BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

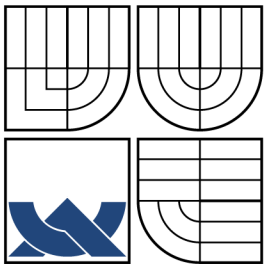
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

SEGMENTWISE DISCRETE WAVELET TRANSFORM

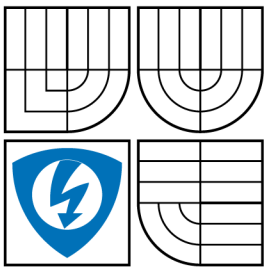
DOCTORAL THESIS
DIZERTAČNI PRÁCE

AUTHOR
AUTOR PRÁCE

Ing. ZDENĚK PRŮŠA



BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

SEGMENTWISE DISCRETE WAVELET TRANSFORM
SEGMENTOVANÁ DISKRÉTNÍ VLNKOVÁ TRANSFORMACE

DOCTORAL THESIS
DIZERTAČNI PRÁCE

AUTHOR
AUTOR PRÁCE
Ing. ZDENĚK PRŮŠA

SUPERVISOR
VEDOUCÍ PRÁCE
Mgr. PAVEL RAJMIC, Ph.D.

BRNO 2012

ABSTRACT

The dissertation deals with SegDWT algorithms performing a segmented (segmentwise) computation of one- and multi-dimensional Discrete Wavelet Transform – DWT. The segmented approach allows one to perform the segment (block) wavelet analysis and synthesis using segment overlaps while preventing blocking artifacts. The parts of the wavelet coefficients of the whole signal wavelet transform corresponding to the actual segment are produced by the analysis part of the algorithm exploiting overlap-save principle. The resulting coefficients belonging to the segment can be processed arbitrarily and than they can transformed back to the original domain. The reconstructed segments are than put together using overlap add principle.

The already known SegDWT algorithm can not be effectively used on multidimensional signals. Several modifications of the algorithm are proposed which makes it possible to generalize it to multidimensional cases using separability property. In addition, the thesis presents SegLWT algorithm adopting ideas of the SegDWT and transferring it to the non-causal lifting filter bank structures.

KEYWORDS

discrete wavelet transform, lifting scheme, real-time, SegDWT, parallelization, overlap-add, overlap-save

ABSTRAKT

Dizertační práce se zabývá algoritmy SegDWT pro segmentový výpočet Diskrétní Waveletové Transformace – DWT jedno i vícedimenzionálních dat. Segmentovým výpočtem se rozumí způsob výpočtu waveletové analýzy a syntézy po nezávislých segmentech (blocích) s určitým překryvem tak, že nevznikají blokové artefakty. Analyzující část algoritmu pracuje na principu odstranění přesahu a produkuje vždy část waveletových koeficientů z waveletové transformace celého signálu, které mohou být následně libovolně zpracovány a podrobeny zpětné transformaci. Rekonstruované segmenty jsou pak skládány podle principu přičtení přesahu.

Algoritmus SegDWT, ze kterého tato práce vychází, není v současné podobně přímo použitelný pro vícerozměrné signály. Tato práce obsahuje několik jeho modifikací a následné zobecnění pro vícerozměrné signály pomocí principu separability. Kromě toho je v práci představen algoritmus SegLWT, který myšlenku SegDWT přenáší na výpočet waveletové transformace pomocí nekauzálních struktur filtrů typu lifting.

KLÍČOVÁ SLOVA

diskrétní waveletová transformace, lifting schéma, reálný čas, SegDWT, paralelizace, metoda přičtení přesahu, metoda odstranění přesahu

PRŮŠA, Zdeněk *Segmentwise Discrete Wavelet Transform*: doctoral thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2012. 105 p. Supervised by Mgr. Pavel Rajmic, Ph.D.

