JOURNAL OF CONCRETE AND APPLICABLE MATHEMATICS, VOL.8, NO.3, 393-406, 2010, COPYRIGHT 2010 EUDOXUS PRESS, LLC

Algorithms for Segmentwise Computation of Forward and Inverse Discrete-time Wavelet Transform

Pavel Rajmic

Faculty of Electrical Engineering and Communications
Brno University of Technology
Purkyňova 118, 612 00, Brno, Czech Republic
E-mail: rajmic@feec.vutbr.cz

Abstract

The paper describes a method of segmented wavelet transform (SegWT) that makes it possible to compute the discrete-time wavelet transform of a signal segment-by-segment, with exactly the same result as if the whole signal were transformed at once. Due to its generality, the method can be utilized in many situations: for wavelet-type processing of a signal in real time or in case we want to process the signal in parallel or in case we need to process a long signal, but the available memory capacity is insufficient (e.g. in the DSPs). In the paper, the background theory and the emerging principles of both the forward and the inverse SegWT are explained.

Keywords segmented discrete-time wavelet transform, real-time wavelet transform, SegWT, signal processing, algorithm

1 Introduction

The discrete-time wavelet transform (DTWT) has many applications in the field of signal and image processing today. Most of them require signals to be completely known when the processing is initiated. The natural need for real-time applications originated in the development of methods allowing the computation of the DTWT without knowing the signal in advance, and possibly with minimum delay at the same time. Applications arising in the image processing field such as wavelet compression segment-by-segment (e.g. JPEG2000 coding for large images) lead to the same category of methods.

We are not interested in algorithms for transforming data received as a continuous stream [4] (e.g. in the FPGAs), because we suppose the data come in the form of nonoverlapping segments. Leaving these algorithms aside, methods

Figure 1: The undesirable artefacts near the segments borders. The image was (strongly) compressed using JPEG2000 algorithm, with tiling option switched on; the square tiles of 128×128 px in size — segments — are transformed separately.

for computing the wavelet transform in succession can be divided into two main classes.

1.1 Inexact Methods

This class includes another two types of methods:

The first-type methods are based on signal windowing and overlapping the resultant segments [2], analogous to the short-time Fourier analysis. However, windowing introduces (apart from potential considerable numerical errors at the window tails) severe problems when the wavelet coefficients are subject to nonlinear processing, e.g. the thresholding step within a denoising algorithm.

The second-type methods approach the particular signal segments independently — they "blindly" extrapolate samples beyond the boundaries of each segment. There exist several such methods, either simpler or more complex [7]. This approach, of course, leads to undesirable artifacts on the signal boundaries after the processing.

A typical example of the described inexactness can be seen in Fig. 1.

1.2 Exact Methods

This class, which we are most interested in, includes methods which actually extend the boundaries by samples from the respective neighbouring segments. Most of the contributions in these problems have come from researchers working with images and video. They were motivated by the idea of splitting the com-

putation into parallel processes run on individual workstations [3, 1]. Within this class of algorithms, we can distinguish another two algorithm types:

The third-type algorithms split the signal into segments which are distributed to the particular processes. During the whole computation the processes do not communicate with each other. Afterwards, the results are joined together. Such an approach is suitable for systems where interchanging information (i.e. wavelet coefficients) is slow and thus increases the total computation time. The disadvantage is that a portion of computation is performed redundantly, in several processes.

In the algorithms of the fourth type the processes mutually interchange data during computation. Clearly, this is suitable in situations where the communication is fast. The principal advantage here is that there is no computation redundancy. To be more concrete, wavelet coefficients located by the segment boundaries computed at each "level" of the transformation are interchanged.

