFT-based algorithm only for very short low-pass filters (although the gain made here is comparatively larger than in the WST case). Both variations of FFT-based algorithms require about the same complexity (again we have left out linear phase considerations). The conclusions are the same as in section III-G.

It should be clear from the discussions made in this paper that using appropriate methods it is always possible to take advantage of specific constraints and to reduce (more or less significantly) the complexity involved in the computation of wavelet coefficients in any case of interest (i.e., for all filter lengths). In any case, the tables provided in this paper for the general case give a rough idea of possible savings achievable by the various methods.

### G. Other CWT Algorithms

Several algorithms for computing CWT coefficients, which differ notably from those described above, have been proposed recently. Gopinath and Burrus [12] proposed a method that also uses DWT's. 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 to precompute the kernel and to obtain a method particularly suited to the computation of CWT coefficients with respect to several wavelets. In contrast, the algorithms described above "oversample" the discretization $a = 2^j$, $b = k2^j$ by computing more coefficients directly from the signal. This should result in a faster implementation compared to [12] which uses a computationally expensive kernel expansion.

Another CWT algorithm, which uses the scaling property of wavelets $\psi(t) \to a^{-1/2}\psi(t/a)$ rather than the convolutional form of (1), (2)—$x(t)$ convolved with $a^{-1/2}\psi(t/a)$—has been proposed by the Bertrands and Ovarlez [2]. Let us briefly outline the derivation of this algorithm. Write (2) in the frequency domain, assuming that the signal $x(t)$ and wavelet $\psi(t)$ are complex analytic. This gives

$$\mathrm{CWT}\{x(t); a, b\} = \int_0^{+\infty} X(f) e^{2i\pi f b} \sqrt{a}\, \psi^*(af)\, df, \tag{61}$$

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

$$\mathrm{CWT}\{x(t); a, b\} = \int_R X(e^{\varphi}) e^{\varphi/2} e^{2i\pi e^{\varphi} b} \psi(e^{\alpha+\varphi}) e^{\frac{\alpha+\varphi}{2}}\, d\varphi. \tag{62}$$

After suitable discretization, this correlation can be performed using an FFT algorithm. As stated in [2], 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 $\varphi = \ln f$:

$$M_x(\beta) = \int_{f>0} X(f) f^{-1/2+2i\pi\beta}\, df = \int e^{\varphi/2} X(e^{\varphi}) e^{2i\pi\beta\varphi}\, d\varphi. \tag{63}$$

As a result, the FFT's involved in the computation of (62) are "discrete Mellin transforms," as defined in [2].

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 (2). One obtains a convolution in $\alpha = \ln a$

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

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

Both algorithms (62), (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 (62) or (64) is difficult to estimate. Finally, in contrast with the octave-by-octave CWT implementation previously described the regular structure of time shifts $b$ has completely disappeared, and one has to recompute the input for each value of $b$. As a result, the complexity of such algorithms (about two

FFT's 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 accelerated algorithm of Section V-C.

However, a nice property of the Bertrands–Ovarlez algorithms (62), (64) is that the CWT coefficients are computed for all desired values of $\ln a$ at the same time (for given value of $b$), which is much more straightforward than in the algorithms previously described. 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.

## VI. Conclusion

This paper has provided several methods for implementing efficiently various kinds of wavelet transforms, from the fully discrete version to the fully continuous one, and for any type of wavelet. 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. A detailed treatment of questions of accuracy remains a topic for future investigation.

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").

Two different classes of fast algorithms have been derived: the first one is based on the FFT, and is efficient for medium to large wavelet prototypes. The second one is based on short-length "fast running FIR algorithms" [23] and is efficient for small to medium size filters. Compared to the situation encountered for fixed coefficient filtering [23], [24], DWT fast algorithms are useful for shorter filters, while the reduction of the arithmetic complexity, although substantial, is lower. The modified "CWT" algorithms are efficient for even shorter wavelet prototypes than in the DWT case, with an improvement which is asymptotically greater.

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.

## Acknowledgment

The authors would like to thank anonymous reviewers for useful comments and suggestions that helped improve the presentation of this paper. Fruitful, passionate discussions with Prof. M. Vetterli of Columbia University, New York, NY, are also gratefully acknowledged.

## References

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Imaging coding using vector quantization in the wavelet transform domain," in *Proc. 1990 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Albuquerque, NM, Apr. 3–6, 1990, pp. 2297–2300.

[2] 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, Apr. 3–6, 1990, pp. 1603–1606.

