FAST ALGORITHMS FOR THE CONTINUOUS WAVELET TRANSFORM

Olivier Rioul

Centre National d'Etudes des Télécommunications (CNET)
Centre Paris B, PAB/RPE/ETP
38-40, rue du Général-Leclerc, F-92131 Issy-Les-Moulineaux, France.

ABSTRACT

Wavelet transforms are becoming of increasing interest for signal analysis (under the form of Continuous Wavelet Transforms (CWT)) and for image compression (under the form of Discrete Wavelet Transforms (DWT)). The computational structure of the DWT was soon recognized to be an octave-band filter bank. This paper shows that filter banks arise also naturally when implementing the CWT. We determine under which conditions the CWT may be computed exactly using discrete filter banks and derive fast CWT algorithms. The complexity of the resulting algorithms increases linearly with the number of octaves. They are easily implemented by repetitive application of identical cells, to which various methods are applied for reducing the number of operations: FFT algorithms are most efficient for large filter lengths; for small lengths, "fast running FIR" algorithms are preferred.

I. DEFINITIONS

We introduce various definitions of wavelet transforms. Our terminology follows the classical notations for Fourier transforms.

The *Continuous Wavelet Transform* (CWT) [1], [2] is a representation of analog signals $x(t)$ depending on two parameters.

$$\mathrm{CWT}(x(t);\, a,\, b) = a^{-1/2} \int x(t)\, \psi^*\left(\frac{t-b}{a}\right) \mathrm{d}t,\ a \neq 0. \qquad (1)$$

The *time-shift* parameter b and the *scale* parameter a vary continuously. One may clearly restrict to $a > 0$ when $x(t)$ and $\psi(t)$ are real-valued or complex analytic. The function $\psi(t)$ is called the *analyzing wavelet.*

When the time-scale parameters are suitably discretized (e.g., $a=2^j$ and $b=k2^j$, where j and k are integers) and when the time t remains continuous, the wavelet coefficients (1) are coefficients of a wavelet series. When $\psi(t)$ is suitably chosen [3], the signal is decomposed into this wavelet series.

Although the initials "DWT" are sometimes used for denoting wavelet series' coefficients, we here use the terminology "Discrete Wavelet Transform" (DWT) [4] when *both* time-scale parameters and time are discrete.

$$\mathrm{DWT}(x[n];\, 2^j,\, k2^j) = \sum_n x[n]\, h_j[n - 2^j k], \qquad (2)$$

Here $j=1, 2, \ldots$; $k=0, \pm1, \pm2, \ldots$; the definition holds to positives j's so that the sampling rate in k of $\mathrm{DWT}(x[n];\, 2^j,\, k2^j)$ is smaller than that of the analyzed signal $x[n]$. The *discrete wavelet* $h_j[n]$ corresponding to the jth octave is a substitute to $2^{-j/2}\psi(2^{-j}t)$; it is a high-pass filter, defined by induction according to

$$h_{j+1}[n] = \sum_m h_j[m]\, g[n - 2m]. \qquad (3)$$

The sequence $g[n]$ is a low-pass filter used to perform this "discrete up-scaling" (3) [4], a discrete version of the operation $\psi(t) \to \frac{1}{\sqrt{2}}\psi(t/2)$.

The DWT can be equivalently seen [2], [4] as an analysis octave-band filter bank [5], whose computational flow-graph is depicted in Figure 1. The classical low-pass and high-pass filters occurring in a filter bank are exactly $g[n]$ and $h_1[n]=h[n]$, respectively. Coding schemes based on DWT's generate additional constraints on $g[n]$ and $h[n]$. Since we use here the DWT only as an intermediate step to compute the CWT, these constraints are not considered here.