State-of-the-art algorithms belonging to both the third and the fourth group, which can be found in the literature, are derived for the special case when each segment length is equal to a power of two. This assumption is their drawback, mainly for larger segments (e.g. the difference between 1024 and 2048 can be inadmissibly big, considering for example that with images, $1024^2 \doteq 10^6$ and $2048^2 \doteq 4 \cdot 10^6$). Also, there are situations where the segment sizes are not a power of two (e.g. the signal buffer size in audio cards running with ASIO driver could be 96 samples). Paper [5] gives directions for the non-power-of-two case. However, there is one more thing: all of the algorithms mentioned are made just for the purpose of forward DTWT, as they are mainly used for blockwise image compression. Moreover, most of the published methods specialize in JPEG2000, which means that they are restricted to the biorthogonal wavelet CDF 9/7.

1.3 Motivation and Goal of SegWT

The objective for the segmented wavelet transform, denoted SegWT, is naturally to allow signal processing by its segments, so that in this manner we get the same result (i.e. the same wavelet coefficients) as in the common DTWT case. At the same time, SegWT should utilize as much from the DTWT computational routines as possible.

In this paper we present the SegWT method, which can be utilized for any wavelet-type segmentwise data processing task, that is to say also in real time.

SegWT includes both the forward and the inverse parts of the transform. The segment length and the wavelet filter can be chosen arbitrarily. In fact, SegWT is of the third type in the sense of the above.

The derivation of the SegWT algorithm requires a very detailed knowledge of the behavior of ordinary DTWT, so before we start with SegWT, we recall the basic algorithm of DTWT.

2 Classical DTWT Algorithm

Algorithm 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* $\mathbf{h}$*,* J *is a positive integer denoting the decomposition depth. Also, the type of boundary treatment [7, ch. 8] has to be known.*

1. *Denote the input signal* $\mathbf{x}$ *by* $\mathbf{a}^{(0)}$ *and set* $j = 0$*.*

2. *One decomposition step:*

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

   (b) Filtering. *Convolve the extended signal with filter* $\mathbf{g}$*.*

   (c) Cropping. *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) Decimation. *Downsample the resultant vector.*

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

3. *Increase* j *by one. If it now holds* $j < J$*, return to item 2, in the other cases the algorithm ends.*

After Algorithm 1 has been finished, we have the wavelet coefficients stored in $J+1$ vectors (of different length) $\mathbf{a}^{(J)}, \mathbf{d}^{(J)}, \mathbf{d}^{(J-1)}, \ldots, \mathbf{d}^{(1)}$.

3 Method of Segmented Wavelet Transform

In the problem, the following parameters play a crucial role: $m \ldots$ wavelet filter length, $m > 0$, $J \ldots$ transform depth, $J > 0$, $s \ldots$ length of the segment, $s > 0$.

Based on a detailed knowledge of DTWT, it is possible to deduce fairly sophisticated rules how to handle the signal segments. It is clear that 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 (extension, overlap). However, the number of such samples depends on m, J and s, and it can be shown that every segment has to be extended by a different length from the left and from the right, and these lengths can also differ from segment to segment! And, of course, the first and the last segments have to be handled in a particular way.

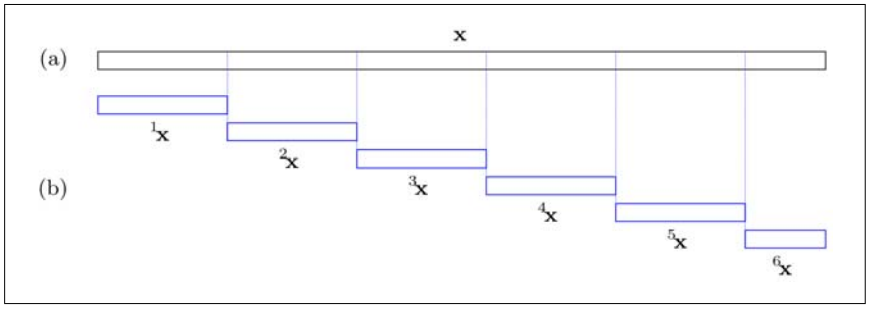

Figure 2: Scheme of signal segmentation. The input signal $\mathbf{x}$ (a) is divided into segments of equal length and the last one can be shorter than this (b).

