REAL-TIME AUDIO PROCESSING VIA SEGMENTED WAVELET TRANSFORM

Pavel Rajmic and Jan Vlach

Dept. of Telecommunications
FEEC, Brno University of Technology
Czech Republic
rajmic@feec.vutbr.cz

ABSTRACT

In audio applications it is often necessary to process the signal in "real time". The method of segmented wavelet transform (SegWT) makes it possible to compute the discrete-time wavelet transform of a signal segment-by-segment, not using the classical "windowing". This means that the method could be utilized for wavelet-type processing of an audio signal in real time, or alternatively in case we just need to process a long signal, but there is insufficient computational memory capacity for it (e.g. in the DSPs). In the paper, the principle of the segmented forward wavelet transform is explained and the algorithm is described in detail.

1. INTRODUCTION

There are a number of theoretical papers and practical applications of the wavelet transform. However, all of them approach the problem from such a point of view as if we knew the whole of the signal (no matter how long it is). Due to this assumption, we cannot perform the wavelet-type signal processing in real time in this sense. Of course there are real-time applications of the wavelet type, but, all of them utilize the principle of overlapping segments of the "windowed" signal (e.g. [1]). In the reconstruction part of their algorithms they certainly introduce errors into the processing, because the segments are assembled using weighted averages.

Processing a signal in "real time" actually means processing it with minimum delay. A signal, which is not known in advance, usually comes to the input of a system piecewise, by mutually independent segments that have to be processed.

The new method, the so-called segmented wavelet transform (SegWT – we introduce abbreviation SegWT (Segmented Wavelet Transform), because SWT is already reserved for stationary wavelet transform), enables this type of processing. It has a great potential application also in cases when it is necessary to process a long signal off-line and no sufficient memory capacity is available. It is then possible to use this method for equivalent segmentwise processing of the signal and thus save the storage space. In this sense SegWT corresponds to the overlap-add algorithm in Fourier-type linear filtering.

Another possible application of the SegWT algorithm is the instantaneous visualization of signal using an imaging technique referred to as "scalogram", see Fig. 1. The decomposition depth is $d = 5$ in this Figure. The bigger is the absolute value of the single coefficient, the whiter is the respective cell in the graph. In fact, plotting scalogram is a technique very similar to plotting a spectrogram in real time. In wavelet transformation (represented by FIR filters) there is an advantage in that the signal need not be weighted with windows, which results in a distortion of the frequency information, as is the case with the spectrogram. Moreover, there is one more good thing about it: a scalogram created by means of the SegWT is quite independent of the chosen length of segment.

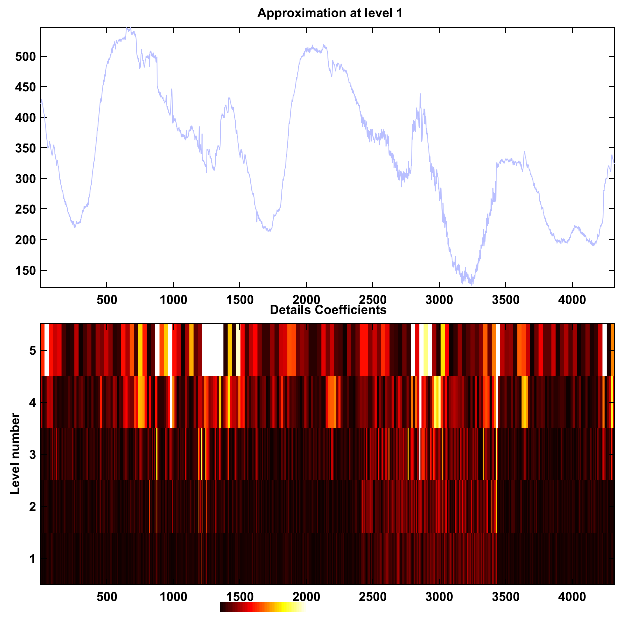

Figure 1: Signal (top) and its scalogram (bottom). Scalogram is a type of graph representing the frequency contents of a signal in time. It is constructed from the wavelet coefficients.