DECLARATION

I declare that I have written my doctoral thesis on the theme of "Segmentwise Discrete Wavelet Transform" independently, under the guidance of the doctoral thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the doctoral thesis I furthermore declare that, as regards the creation of this doctoral thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

(author's signature)

ACKNOWLEDGEMENT

I am deeply grateful to my supervisor Mg. Pavel Rajmic, Ph.D. for his guidance and support. The original SegDWT algorithm is his brainchild and I feel honored that I could participate in its further development.

I would like to express gratitude to my parents, to Markéta and to others for their patience, understanding and their infinite support.

Brno ....

(author's signature)

CONTENTS

LIST OF FIGURES

INTRODUCTION

The discrete wavelet transform (DWT) has been extensively studied over the recent decades. Many applications have been proposed but the true power of the wavelet transform lies in its performance in compression and denoising schemes. The existence of fast algorithms for its computation is another important factor. The well known Mallat's algorithm employs a perfect reconstruction two-channel filter bank iteratively and the filter bank can be equally represented by a polyphase lifting scheme. The iterative application of the lifting scheme (LWT – Lifting Wavelet Transform) results in the same coefficients as the DWT does.

The present thesis deals with the problem of computing the one- and multi-dimensional wavelet transform segmentwise. Often, it is impractical or even impossible to load the whole input signal at once. When using common border extension methods (e.g. zero-padding, periodization, symmetrical extension) the wavelet analysis results in "false" coefficients, which, in turn, result in distortion at borders of segments after the synthesis, provided the wavelet coefficients were modified (e.g. thresholded). The thesis presents an algorithm which circumvents the described border artifacts by employing segment overlaps whose lengths are derived from the actual discrete wavelet transform setup.

The idea of the algorithm (Segmentwise DWT – SegDWT) for one-dimensional signals was originated by Mgr. Pavel Rajmic, Ph.D. in his dissertation *Utilization of Wavelet Transform and Mathematical Statistics for Separating Signal from Noise* [10]. The present thesis builds upon the algorithm and extends it in several ways as you can read in the summary in the chapter 3.

The thesis is organized as follows. Chapter 1 contains a brief introduction to the wavelet transform theory on sequences and finite-length signals and highlights areas which are treated in greater detail for they are used later in the thesis. These areas are Mallat's algorithm, noble multirate identity, lifting scheme and extension of the wavelet transform to multidimensional signals.

The next chapter 2 discusses other approaches to segmentwise computations of the wavelet transform found in the literature and the main part of the chapter is devoted to description of the original SegDWT algorithm and its parts.

Chapter 3 describes the main drawbacks of the original algorithm and states objectives of the thesis.

Starting from the chapter 4 the presented ideas are solely an original contribution of the author of the thesis. Chapter 4 is devoted to modifications of the original algorithm which is not directly usable for multidimensional signals. The new possible application of the algorithm rises from the presented modifications viz. Region of Interest – ROI wavelet coefficients processing.

The next chapter, chapter 5, adapts the ideas of the segmentwise computation to the lifting wavelet transform. The process of finding the correct overlaps is fundamentally different and more complex. Also the non-causality of a lifting scheme makes it difficult to cope with the extensions.

Chapter 6 contains a generalization of the SegDWT algorithm to multiple dimensions (images, videos, MRI images/videos).

Several programs were created as "proof-of-concept" of the proposed algorithms. Therefore, chapter 7 describes VST plugin for a real-time wavelet audio processing. Another application described in chapter 7 is the parallel wavelet transform computation of large images.

The thesis is concluded by an evaluation of the presented contributions and by stating directions for further research.

1 THEORETICAL BACKGROUND

Despite the relatively short time of its existence, the wavelet transform (WT) established itself as a standard tool for digital signal processing. The brisk evolution of the WT was driven by the need of a tool which would provide more effective representations of signals than the already known ones. Usually, the different kinds of representations were compared witch each other using *sparsity* or *compressibility* property (number of nonzero or important coefficients) i.e. signal recovery accuracy using just a minor number of the representation coefficients. Naturally, the property of the WT resulted in its usage in many compression schemes for images e.g. EZW [11], SPIHT [12] and its modifications, EBCOT [13] in JPEG2000 standard and others and their extensions for videos and more dimensional signals.