3.1 Important Theorems Derived from DTWT Algorithm

Before we introduce a detailed description of the SegWT algorithm, several theorems must be presented. More theorems including proofs can be found in [6, 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 less long than s and the number S does not have to be known in advance. See Fig. 2. The signal boundary treatment considered in this paper is "zero-padding", when the boundaries are extended by zeros (most suitable for processing audio recordings, for example), but switching to another type of treatment is easy.

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 Figures 2 and 3, so that applying the DTWT of depth k, with step 2a) omitted (cf. Algorithm 3, page 9), separately to both the segments, let us say ${}^n\mathbf{x}$ and ${}^{n+1}\mathbf{x}$, and joining the resultant coefficients together leads to the situation that the last coefficient computed from ${}^n\mathbf{x}$ and the first coefficient computed from ${}^{n+1}\mathbf{x}$ would be neighboring in case the signal is transformed by the ordinary DTWT.

Such a situation is desirable and the theorems below lead to proper handling of the consecutive segments.

Theorem 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 J it is necessary to have $r(J) = (2^J - 1)(m - 1)$ common samples in the two consecutive segments after they have been extended. This extension must be divided into the right extension of the first segment (of length R) and the left extension of the following segment (of

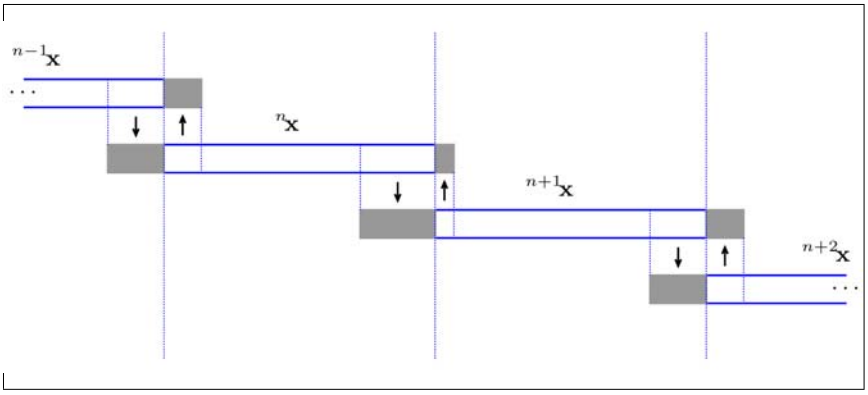

Figure 3: Illustration of extending the segments.

length L), while $r(J) = R + L$. However, the lengths $L, R \geq 0$ cannot be chosen arbitrarily. In general, the numbers L and R are not uniquely determined and must comply with strict rules that will be shown. The formula for the choice of extension $L_{\max}$, which is unique and the most appropriate in the case of real-time signal processing, is given in Theorem 2.

For the purpose of the following, we assign the number of the respective segment to the variables $L_{\max}, R_{\min}, l$, so that 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 $l(n)$.

Theorem 2 *Let the n-th segment be given, whose length including its left extension is $l(n)$. The maximum possible left extension of the next segment, $L_{\max}(n+1)$, can be computed by the formula*

$$L_{\max}(n+1) = l(n) - 2^J \ \text{ceil}\left(\frac{l(n) - r(J)}{2^J}\right). \tag{2}$$

The minimum possible right extension of the given segment is then

$$R_{\min}(n) = r(J) - L_{\max}(n+1). \tag{3}$$

Theorem 3 *The length of the right extension of the n-th segment, must comply with*

$$R_{\min}(n) = 2^J \ \text{ceil}\left(\frac{ns}{2^J}\right) - ns, \qquad n = 1, 2, \ldots, S-2. \tag{4}$$

From (4) it is clear that $R_{\min}$ is periodic with respect to s with period 2^J, i.e. $R_{\min}(n + 2^J) = R_{\min}(n)$. This property, among other things, can be seen in Table I.