In the available literature, this way of performing the wavelet transform is practically neglected, and this was the reason why our effort was devoted to developing modified algorithm. In fact, a modified method of *forward* wavelet transform is presented in this paper.

2. THE CLASSICAL DTWT ALGORITHM

Algorithm 2.1: (decomposition pyramidal algorithm DTWT) Let $\mathbf{x}$ be a discrete input signal of length s, the two wavelet decomposition filters of length m are defined, highpass $\mathbf{g}$ and lowpass $\mathbf{h}$, d is a positive interger determining the decomposition depth. Also, the type of boundary treatment [2, ch. 8] must be known.

1. We denote the input signal $\mathbf{x}$ as $\mathbf{a}^{(0)}$ and set $j = 0$.
2. One decomposition step:

(a) *Extending the input vector.* We extend $\mathbf{a}^{(j)}$ from both the left and the right side by $(m-1)$ samples, according to the type of boundary treatment.

(b) *Filtering.* We filter the extended signal with filter $\mathbf{g}$, which can be expressed by their convolution.

(c) *Cropping.* We take from the result just its central part, so that the remaining "tails" on both the left and the right sides have the same length $m-1$ samples.

(d) *Downsampling (decimation).* We downsample the resultant vector.

We denote the resulting vector $\mathbf{d}^{(j+1)}$ and store it.We repeat items (b)–(d), now with filter $\mathbf{h}$, denoting and storing the result as $\mathbf{a}^{(j+1)}$.

3. We increase j by one. If it now holds $j < d$, we return to item 2., in the other case the algorithm ends.

Remark. After algorithm 2.1 has been finished, we hold the wavelet coefficients stored in $d+1$ vectors $\mathbf{a}^{(d)}, \mathbf{d}^{(d)}, \mathbf{d}^{(d-1)}, \ldots, \mathbf{d}^{(1)}$.

3. THE METHOD OF SEGMENTED WAVELET TRANSFORM

3.1. Motivation and Aim of the Method

Regularly used discrete-time wavelet transform (see Section 2) is suitable for processing signals "off-line", i.e. known before processing, even if very long. The task for the segmented wavelet transform, SegWT, is naturally to allow signal processing by its segments, so that in this manner we get the same result (same wavelet coefficients) as in the ordinary DTWT case. In this problem, the following parameters play a crucial role.

m wavelet filter length, $m > 0$,
d transform depth, $d > 0$,
s length of segment, $s > 0$.

The derivation of the SegWT algorithm requires a very detailed knowledge of the DTWT algorithm. Thanks to this it is possible to deduce fairly sophisticated rules how to handle the signal segments. We have found that in dependence on m, d, s, it is necessary to extend every segment from the left by an exact number of samples from the preceding segment and from the right by another number of samples from the subsequent segment. However, every segment has to be extended by a different length from the left and the right, and these lengths can also differ from segment to segment! Also the first and the last segments have to be handled in a particular way.

3.2. Important Theorems Derived from the DTWT Algorithm

Before we introduce detailed description of the SegWT algorithm, several theorems must be presented. More of them and their proofs can be found in [3, ch. 8]. We assume that the input signal $\mathbf{x}$ is divided into $S \geq 1$ segments of equal length s. Single segments will be denoted ${}^1\mathbf{x}, {}^2\mathbf{x}, \ldots, {}^S\mathbf{x}$. The last one can be of a length lower than s. See Fig. 2.

By the formulation that *two sets of coefficients from the k-th decomposition level follow-up on each other* we mean a situation when two consecutive segments are properly extended see Figs. 2, 3, so that applying the DTWT, with step 2(a) omitted, of depth k separately to both the segments and joining the resultant coefficients together lead to the same set of coefficients as computing it via the DTWT applied to the two segments joined first.