The properties of the WT also allow an effective denoising [14] which can be also found in many modifications.

In addition, the WT was successfully used in areas like image watermarking [15] or computer vision [16]. Additional uncommon image operations in the wavelet domain were presented in [17].

Preliminaries This paragraph establishes common mathematical notation held in the thesis. Since the reader is expected to be familiar with the basic concept of the continuous WT (scale function, MRA, dilatation equations, there are many introductory books and publications, to name a few: [18–20]), the theoretical background given in this chapter is limited to the discrete setting only. At first, signals $\boldsymbol{x}$ will be considered to be possibly infinite but finite-energy sequences belonging to Hilbert space $\ell^2(\mathbb{Z})$ with a scalar product induced norm, later, a transition to finite-length discrete signals (vectors) from Euclidean space $\mathbb{C}^N$ will be made. Moreover, only MRA compact support wavelets are considered allowing usage of the fast Mallat's algorithm for computing wavelet coefficients using FIR filter banks.

The signals will be denoted as vectors $\boldsymbol{x}$, their nth element will be denoted as $x[n]$ and the subset of element as $x[n]_{n\in\mathcal{I}}$, where $\mathcal{I}$ denotes an indexing set. Whenever the finite-length signal indexing is used, the zero index denotes the foremost sample. Moreover, throughout the text J denotes the depth of the wavelet decomposition and m denotes the length of the wavelet filters. The list of used symbols is summarized at page 88.

The notion of the *odd* and the *even* downsampling (decimation) and upsampling (interpolation) will become important when dealing with finite-length signals. Regarding the downsampling, the factor N even downsampling repeats two steps starting with the zero index sample: remove $N-1$ samples and leave the N sample, whereas the odd downsampling does the steps in a reverse order: leave a sample

and than remove $N-1$ samples. Similarly, the factor N upsampling adds $N-1$ zero samples "between every two samples". The even upsampling adds $N-1$ zeros at beginning and at the end of the signal whereas odd upsampling does not. If it is not said otherwise, the even type will be considered throughout the text.

1.1 Wavelet Expansions on Sequences

The continuous WT theory concludes [21] that any practically interesting MRA compact support dyadic wavelets have a characteristic finite-length dilatation coefficient vectors $\boldsymbol{h}_{\mathrm{mr}}$, $\boldsymbol{g}_{\mathrm{mr}}$ associated with them. In the discrete setting, using dilatation coefficients and given number of scales $J > 0$, one can build basis for MRA nested subspaces $\mathcal{V}^{(j)}$ and their orthogonal complements $\mathcal{W}^{(j)}$. In the **orthogonal** wavelet case, at each j-th scale (level of decomposition), there is such set of sequences $\left\{\boldsymbol{\varphi}_p^{(j)}\right\}_{p\in\mathbb{Z}}$ which form orthogonal basis for subspace $\mathcal{V}^{(j)}$ and for given j the sequences are shifted versions of the original one $\boldsymbol{\varphi}_0^{(j)}$ such as

$$\varphi_p^{(j)}[k] = \varphi_0^{(j)}\left[k - p2^j\right]_{k\in\mathbb{Z}} \tag{1.1}$$

where $\boldsymbol{\varphi}_0^{(j)}$ is constructed using scale dilatation coefficients for $j \geq 1$

$$\boldsymbol{\varphi}_0^{(j)} = \sum_k h_{\mathrm{mr}}[k]\boldsymbol{\varphi}_k^{(j-1)}, \tag{1.2}$$

and arbitrary $\boldsymbol{\varphi}_p^{(j)}$ as

$$\boldsymbol{\varphi}_p^{(j)} = \sum_k h_{\mathrm{mr}}[k]\boldsymbol{\varphi}_{k+2p}^{(j-1)} \quad \text{or} \quad \boldsymbol{\varphi}_p^{(j)} = \sum_k h_{\mathrm{mr}}[k-2p]\boldsymbol{\varphi}_k^{(j-1)}. \tag{1.3}$$