In this paper, we focus on the computation of the CWT for signal analysis purposes. The objective is therefore to compute the CWT on a fine, regular sampling in the time-scale plane (b, a) of the form $a = a_0^j$, $b = kT$, where $a_0>1$ and T is the sampling period of the discretized signal (we assume $T=1$ for convenience in the following). This will be done in several steps. We first determine the conditions under which the DWT computes the wavelet series' coefficients on the dyadic grid $a=2^j$, $b=k2^j$. The missing points in the CWT computation are then obtained as follows. We "fill the holes" in time, i.e., compute the CWT at $a=2^j$, $b=k$, and finally extend the octave by octave computation $a=2^j$ to arbitrary resolution in scale $a=a_0^j$.

II. DWT COMPUTATION OF WAVELET SERIES' COEFFICIENTS

In this section we derive conditions under which the DWT exactly computes $\mathrm{CWT}(x(t);\, 2^j,\, k2^j)$. We consider the general case where the discrete signal $x[n]$ and wavelets $h_j[n]$ at the jth octave do not necessarily result from perfect sampling of their analog counterparts $x(t)$ and $2^{-j/2}\psi(2^{-j}t)$, respectively. More precisely, assume that the discrete signal $x[n]$ results from $x(t)$ by a non-perfect sampler such that

$$x(t) = \sum_n x[n]\chi(t - n). \qquad (4)$$

CH2977-7/91/0000-2213 $1.00 © 1991 IEEE

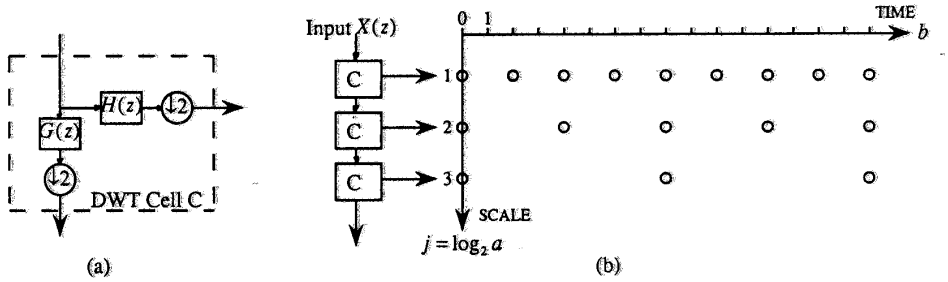

Figure 1.(a) Elementary DWT cell (b) DWT flow graph and corresponding sampling grid in the time-scale plane.

Also assume a similar correspondence for wavelets, of the form

$$2^{-j/2}\psi(2^{-j}t) = \sum_n h_j[n]\phi(t-n). \qquad (5)$$

Since we want to use the DWT as an intermediate step to compute the CWT, we require that the discrete wavelets $h_j[n]$ be determined from $h_1[n] = h[n]$ by (3), as in the DWT. It follows from (3), (5), that $\phi(t)$ is a solution to a "two-scale difference equation" [6],

$$\phi(t) = \sum_n g[n]\phi(2t-n). \qquad (6)$$

For example, if $\psi(t/2)$ is band-limited to [-1/2; 1/2], then, by Nyquist's sampling theorem, $\phi(t) = \frac{\sin \pi t}{\pi t}$ is a solution that ensures $h_j[n]=2^{-j/2}\psi(2^{-j}n)$. Another solution for $\phi(t)$ is the basic spline interpolating function of some degree, in which case the $g[n]$'s are the corresponding binomial coefficients. In general, given the basic wavelet $\psi(t)$, it is possible to find a solution $\phi(t)$ and a sequence $h[n]$ such that (5) is satisfied with great accuracy. The above spline example is characteristic: the determination of $\phi(t)$ and $h[n]$ reduces in this case to a classical spline curve fitting problem.

By expanding (4), (5) into the expression of $\mathrm{CWT}(x(t); 2^j, k2^j)$ we obtain, after some calculation,

$$\mathrm{CWT}(x(t); 2^j, k2^j) = \mathrm{DWT}(x'[n]; 2^j, k2^j) \qquad (7)$$

where $x'[n]$ is a corrected version of $x[n]$, namely $x[n]$ filtered by the sequence $\int \chi(t)\phi^*(t-n)\,dt$.