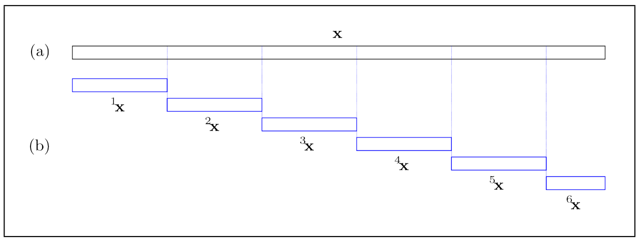

Figure 2: Scheme of signal segmentation. The input signal $\mathbf{x}$ (a) is divided into segments of equal length, the last one can be shorter than this (b); the n-th segment of $\mathbf{x}$ is denoted by ${}^n\mathbf{x}$.

Theorem 3.1: *In case that the consecutive segments have*

$$r(k) = (2^k - 1)(m-1) \tag{1}$$

common input signal samples, the coefficients from the k-th decomposition level follow-up on each other.

Thus, for a decomposition depth equal to d it is necessary to have $r(d) = (2^d - 1)(m-1)$ common samples in the two consecutive extended segments.

The aim of the following part is to find the proper extension of every two consecutive signal segments. We will show that the length of such extension must comply with the strict rules.

The extension of a pair of consecutive segments, which is of total length $r(d)$, can be divided into the right extension of the first segment (of length R) and the left extension of the following segment (of length L), while $r(d) = R + L$. However, the lengths $L \geq 0, R \geq 0$ cannot be chosen arbitrarily. The lengths L, R are not uniquely determined in general. The formula for the choice of extension $L_{\max}$, which is unique and the most appropriate in case of real-time signal processing, is given in Theorem 3.2.

Theorem 3.2: *Let a segment be given whose length including its left extension is l. The maximal possible left extension of the next segment, $L_{\max}$, can be computed by the formula*

$$L_{\max} = l - 2^d \operatorname{ceil}\left(\frac{l - r(d)}{2^d}\right). \tag{2}$$

The minimal possible right extension of the given segment is then

$$R_{\min} = r(d) - L_{\max}. \tag{3}$$

For the purposes of the following text, it will be convenient to assign the number of the respective segment to the variables $L_{\max}, R_{\min}, l$, i.e. the left extension of the n-th segment will be of length $L_{\max}(n)$, the right extension will be of length $R_{\min}(n)$ and the length of the original n-th segment with the left extension joined will be denoted $l(n)$. Using this notation we can rewrite equation (3) as

$$R_{\min}(n) = r(d) - L_{\max}(n+1). \tag{4}$$

Let us now comment on the special situation of the first or the last segment. These naturally represent the "boundaries" of the signal. The discrete-time wavelet transform uses several modes how to treat the boundaries and we must preserve these modes also in our modified algorithm. Therefore we must treat the first

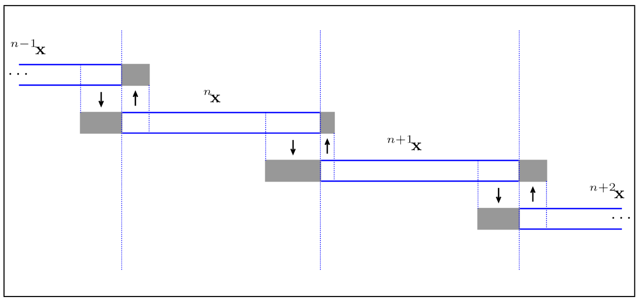

Figure 3: Illustration of extending of the segments.

and the last segment separately and a bit differently from the other segments. For details and proofs we again refer to [3]. The appropriate procedure is to extend the first segment from the left by $r(d)$ zero samples, i.e. $L_{\max}(1) = r(d)$, and to process it using Algorithm 3.7. Similarly the last segment has to be extended by $r(d)$ zeros from the right and processed using Algorithm 3.8.

Theorem 3.3: *The length of the right extension of the n-th segment, $n = 1, 2, \dots, S-2$, must comply with*

$$R_{\min}(n) = 2^d \operatorname{ceil}\left(\frac{ns}{2^d}\right) - ns, \qquad (5)$$

and the length of the left extension of the $(n+1)$-th segment is $L_{\max}(n+1) = r(d) - R_{\min}(n)$.