Theorem 4 *(on the total length of segment) After the extension, the n-th segment of original length s will be of total length* $\sum(n)$*, which can acquire one of two values, either*

$$r(J) + 2^J \operatorname{ceil}\left(\frac{s}{2^J}\right) \quad \text{or} \quad r(J) + 2^J \operatorname{ceil}\left(\frac{s}{2^J}\right) - 2^J. \tag{5}$$

The overall illustration of SegWT can be seen in Fig. 4. The particular algorithms are described in detail in the next sections.

3.2 Algorithm of Forward SegWT

The algorithm works such that it reads (receives) individual segments of the input signal, makes them extend each other in a proper way, then it computes the wavelet coefficients in a modified way and, in the end, it easily joins the coefficients. There is no need to know how many segments will be in total, we only require that in the moment when the last segment is received, we know that information.

Algorithm 2 *Let the wavelet filters* $\mathbf{g}$ *and* $\mathbf{h}$ *be of length m and the decomposition depth be J. The boundary treatment mode is "zero-padding". The segments of length* $s > 0$ *of the input signal* $\mathbf{x}$ *are denoted* ${}^1\mathbf{x}, {}^2\mathbf{x}, {}^3\mathbf{x}, \ldots$ *The last segment contains* $s' \leq s$ *samples.*

1. *Set* $n = 1, last = 0$.
2. *Read the first segment,* ${}^1\mathbf{x}$*. Extend it from the left by* $r(J)$ *zero samples. Update 'last'.*
3. ***If**, at the same time, the n-th segment is the last one*
   (a) *Extend the n-th segment from the right by such a number of zero samples that its total length will be* $L_{\max}(n) + s$.
   (b) *Extend the n-th segment from the right by* $r(J)$ *zero samples.*
   (c) *Compute the transform of depth J of the extended segment using Algorithm 3.*
   (d) *Modify the vectors containing the wavelet coefficients by trimming off a certain number of redundant coefficients from the left side, specifically: on the k-th level,* $k = 1, 2, \ldots, J-1$*, trim off* $r(J-k)$ *coefficients.*
   (e) *Trim off redundant coefficients from the right so that on the k-th level* $\operatorname{floor}\left(2^{-k}(L_{\max}(S) + s')\right)$ *coefficients remain.*
   (f) *Trim off the vectors in the same manner as in 3d, but this time from the right.*
   (g) *Store the result as* ${}^n\mathbf{a}^{(J)}, {}^n\mathbf{d}^{(J)}, {}^n\mathbf{d}^{(J-1)}, \ldots, {}^n\mathbf{d}^{(1)}$.