III. OCTAVE-BY-OCTAVE COMPUTATION OF THE CWT

Similarly as (7) was derived, one easily obtains

$$\mathrm{CWT}(x(t); 2^j, k) = \sum_n x[n]\, h_j[n-k] \qquad (8)$$

under the same conditions (4)-(6). The right-hand side of (8) is a new transform, which will be refered to in this paper as the "Continuous Discrete Wavelet Transform" (CDWT).

$$\mathrm{CDWT}(x[n]; 2^j, k) = \sum_n x[n]\, h_j[n-k] \qquad (9)$$

The CDWT can be computed using only DWT's (2) since, when $k=2^j k_1+k_2$, $k_2=0, \ldots, 2^j-1$, we have

$$\mathrm{CDWT}(x[n]; 2^j, k) = \mathrm{DWT}(x[n+k_2]; 2^j, k_1 2^j). \qquad (10)$$

Therefore, any octave-band filter bank DWT algorithm, fed by successively delayed inputs, can be used to compute the CWT! This is shown in Figure 2 (a),(c): Compared to the DWT computation of Figure 1, DWT's with delays are included to obtain the missing points.

Holschneider *et al.* [7] have derived similar conditions for (8), in a more restrictive framework. They assume that discrete signals and wavelets are obtained by perfect sampling of their analog counterparts, i.e., that the CWT is obtained as a simple discretization of the integral in (1). Therefore $g[n] = \phi(n/2)$ in (6), which implies $g[2n] = 1$ if $n=0$, zero otherwise (this is called the "*à trous*" ("with holes") property in [7]). But a good approximation of the CWT by a CDWT depends crucially on a good approximation of $\psi(t)$ by interpolating the $h[n]$'s by $\phi(t)$ *via* (5). The "à trous" property strongly restricts the choice of $\phi(t)$. For example, high order spline interpolation is not usable under the "à trous" restriction. Thus, the "à trous" CWT algorithm described in [7] is a particular case of a CDWT implementation.

IV. FINER SAMPLING IN SCALE

For signal analysis an octave by octave computation of the CWT is generally not enough. Assume for example that V voices per octave are desired, i.e., that we want to compute the CWT (1) with $a=2^{j/V}$. Since the scale parameter a is only a relative notion (its definition depends on $\psi(t)$), one computes the CWT for $a=2^j 2^{1/V}$ by replacing $\psi(t)$ by $\psi(t2^{-1/V})$. The other points are similarly obtained. The whole computation thus results from successive application of the same algorithm to V slightly stretched analyzing wavelets.

V. FILTER BANK IMPLEMENTATION OF THE CWT

We have seen that when the discrete signal $x[n]$ and analysing wavelet $h[n]$ have been determined, the general computation of the CWT reduces to the computation of a CDWT (9).

Rather than using directly DWT's as in (10), it is convenient to transform (10) into a specific filter-bank implementation shown in Figure 2 (b),(c). In contrast to the DWT filter-bank of Figure 1, there are 2^{j-1} elementary cells at the jth octave in the CDWT filter-bank. Note that all cells are identical but "work" at a different rate: a cell at jth octave has input subsampled by 2^{j-1}. Therefore, if we define the computational complexity as the number of multiplications/additions per input point, the total complexity of the CDWT is exactly J times the complexity of one elementary