Similarly, at each scale j, there is a set of sequences $\left\{\boldsymbol{\psi}_p^{(j)}\right\}_{p\in\mathbb{Z}}$ which forms an orthogonal basis for subspace $\mathcal{W}^{(j)}$

$$\psi_p^{(j)}[k] = \psi_0^{(j)}\left[k - p2^j\right]_{k\in\mathbb{Z}} \tag{1.4}$$

and using wavelet dilatation coefficients

$$\boldsymbol{\psi}_0^{(j)} = \sum_k g_{\mathrm{mr}}[k]\boldsymbol{\varphi}_k^{(j-1)}. \tag{1.5}$$

$$\boldsymbol{\psi}_p^{(j)} = \sum_k g_{\mathrm{mr}}[k]\boldsymbol{\varphi}_{k+2p}^{(j-1)} \quad \text{or} \quad \boldsymbol{\psi}_p^{(j)} = \sum_k g_{\mathrm{mr}}[k-2p]\boldsymbol{\varphi}_k^{(j-1)}. \tag{1.6}$$

The nested subspaces $\mathcal{V}^{(j)}$ are organized as follows:

$$\mathcal{V}^{(J)} \subset \ldots \subset \mathcal{V}^{(2)} \subset \mathcal{V}^{(1)} \subset \mathcal{V}^{(0)}, \tag{1.7}$$

where $\mathcal{V}^{(0)} = \ell^2(\mathbb{Z})$ and $\left\{\boldsymbol{\varphi}_p^{(0)}\right\} = \delta[n-p]_{p\in\mathbb{Z}}$ being Dirac train and for $j \geq 1$ the $\mathcal{V}^{(j-1)} = \mathcal{V}^{(j)} \bigoplus^{\perp} \mathcal{W}^{(j)}$ and $\mathcal{W}^{(j)} \cap \mathcal{W}^{(i)} = \emptyset$ for $j \neq i$ holds. Consequently the union of subspaces $\mathcal{V}^{(J)}$ and $\mathcal{W}^{(j)}$ for $j = 1, \ldots, J$ covers whole $\ell^2(\mathbb{Z})$ space

$$\ell^2(\mathbb{Z}) = \underbrace{\underbrace{\mathcal{V}^{(J)} \cup \mathcal{W}^{(J)}}_{\mathcal{V}^{(J-1)}} \cup \mathcal{W}^{(J-1)}}_{\mathcal{V}^{(J-2)}} \cup \ldots \cup \mathcal{W}^{(1)}. \tag{1.8}$$

Denoting $a^{(J)}[p]$ as approximation wavelet coefficients at level J and $d^{(j)}[p]$ as detail wavelet coefficients at the level j , the wavelet expansion of the input signal $\boldsymbol{x}$ can be written as

$$\boldsymbol{x} = a^{(J)}[p]\,\boldsymbol{\varphi}_p^{(J)} + \sum_{j=1}^{J} d^{(j)}[p]\,\boldsymbol{\psi}_p^{(j)}. \tag{1.9}$$

As $\left\{\boldsymbol{\varphi}_p^{(J)}\right\}$ and $\left\{\boldsymbol{\psi}_p^{(j)}\right\}_{j=1,\ldots,J}$ are orthogonal basis vectors for $\ell^2(\mathbb{Z})$, the wavelet coefficients of the input signal $\boldsymbol{x}$ are given by scalar products

$$a^{(J)}[p] = \left\langle \boldsymbol{x}, \boldsymbol{\varphi}_p^{(J)} \right\rangle, \qquad d^{(j)}[p] = \left\langle \boldsymbol{x}, \boldsymbol{\psi}_p^{(j)} \right\rangle, \tag{1.10}$$