   Otherwise

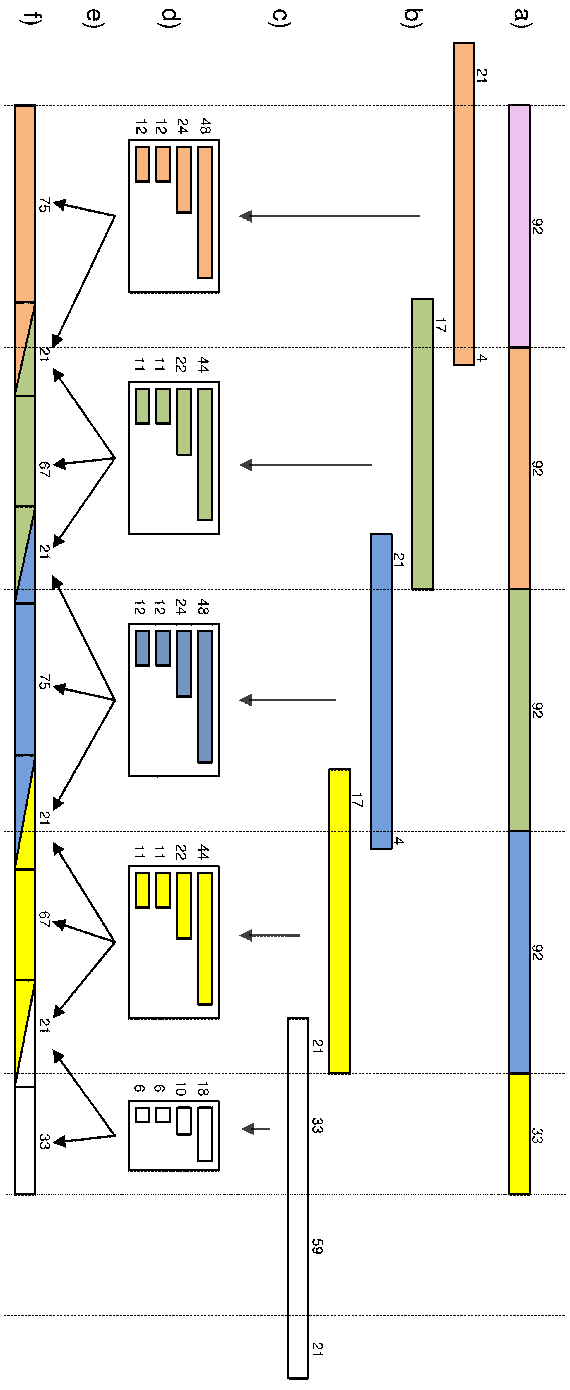

Figure 4: Overall scheme of the SegWT, forward and inverse parts, here in a particular case when $s = 92$: a) input signal segments, b) extending them (the left and right lengths differ from segment to segment), c) computation of the forward part, d) the computed blocks of coefficients, e) computation of the inverse part, f) the reconstructed signal.

(h) Read the $(n+1)$-th segment, update 'last'.

(i) Compute $L_{\max}(n+1)$ and $R_{\min}(n)$ (see Theorem 2).

(j) Extend the n-th segment from the right side:
***If** last $= 1$ (i.e. we have the last segment)*

i. Compute the difference diff $= \max(0, R_{\min}(n) - s')$.

*ii. **If** diff > 0 (i.e. not enough samples in the last segment for extension by $R_{\min}(n)$)*

A. Extend the n-th segment from the right side by s' samples from the last segment.

B. Extend the n-th segment from the right side by another diff zero samples.

Otherwise

C. Extend the n-th segment from the right side by $R_{\min}$ samples taken from the last segment.

Otherwise

iii. Extend the n-th segment from the right side by $R_{\min}$ samples taken from the last segment.

(k) Extend the $(n+1)$-th segment from the left side by $L_{\max}(n+1)$ samples taken from segment n.

(l) Compute the DTWT of depth J from the (extended) n-th segment using Algorithm 3.

(m) Modify the particular vectors containing the coefficients in the same manner as in 3d.

(n) Store the result as ${}^n\mathbf{a}^{(J)}, {}^n\mathbf{d}^{(J)}, {}^n\mathbf{d}^{(J-1)}, \ldots, {}^n\mathbf{d}^{(1)}$.

(o) Increase n by 1 and go to item 3.

Algorithm 3 *This sub-algorithm is identical to Algorithm 1 with the exception that we omit step 2a), i.e. we do not extend the vector.*

The output of Algorithm 2 is $S(J+1)$ vectors of wavelet coefficients

$$\{ {}^i\mathbf{a}^{(J)}, {}^i\mathbf{d}^{(J)}, {}^i\mathbf{d}^{(J-1)}, \ldots, {}^i\mathbf{d}^{(1)} \}_{i=1}^{S}. \tag{6}$$

If we simply join these vectors together, we obtain a set of $J+1$ vectors, which are identical to the wavelet coefficients of signal $\mathbf{x}$.

The flowchart of Algorithm 2 is in Fig. 5.

3.3 Corollaries and Limitations of SegWT Algorithm

In [6] several practical corollaries for SegWT can be found, e.g. that the segments cannot be shorter than 2^J. From the description in the above sections it should be clear that the delay of Algorithm 2 is one segment (i.e. s samples) plus the time needed for the computation of the coefficient from the current segment. It can be easily shown that in the special case of s being divisible by 2^J it even holds $R_{\min}(n) = 0$ for every $n \in \mathbb{N}$ (see Theorem 3), i.e. the delay of the forward method is determined only by the computation time!