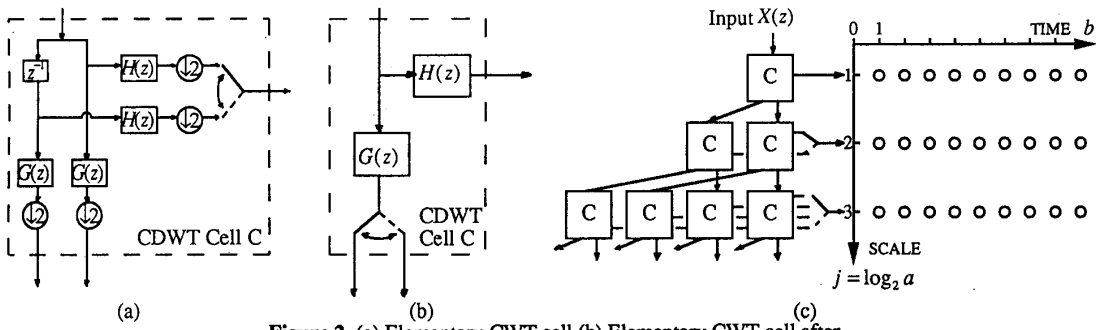

Figure 2. (a) Elementary CWT cell (b) Elementary CWT cell after transformation (c) Octave-by-ocatve CWT flow graph and corresponding sampling grid in the time-scale plane.

cell, where J is the number of octaves on which the CDWT is computed. When V voices per octave are computed as shown in the preceding section, this complexity is simply multiplied by V. As a general rule:

The complexity of a filter bank implementation (via a CDWT*) of a* CWT *grows linearly with the number of octaves.*

This is of course a substantial improvement compared to a direct implementation of (1), that does not take the dilation operation of wavelets into account. This latter requires an exponentially increasing complexity with the number of octaves.

From now on, we assume real data and filters; the complex analytic case yields the same complexity. The criterion of complexity we use in this paper is the total number of operations (mults+adds) required by the algorithm. With today's technology, this criterion is generally more useful than that of the sole number of multiplications.

To implement the CWT it remains to specify an algorithm of the elementary cell of Figure 2. Assume that both filters $G(z)$ and $H(z)$ (corresponding to $g[n]$ and $h[n]$, respectively) are of same length L (otherwise simply pad with zeros; since the filter lengths are often of comparable order, this will not greatly influence the complexity). A direct implementation of the filters requires

$$2L \text{ multiplications per input point}$$
$$2(L\text{-}1) \text{ additions per input point} \qquad (11)$$

for each elementary cell. Hence the total complexity for a direct filter-bank CWT algorithm on J octaves with V voices per octave using L-tap filters is simply JV times the complexity (11).

VI. FFT-BASED ALGORITHM

Significant improvement can be made on (11) for medium and large filter lengths L using FFT computation of filters. With both "overlap-add" or "overlap-save" methods [8], the input is divided into blocks of length B, and the FFT length N must be greater or equal to $B+L-1$ to eliminate wrap-around effects. From now on, we use the "split-radix FFT algorithm" [9] for $N=2^n$, which for real data requires $2^{n-1}(n-3)+2$ (real) multiplications and $2^{n-1}(3n-5)+4$ (real) additions.

Each elementary cell of Figure 2 (b) is therefore carried out by first computing an FFT 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 apply two inverse FFT's on the results. Wrap-around effects are then eliminated in the time-domain and one waits for one block before entering the next cell, so that each cell has input length B. Assuming $B=N-L+1$, this yields

$$(3\,2^{n-1}(n\text{-}1) + 6) / (2^n\text{-}L+1) \text{ mults per input point}$$
$$(9\,2^{n-1}(n\text{-}1) +12) / (2^n\text{-}L+1) \text{ adds per input point} \qquad (12)$$

for each elementary cell. Table I shows the resulting complexities for different lengths, when minimized against N. By deriving the total number of operations with respect to N, one finds that the optimal FFT length is about $N=(0.69\,n+0.31)\,(L\text{-}1)$, with minimized total number of operations per point equal to $6n+2.65 = 6\log_2 L + O(\log_2 \log_2 L)$. This is a significant improvement compared to (11).