Using (1.3),(1.6) and $\boldsymbol{a}^{(0)} = \boldsymbol{x}$, the $\boldsymbol{a}^{(j)}$ and $\boldsymbol{d}^{(j)}$ for $j \geq 1$ can be written as

$$a^{(j)}[p] = \sum_{n\in\mathbb{Z}} h_{\mathrm{mr}}[n-2p] a^{(j-1)}[n], \quad d^{(j)}[p] = \sum_{n\in\mathbb{Z}} g_{\mathrm{mr}}[n-2p] a^{(j-1)}[n]. \tag{1.11}$$

and similarly in the reverse direction for $j = J, \ldots, 1$

$$a^{(j-1)}[p] = \sum_{n\in\mathbb{Z}} h_{\mathrm{mr}}[p-2n] a^{(j)}[n] + \sum_{n\in\mathbb{Z}} g_{\mathrm{mr}}[p-2n] d^{(j)}[n] \tag{1.12}$$

It is well known, that the equations (1.11) can be rewritten in a form of a convolution followed by the downsampling (see fig. 1.1) and for the given J form iterated two-channel filter bank. This way of computing the wavelet coefficients is referred to as *fast (discrete) wavelet transform* or *Mallat's algorithm* [20].

In the **biorthogonal wavelet** case, there are two sets of dilatation coefficient vectors $\boldsymbol{h}_{\mathrm{mr}}, \boldsymbol{g}_{\mathrm{mr}}$ and $\widetilde{\boldsymbol{h}}_{\mathrm{mr}}, \widetilde{\boldsymbol{g}}_{\mathrm{mr}}$ and also two sets of hierarchical subspaces $\mathcal{V}^{(j)}$ and $\widetilde{\mathcal{V}}^{(j)}$

$$\mathcal{V}^{(J)} \ldots \subset \mathcal{V}^{(2)} \subset \mathcal{V}^{(1)} \subset \mathcal{V}^{(0)} \qquad \text{and} \qquad \widetilde{\mathcal{V}}^{(J)} \ldots \subset \widetilde{\mathcal{V}}^{(2)} \subset \widetilde{\mathcal{V}}^{(1)} \subset \widetilde{\mathcal{V}}^{(0)}, \tag{1.13}$$

having basis vectors $\left\{\boldsymbol{\varphi}_p^{(j)}\right\}$, $\left\{\widetilde{\boldsymbol{\varphi}}_p^{(j)}\right\}$ respectively with $\mathcal{V}^{(0)} = \widetilde{\mathcal{V}}^{(0)} = \ell^2(\mathbb{Z})$ and $\left\{\boldsymbol{\varphi}_p^{(0)}\right\} = \left\{\widetilde{\boldsymbol{\varphi}}_p^{(0)}\right\} = \delta[n-p]_{p\in\mathbb{Z}}$. For the given j, the bases are *dual* to each other, which means that projecting vector onto one base gives coordinates in the second one and vice versa. Similarly, there are complementary spaces $\mathcal{W}^{(j)}$ and $\widetilde{\mathcal{W}}^{(j)}$ with basis sequences $\left\{\boldsymbol{\psi}_p^{(j)}\right\}$, $\left\{\widetilde{\boldsymbol{\psi}}_p^{(j)}\right\}$ respectively. The orthogonal complements are

$\mathcal{V}^{(j-1)} = \widetilde{\mathcal{V}}^{(j)} \oplus^{\perp} \mathcal{W}^{(j)}$ and $\widetilde{\mathcal{V}}^{(j-1)} = \mathcal{V}^{(j)} \oplus^{\perp} \widetilde{\mathcal{W}}^{(j)}$. Consequently, the union of subspaces $\mathcal{V}^{(J)}$ and $\mathcal{W}^{(j)}$ and union of subspaces $\widetilde{\mathcal{V}}^{(J)}$ and $\widetilde{\mathcal{W}}^{(j)}$ for $j = 1, \ldots, J$ forms MRA biorthogonal bases for the $\ell^2(\mathbb{Z})$ space

$$\ell^2(\mathbb{Z}) = \underbrace{\underbrace{\mathcal{V}^{(J)} \cup \mathcal{W}^{(J)}}_{\mathcal{V}^{(J-1)}} \cup \mathcal{W}^{(J-1)}}_{\mathcal{V}^{(J-2)}} \cup \ldots \cup \mathcal{W}^{(1)}, \tag{1.14}$$

$$\ell^2(\mathbb{Z}) = \underbrace{\underbrace{\widetilde{\mathcal{V}}^{(J)} \cup \widetilde{\mathcal{W}}^{(J)}}_{\widetilde{\mathcal{V}}^{(J-1)}} \cup \widetilde{\mathcal{W}}^{(J-1)}}_{\widetilde{\mathcal{V}}^{(J-2)}} \cup \ldots \cup \widetilde{\mathcal{W}}^{(1)} \tag{1.15}$$