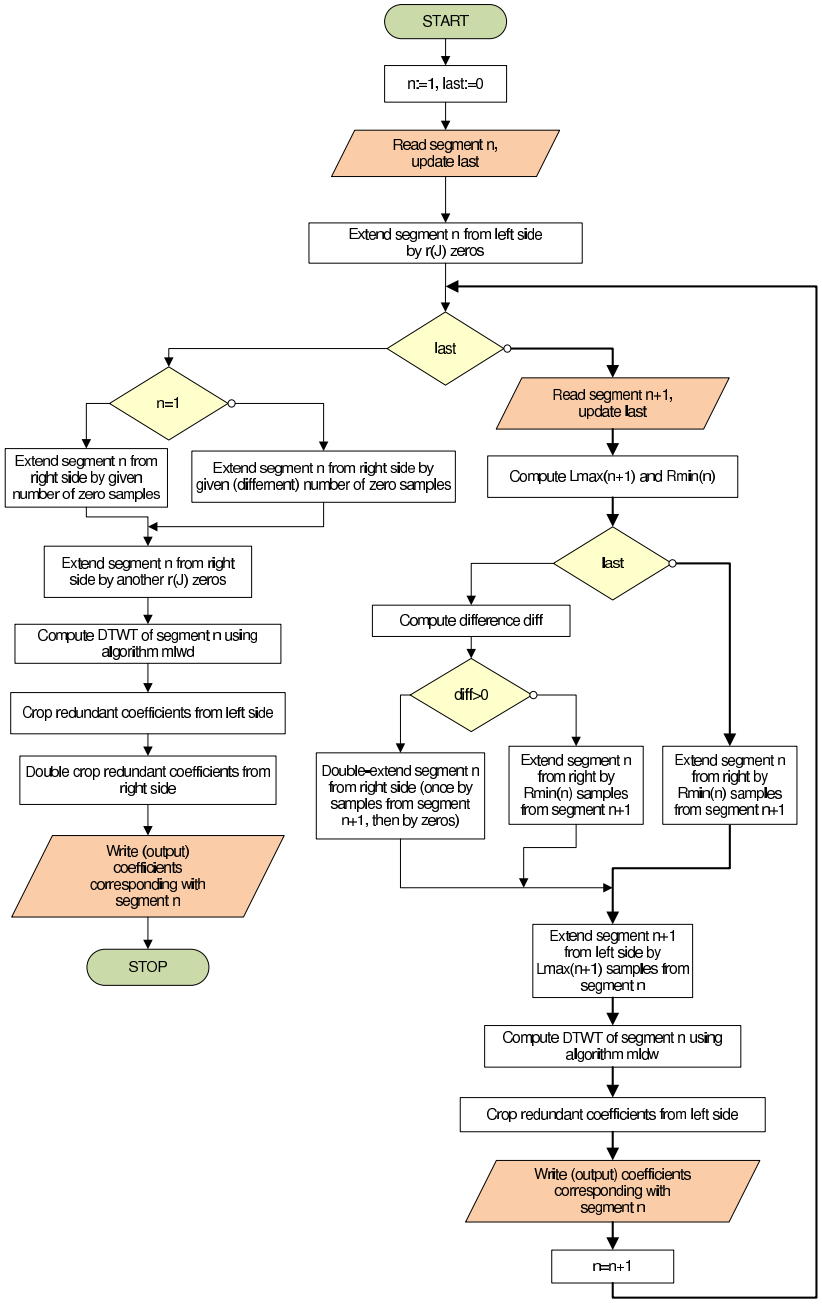

Figure 5: Flowchart of the forward SegWT, with zero-padding treatment of the signal boundaries. The main loop, which is applied to all the segment but the first and last ones, is emphasized by the thicker line.