Remark. From (5) it is clear that $R_{\min}$ is periodic with respect to s with period 2^d, i.e. $R_{\min}(n + 2^d) = R_{\min}(n)$. This relation and also some more can be seen in Table 1.

Theorem 3.4: ***(on the total length of segment)***
After the extension the n-th segment (of original length s) will be of total length

$$\sum(n) = r(d) + 2^d \left[\operatorname{ceil}\left(\frac{ns}{2^d}\right) - \operatorname{ceil}\left(\frac{(n-1)s}{2^d}\right)\right]. \qquad (6)$$

This expression can acquire only one of two values, either

$$r(d) + 2^d \operatorname{ceil}\left(\frac{s}{2^d}\right) \quad \textit{or} \quad r(d) + 2^d \operatorname{ceil}\left(\frac{s}{2^d}\right) - 2^d. \qquad (7)$$

3.3. The Algorithm of Segmented Wavelet Transform

The algorithm SegWT works such that it reads (receives) single segments of the input signal, then it extends – overlaps them in a proper way, then it computes the wavelet coefficients in a modified way and, in the end, it easily joins the coefficients.

Algorithm 3.5: Let the wavelet filters $\mathbf{g}, \mathbf{h}$ of length m, the decomposition depth d, and the boundary treatment mode be given. The segments of length $s > 0$ of the input signal $\mathbf{x}$ are denoted ${}^1\mathbf{x}, {}^2\mathbf{x}, {}^3\mathbf{x}, \dots$. The last segment can be shorter than s.

1. Set $N = 1$.
2. Read the first segment, ${}^1\mathbf{x}$, and label it "current". Extend it from the left by $r(d)$ zero samples.
3. ***If*** the first segment is at the same time the last one
   (a) It is the case of regular wavelet transform. Compute the DTWT of this single segment using Algorithm 2.1.
   (b) The Algorithm ends.
4. Read $(N+1)$-th segment and label it "next".
5. ***If*** this segment is the last one
   (a) Join the current and next segment together and label it "current". (The current segment becomes the last one now.)
   (b) Extend the current vector from the right by $r(d)$ zero samples.
   (c) Compute the DTWT of depth d from the extended current segment using Algorithm 3.8.

   Otherwise

   (d) Compute $L_{\max}$ for the next segment and $R_{\min}$ for the current segment (see Theorem 3.2).
   (e) Extend the current segment from the right by $R_{\min}$ samples taken from the next segment. Extend the next segment from the left by $L_{\max}$ samples taken from the current segment.
   (f) ***If*** the current segment is the first one, compute the DTWT of depth d from the extended current segment using Algorithm 3.7. ***Otherwise*** compute the DTWT of depth d from the extended current segment using Algorithm 3.6.
6. Modify the vectors containing the wavelet coefficients by trimming off a certain number of redundant coefficients from the left side, specifically:at the k-th level, $k = 1, 2, \dots$ $\dots, d-1$, trim off $r(d-k)$ coefficients from the left.
7. ***If*** the current segment is the last one, then in the same manner as in the last item trim the redundant coefficients, this time from the right.
8. Store the result as ${}^N\mathbf{a}^{(d)}, {}^N\mathbf{d}^{(d)}, {}^N\mathbf{d}^{(d-1)}, \dots, {}^N\mathbf{d}^{(1)}$.
9. ***If*** the current segment is not the last one
   (a) Label the next segment "current".
   (b) Increase N by 1 and go to item 4.

Remark. If the input signal has been divided into $S > 1$ segments, then $(S-1)(d+1)$ vectors of wavelet coefficients

$$\{{}^i\mathbf{a}^{(d)}, {}^i\mathbf{d}^{(d)}, {}^i\mathbf{d}^{(d-1)}, \dots, {}^i\mathbf{d}^{(1)}\}_{i=1}^{S-1}.$$

are the output of the Algorithm. If we join these vectors together in a simple way, we obtain a set of $d+1$ vectors, which are identical with the wavelet coefficients of signal $\mathbf{x}$.