This algorithm can be generalized by gathering several consecutive blocks using a method that Vetterli derived for the DWT [10]. The idea is to avoid subsequent inverse FFT's and FFT's by performing the sub-sampling in the frequency domain. The FFT length is then necessarily halved at each stage, whereas the filter lengths remain constant. Therefore, these schemes have two disadvantages. First, the structure of computation is less regular (FFT's of different lengths). Secondly, 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!

We here provide an example, where two octaves are gathered together. Three elementary cells of Figure 2 are merged into one 1-input, 7-output cell that covers two octaves. The FFT length $N=2^n$ must here be greater than or equal to $B+3(L-1)$ to avoid wrap-around effects, where B is the input block length. This results in

$$(\,2^{n-1}(2n\text{-}1) + 6) / (2^n\text{-}3L+3) \text{ mults per input point}$$
$$(6\,2^{n-1}(n\text{-}1) +12) / (2^n\text{-}3L+3) \text{ adds per input point} \qquad (13)$$

per octave, assuming the CWT is computed on an even number of octaves. Table I shows the resulting complexities, when minimized against n. They are significantly better than (12) for large lengths only (here $L \geq 8$). The price to pay is a more involved implementation, with much larger FFT lengths.

VII. SHORT-LENGTH ALGORITHM

For short lengths L, the complexity of two filters of length L (11) required for each elementary cell of Figure 2 can be significantly reduced by applying short-length "fast running FIR" algorithms [11]. These algorithms are interesting because they retain partially the multiply/accumulate structure and therefore are easily implemented. In such algorithms, the involved sequences (input, output and filter) are decimated with some ratio M. The initial filter of length L is then decomposed as follows. The decimated inputs are first combined with some pre-additions. The resulting sequences are then input to several *sub-filters* of length L/M. The outputs are finally recombined with some post-additions and delays to provide the exact decimated outputs. Further application of this algorithm is feasible, since the sub-filters are still amenable to further decomposition.

Here, since the elementary cell of the CWT has two filters sharing the same input, all pre-additions can be combined on a single input. In this paper we use three different fast running FIR algorithms for M=2, 3 and 5, that can be found in [11]. Their number of pre-additions, sub-filters and post-additions are, respectively, (2,3,2) for M=2, (4,6,6) for M=3, and (14, 12, 26) for M=5.

Table I lists the resulting complexities, using the fast running FIR decomposition that minimizes the total number of operations (mults+adds). When two different decompositions yield the same total number of operations we have chosen the one that minimizes the number of multiplications. We have also restricted ourselves to at most two successive applications of fast running FIR algorithms. Beyond this depth, the resulting implementation becomes complicated.

Table I shows that short-length FIR algorithms are more efficient than the simple FFT-based algorithm of section VI for lengths up to 20. It even remains more efficient than the two-octave Vetterli FFT-based algorithm for lengths up to 12. We may envision that the CWT will generally be computed with medium filter lengths (e.g., $4 \le L \le 16$) to maintain the complexity at a reasonable level. In this respect, this short-length algorithm gives the best alternative we could find.

Acknowledgement. Part of this work was done with P. Duhamel at CNET Paris, whom the author wishes to thank here.

REFERENCES

[1] *Wavelets*, J.M. Combes *et al.* eds., Springer, Berlin, 1989.

[2] O. Rioul and M. Vetterli, "Wavelet Transforms in Signal Processing," *IEEE Signal Processing Magazine*, fall 1991, to appear.

[3] I. Daubechies, "The Wavelet Transform, Time-Frequency Localization and Signal Analysis," *IEEE Trans. Info. Theory*, Vol. 36, No. 5, pp. 961-1005, Sept. 1990.

[4] O. Rioul, "A Unifying Discrete-Time Theory for Filter Banks, Wavelet and Pyramid Transforms," *IEEE Trans. Signal Proc.*, submitted 1990.

[5] M.J.T. Smith and T. P. Barnwell III, "Exact Reconstruction Techniques for Tree-Structured Subband Coders," *IEEE Trans. Acoust., Speech, Signal Proc.*, Vol. ASSP-34, June 1986.