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

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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

3.4 Few Examples

- For $J = 5$ 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 $J = 5$ and $m = 8$ we set $s = 300$, which is not divisible by 2^5. Thus $R_{\min}$ and $L_{\max}$ will alternate with period 8 such that $0 \leq R_{\min} \leq 31$ and $186 \leq L_{\max} \leq 217$. The total length of segment after extension will be either 505 or 537.

- (Example illustrated in Fig. 4) For $J = 3$, $m = 4$, $s = 92$, the extensions will alternate between two states, either $R_{\min} = 4$ and $L_{\max} = 17$ or $R_{\min} = 0$ and $L_{\max} = 21$. The length of the extended segments will be 109 or 117 samples.

The increase of the samples entering the computation is naturally a price paid for the fact that no errors will originate during processing the boundaries.

3.5 Algorithm of Inverse SegWT

The inverse algorithm is described below, in less detail than the forward one. Blocks of wavelet coefficients (6) produced segment-by-segment by the forward SegWT constitute the input for the inverse algorithm. Analog to the forward case, we use the boolean flag *last*, which becomes true if the very last segment has to be processed.

In addition to that, due to the downsampling step of the forward transform, we loss information about the total length of the signal, more precisely we do not know if the original length was a or $a + 1$ for some integer a. We could solve this problem by accumulating the lengths of individual inverted

segments, however, such a number could be very large, possibly overflowing the processor arithmetics. A better solution is just to keep the signal parity (i.e. if the accumulated length is is even or odd). The information is then used at the very end of the signal for deciding to cut or not to cut the last reconstructed sample.

The inverse SegWT partly utilizes the overlap-add principle for joining the reconstructed pieces of the time-domain signal. The length of the overlap stays $r(J)$ all the time. As for the illustration, we again refer to Fig. 4.

Algorithm 4 *Let the decomposition depth J be given, as well as wavelet reconstruction filters $\tilde{\mathbf{g}}$ and $\tilde{\mathbf{h}}$ of length m, and coefficients ${}^n\mathbf{a}^{(J)}, {}^n\mathbf{d}^{(J)}, {}^n\mathbf{d}^{(J-1)}, \ldots, {}^n\mathbf{d}^{(1)}$ for all n.*

1. *Set $n = 1$. Set last $= 0$.*
2. ***If** last $= 1$ then the Algorithm ends.*
3. *Read the n-th block of coefficients and update 'last'.*
4. *Extend the detail coefficients: on the k-th level, $k = 1, \ldots, J-1$, append $r(J-k)$ zero coefficients from the left side.*
5. *Compute the inverse transform of depth J using Algorithm 5.*
6. ***If** $n \neq 1$, recall the samples for the overlap, saved in the last cycle, and add them to the current inverted block.*
7. *Update the parity of the signal.*
8. ***If** last $\neq 1$, append the central, non-overlapping part to the output. Save the samples of the overlap of the current inverted segment for the next cycle.*
   ***Otherwise** Append the whole inversion to the output. Eventually crop several samples from the end of the signal.*
9. *The output (a segment of a time-domain signal) is now complete and prepared to be "sent".*
10. *Increase n by 1 and return to item 2.*

Algorithm 5 *This algorithm is identical to the ordinary inverse wavelet transform (i.e. upsampling – filtering – summing – cropping), but the cropping phase is omitted here.*

The flowchart of Algorithm 4 can bee seen in Fig. 6.