Next we present the "subalgorithms" of the SegWT method. The second and third algorithms serve to process the first and the last segment.

Algorithm 3.6: This algorithm is identical with Algorithm 2.1 with the exception that we omit step 2(a), i.e. we do not extend the vector.

Algorithm 3.7: This algorithm is identical with Algorithm 2.1 with the exception that we replace step 2(a) by the step:

> *Modify the coefficients of vector $\mathbf{a}^{(j)}$ on positions $r(d-j) - m + 2, \dots, r(d-j)$, as it corresponds to the given boundary treatment mode.*

Algorithm 3.8: This algorithm is identical with Algorithm 2.1 with the exception that we replace step 2(a) by the step:

> *Modify the coefficients of vector $\mathbf{a}^{(j)}$ on positions $r(d-j) - m + 2, \dots, r(d-j)$, as it corresponds to the given boundary treatment mode, however this time taken* from the right side *of $\mathbf{a}^{(j)}$.*

s	n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	...
512	$L_{\max}(n)$	105	105	105	105	105	105	105	105	105	105	105	105	...
	$R_{\min}(n)$	0	0	0	0	0	0	0	0	0	0	0	0	...
	$\sum(n)$	617	617	617	617	617	617	617	617	617	617	617	617	...
513	$L_{\max}(n)$	105	98	99	100	101	102	103	104	105	98	99	100	...
	$R_{\min}(n)$	7	6	5	4	3	2	1	0	7	6	5	4	...
	$\sum(n)$	625	617	617	617	617	617	617	617	625	617	617	617	...
514	$L_{\max}(n)$	105	99	101	103	105	99	101	103	105	99	101	103	...
	$R_{\min}(n)$	6	4	2	0	6	4	2	0	6	4	2	0	...
	$\sum(n)$	625	617	617	617	625	617	617	617	625	617	617	617	...
515	$L_{\max}(n)$	105	100	103	98	101	104	99	102	105	100	103	98	...
	$R_{\min}(n)$	5	2	7	4	1	6	3	0	5	2	7	4	...
	$\sum(n)$	625	617	625	617	617	625	617	617	625	617	625	617	...
516	$L_{\max}(n)$	105	101	105	101	105	101	105	101	105	101	105	101	...
	$R_{\min}(n)$	4	0	4	0	4	0	4	0	4	0	4	0	...
	$\sum(n)$	625	617	625	617	625	617	625	617	625	617	625	617	...
517	$L_{\max}(n)$	105	102	99	104	101	98	103	100	105	102	99	104	...
	$R_{\min}(n)$	3	6	1	4	7	2	5	0	3	6	1	4	...
	$\sum(n)$	625	625	617	625	625	617	625	617	625	625	617	625	...
518	$L_{\max}(n)$	105	103	101	99	105	103	101	99	105	103	101	99	...
	$R_{\min}(n)$	2	4	6	0	2	4	6	0	2	4	6	0	...
	$\sum(n)$	625	625	625	617	625	625	625	617	625	625	625	617	...
519	$L_{\max}(n)$	105	104	103	102	101	100	99	98	105	104	103	102	...
	$R_{\min}(n)$	1	2	3	4	5	6	7	0	1	2	3	4	...
	$\sum(n)$	625	625	625	625	625	625	625	617	625	625	625	625	...
520	$L_{\max}(n)$	105	105	105	105	105	105	105	105	105	105	105	105	...
	$R_{\min}(n)$	0	0	0	0	0	0	0	0	0	0	0	0	...
	$\sum(n)$	625	625	625	625	625	625	625	625	625	625	625	625	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Table 1: Example – lengths of extensions for different lengths of segments s. The depth of decomposition is $d = 3$ and the filter length is $m = 16$.