[3] A. Cohen, I. Daubechies, and J. C. Feauveau, "Biorthogonal bases of compactly supported wavelets," Tech. Memo. #11217-900529-07, AT&T Bell Labs.; to appear in *Comm. Pure Applied Math.*

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

[5] ——, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Trans. Inform. Theory*, vol. 36, pp. 961–1005, Sept. 1990.

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

[7] G. Deslauriers and S. Dubuc, "Symmetric iterative interpolation processes," *Constructive Approximation*, vol. 5, pp. 49–68, 1989.

[8] P. Duhamel, "Implementation of split-radix FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-34, pp. 285–295, Apr. 1986.

[9] P. Dutilleux, "An implementation of the 'Algorithme à Trous' to compute the wavelet transform," in *Wavelets, Time-Frequency Methods and Phase Space*, J. M. Combas, A. Grossmann, and Ph. Tchamitchian, Eds. Berlin: Springer, IPTI, 1989, pp. 298–304.

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

[11] P. Flandrin, "Some aspects of non-stationary signal processing with emphasis on time-frequency and time-scale methods," in *Wavelets, Time-Frequency Methods and Phase Space*, J. M. Combes, A. Grossmann, and Ph. Tchamitchian, Eds. Berlin: Springer, IPTI, 1989, pp. 68–98.

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

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

[14] A. Grossmann and R. Kronland-Martinet, "Time and scale representations obtained through continuous wavelet transforms," in *Proc. Int. Conf. EUSIPCO'88, Signal Processing IV: Theories and Applications*, J. L. Lacoume *et al.*, Eds. New York: Elsevier Science Pub., 1988, pp. 475–482.

[15] C. E. Heil and D. F. Walnut, "Continuous and discrete wavelet transforms," *SIAM Rev.*, vol. 31, no. 4, pp. 628–666, Dec. 1989.

[16] C. E. Heil, "Wavelets and frames," in *Signal Processing, Part I: Signal Processing Theory, vol. 22*, L. Auslander, T. Kailath, S. Mitter, Eds., Institute for Mathematics and its Applications. New York: Springer, 1990.

[17] 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 *Wavelets, Time-Frequency Methods and Phase Space*, J. M. Combes, A. Grossmann, and Ph. Tchamitchian, Eds. Berlin: Springer, IPTI, 1989, pp. 286–297.

[18] R. Kronland-Martinet, J. Morlet, and A. Grossmann, "Analysis of sound patterns through wavelet transforms," *Int. J. Pattern Recogn. Artificial Intell.*, vol. 1, no. 2, 1987. pp. 273–302.

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

[20] ——, "Multifrequency channel decompositions of images and wavelet models," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 2091–2110, Dec. 1989.

[21] S. Mallat and S. Zhong, "Signal characterization from multiscale edges," in *Proc. 10th Int. Conf. Pattern Recogn., Pattern Recogn., Syst. Applica.*, Los Alamitos, CA, 16-21 June 1990, pp. 891–896.

[22] Y. Meyer, *Ondelettes et Opérateurs, Tome I*. Paris: Herrmann, 1990.

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

[26] M. J. Shensa, "Affine wavelets: Wedding the Atrous and Mallat algorithms," to appear in *IEEE Trans. Acoust., Speech, Signal Proc.*

[27] M. J. T. Smith and T. P. Barnwell, "Exact reconstruction for tree-structured subband coders," *IEEE Trans. Acoust., Speech,*

*Signal Processing*, vol. ASSP-34, pp. 434–441, June 1986.

[28] P. P. Vaidyanathan and P.-Q. Hoang, "Lattice structures for optimal design and robust implementation of two-channel perfect-reconstruction QMF banks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 81–94, Jan. 1988.

[29] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proc. IEEE*, vol. 78, pp. 56–93, Jan. 1990.

[30] M. Vetterli, "Analyse, Synthèse et Complexité de Calcul de Bancs de Filtres Numériques," Ph.D. thesis, Ecole Polytechique Fédérale de Lausanne, 1986.

[31] ——, "Running FIR and IIR filtering using multirate filter banks," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-36, pp. 730–738, May 1988.

[32] M. Vetterli and C. Herley, "Wavelets and filter banks: Relationships and new results," in *Proc. 1990 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Albuquerque, NM, Apr. 3–6, 1990, pp. 1723–1726.

[33] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," to appear in *IEEE Trans. Acoust., Speech, Signal Processing*.

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