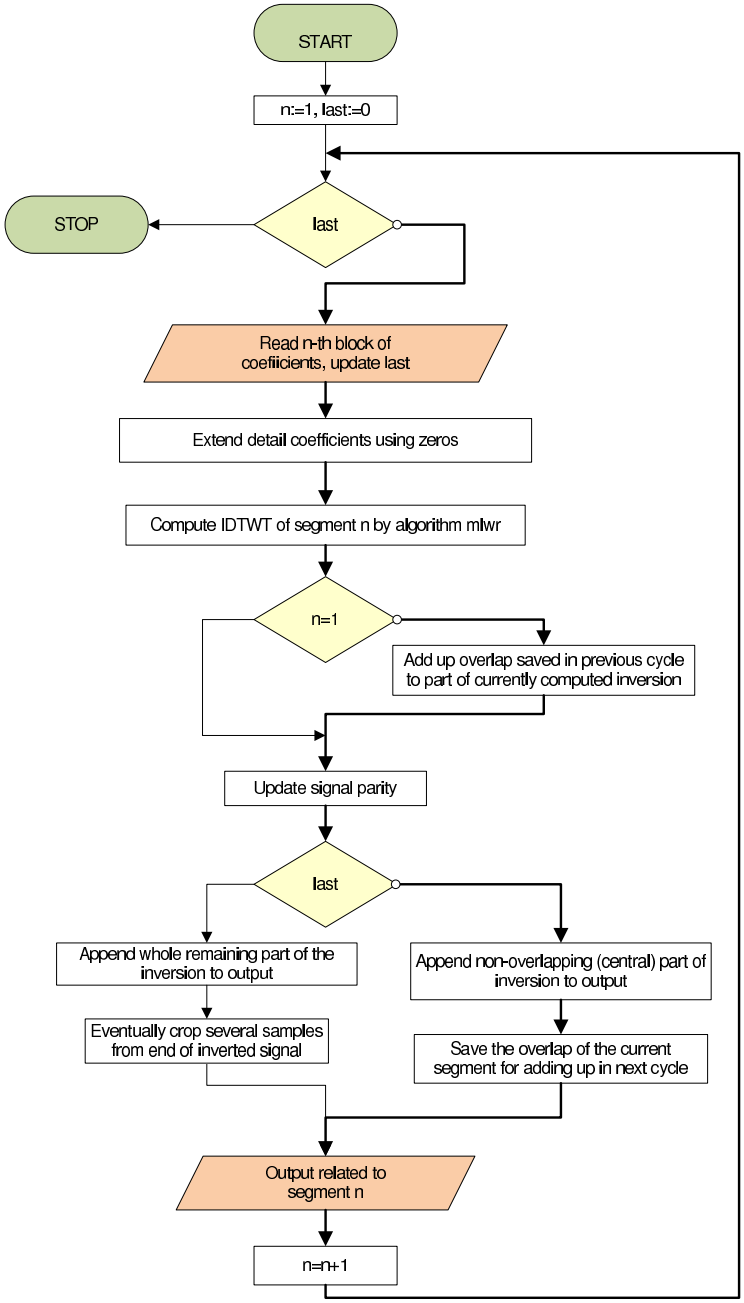

Figure 6: Flowchart of the inverse SegWT. The main loop is emphasized by the thicker line.

3.6 Joining Forward and Inverse Parts to Form Algorithm Capable of Real-Time Performance

The algorithms in Sec. 3.2 and 3.5 were presented separately for clarity. However, their easy integration into a simple joint loop forms a universal algorithm for any wavelet-type processing task in real time. It can be shown that in the case of s being divisible by 2^J the total delay is no bigger than s samples, in other cases no bigger than $2s$.

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

References

[1] Ch. Chrisafis and A. Ortega, Line-Based, Reduced Memory, Wavelet Image Compression. *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 378–389, 2000.

[2] 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.

[3] W. Jiang and A. Ortega, Lifting factorization-based discrete wavelet transform architecture design. *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 11, no. 5, pp. 651–657, 2001.

[4] Hd.O. Mota and F.H. Vasconcelos and R.M. Silva, Real-time wavelet transform algorithms for the processing of continuous streams of data, *Intelligent Signal Processing, 2005 IEEE International Workshop on,* 1–3 Sept. 2005, pp. 346–351. ISBN 0-7803-9030-x

[5] J.H. Nealand and A.B. Bradley and M. Lech, Overlap-Save Convolution Applied to Wavelet Analysis. *IEEE Signal Processing Letters*, vol. 10, no. 2, pp. 47–49, 2003.

[6] 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.

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