3.4. Corollaries and Limitations of the SegWT Algorithm

In [3] there can be found several practical corollaries for SegWT, e.g. that the segments cannot be shorter then 2^d.

From the description in the above sections it should be clear that the time lag of Algorithm 3.5 is one segment (i.e. s samples) plus the time needed for the computation of the coefficient from the current segment. In a special case when s is divisible by 2^d it holds even $R_{\min}(n) = 0$ for every $n \in \mathbb{N}$ (see Theorem 3.3), i.e. the lag is determined only by the computation time!

3.5. A Few Examples

- For $d = 4$ and $m = 12$, the minimum segment length is just 16 samples. When we set $s = 256$, $R_{\min}$ will always be zero and $L_{\max} = r(4) = 165$. The length of every extended segment will be $256 + 165 = 421$ samples.
- For $d = 5$ and $m = 8$, the minimum segment length is 32 samples. When we set $s = 256$, $R_{\min}$ will always be zero and $L_{\max} = r(5) = 217$. The length of every extended segment will be $256 + 217 = 473$ samples.
- For $d = 5$ and $m = 8$ we set $s = 300$, which is not divisible by 2^5. Thus $R_{\min}$ and $L_{\max}$ will alternate such that $0 \leq R_{\min} \leq 31$ and $186 \leq L_{\max} \leq 217$. The length of every extended segment will be $300 + r(5) = 473$ samples.

3.6. Implementation

The SegWT algorithm had been implemented in C++ and its functionality had been verified. We implemented a simple “band-stop filter” as a VST plug-in module utilizing the SegWT Algorithm and a reduced version of the inverse transform. The testing of the efficiency showed that the most demanding part of the algorithm is the computation of the convolution which must be done in each stage of the transform.

4. CONCLUSION

The paper contains a description of the algorithm which allows us to perform the wavelet transform in real time. The algorithm works on the basis of calculating the optimal extension (overlap) of signal segments, and subsequent performance of the modified transform.

In the future it would be convenient to improve the computational effectivity by reducing redundant computations at the borders of the segments, as it follows from the Algorithm 3.5. Also, it should not be very difficult to generalize the SegWT method to include biorthogonal wavelets and more general types of decimation [4, 5], because the parameters of SegWT can be chosen in a fairly general way.

Another important part of the future work is the derivation of an efficient counterpart to the introduced method – the segmented *inverse* transform. In fact, we made first experience, in which it turned out, above all, that the time lag in the consecutive forward-inverse processing will be, unfortunately, always nonzero.

5. ACKNOWLEDGEMENTS

The paper was prepared within the framework of No. 102/06/P407, No. 102/06/1233 and No. 102/07/1303 projects of the Grant Agency of the Czech Republic and No. 1ET301710509 project of the Czech Academy of Sciences.

6. REFERENCES

[1] D. Darlington, L. Daudet and M. Sandler, “Digital Audio Effects in the Wavelet Domain.” In *Proc. of the 5th Int. Conf. on Digital Audio Effects (DAFX-02)*, Hamburg (2002)

[2] G. Strang. and T. Nguyen, *Wavelets and Filter Banks*. Wellesley Cambridge Press (1996)

[3] P. Rajmic, *Exploitation of the wavelet transform and mathematical statistics for separation signals and noise, (in Czech)*, PhD Thesis, Brno University of Technology, Brno (2004)

[4] P. Dutilleux, “An implementation of the “algorithme à trous” to compute the wavelet transform.” In *Wavelets: Time-Frequency Methods and Phase Space*, Inverse Problems and Theoretical Imaging, editors J.-M. Combes, A. Grossman, P. Tchamitchian. pp. 298–304, Springer-Verlag, Berlin (1989)

[5] G.P. Nason and B.W. Silverman, The stationery wavelet transform and some statistical applications. In *Wavelets and Statistics*, volume 103 of *Lecture Notes in Statistics*, editors A. Antoniadis, G. Oppenheim, pp. 281–300, Springer-Verlag, New York (1995)