[6] I. Daubechies and J.C. Lagarias, "Two-Scale Difference Equations I," *SIAM J. Math. Anal.*, to appear.

[7] M. Holschneider, R. Kronland, J. Morlet, and P. Tchamitchian, "A Real-Time Algorithm for Signal Analysis with the Wavelet Transform," in [1].

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

[9] P. Duhamel, "Implementation of 'Split-Radix' FFT Algorithms for Complex, Real, and Real-Symmetric Data," *IEEE Trans. Acoust., Speech, Signal Proc.*, Vol. ASSP-34, No.2,pp. 285-295, April 1986.

[10] M. Vetterli and C. Herley, "Wavelets and Filter Banks: Relationships and New Results," in *Proc. ICASSP'90*, Albuquerque, NM, pp. 1723-1726, Apr. 3-6, 1990.

[11] Z.J. Mou and P. Duhamel, "Short-Length FIR Filters and Their Use in Fast Nonrecursive Filtering," *IEEE Trans. Signal. Proc.*, June 1991.

Filter length L	Direct Method eq.(11)	FFT-based algorithm eq.(12)	FFT-based (two octaves merged) eq.(13)	Short-Length Algorithm
2	4 + 2	4 + 10 (4)	4.8 + 12 (16)	*3 + 3* (2)
3	6 + 4	5 + 14 (8)	5.8 + 15.2 (32)	*4 + 5.3* (3)
4	8 + 6	6 + 16.8 (8)	6.5 + 17.2 (32)	*4.5 + 7.5*(2x2)
5	10 + 8	6.5 + 19 (16)	6.9 + 18.7 (64)	*4.8 + 13.2* (5)
6	12 + 10	7.1 + 20.7 (16)	7.3 + 19.8 (64)	*6 + 11* (2x3)
8	16 + 14	7.9 + 23.5 (32)	7.8 + 21.6 (128)	*9 + 12* (2x2)
9	18 + 16	8.2 + 24.5 (32)	8.1 + 22.3 (128)	*8 + 16* (3x3)
10	20 + 18	8.6 + 25.6 (32)	8.3 + 22.9 (128)	*7.2 + 20.4*(5x2)
12	24 + 22	9.2 + 27.4 (64)	8.6 + 24.2 (256)	*12 + 17* (2x3)
15	30 + 28	9.7 + 29 (64)	*9 + 25.2* (256)	9.6 + 26 (5x3)
16	32 + 30	9.9 + 29.6 (64)	*9.1 + 25.5* (256)	18 + 21 (2x2)
18	36 + 34	10.3 + 30.9 (64)	*9.4 + 26.3* (256)	16 + 24 (3x3)
20	40 + 38	10.6 + 31.8 (128)	*9.6 + 27* (512)	14.4 + 27.6(5x2)
24	48 + 46	11 + 33 (128)	*9.8 + 27.8* (512)	24 + 29 (2x3)
25	50 + 48	11.1 + 33.3 (128)	*9.9 + 27.9* (512)	11.5 + 44.9(5x5)
27	54 + 52	11.3 + 34 (128)	*10 + 28.3* (512)	24 + 32 (3x3)
30	60 + 58	11.7 + 35 (128)	*10.2+ 28.9* (512)	19.2 + 35.6(5x3)
32	64 + 62	11.9 + 35.7 (128)	*10.4+ 29.4* (512)	36 + 39 (2x2)
64	128 + 126	13.7 + 41.1 (512)	*11.6+ 33.1* (2048)	72 + 75 (2x2)
128	256 + 254	15.4 + 46.2 (1024)	*12.7+ 36.4* (4096)	144 + 147 (2x2)

Table I. Complexity Results (per octave per input point) for various filter lengths. Complexities are shown in the form *mults+adds*. The minimum achieved is typed in *italics*. FFT lengths (N=2^r) and successive decimation ratios M (in correct order and in italics) appear between brackets (see text).