As in the orthogonal case, the wavelet expansion is given by (1.9) but the wavelet coefficients are calculated differently as projections onto the dual bases

$$a^{(J)}[p] = \left\langle \boldsymbol{x}, \widetilde{\boldsymbol{\varphi}}_p^{(J)} \right\rangle, \quad d_j[p] = \left\langle \boldsymbol{x}, \widetilde{\boldsymbol{\psi}}_p^{(j)} \right\rangle. \tag{1.16}$$

Again, the projections and expansion can be calculated iteratively using fast Mallat's algorithm.

The two channel filter bank view of wavelet transform allowed new approaches to wavelet transform design in a form of an orthogonal or biorthogonal filter bank solutions satisfying a perfect reconstruction criterion [22, 23]. The approaches will not be discussed further, just the main results concerning filter's support and length.

The analyzing low-pass filter is denoted by $\boldsymbol{h}$, the high-pass filter by $\boldsymbol{g}$, the reconstructing low-pass filter by $\tilde{\boldsymbol{h}}$, and the high-pass filter by $\tilde{\boldsymbol{g}}$, see fig. 1.1. The filters directly define the dilatation coefficients $\boldsymbol{h}_{\mathrm{mr}}, \boldsymbol{g}_{\mathrm{mr}}$ (and $\widetilde{\boldsymbol{h}}_{\mathrm{mr}}, \widetilde{\boldsymbol{g}}_{\mathrm{mr}}$ in the biorthogonal case).

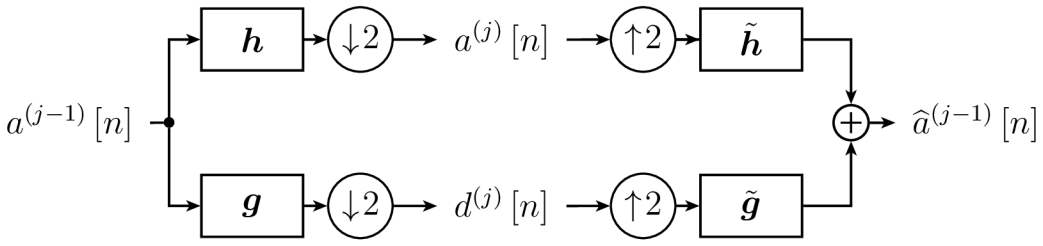


Figure 1.1: Perfect reconstruction two channel filter bank.

Wavelet Filters in Detail

The filters can be of both odd and even length. (Satisfying $m \geq 2$ at the same time, to make the filtering significant.) The so-called quadrature mirror filters, which are

always orthogonal, have all four identical lengths of $\boldsymbol{h}$, $\boldsymbol{g}$, $\tilde{\boldsymbol{h}}$, $\tilde{\boldsymbol{g}}$. The biorthogonal filters nevertheless can have different effective lengths and can achieve properties that orthogonal filters cannot (symmetry, linear phase). According to [22], one of the following cases is true (both for the decomposition and the reconstruction stages):

1. Both filters have odd lengths which differ by an odd multiple of two.
2. Both filters have even lengths, being either equal or differing by an even multiple of two.
3. One filter is of odd length, the other is of even length, and the zeros of both the filters are located at the unit circle.

To work with filters of different lengths consistently, the shorter one is zero-padded to the length of the longer one. Zeros, of course, do not affect the values at the output of the filter. (The "lifting scheme" [24] which would make use of the shorter length, is not exploited.) The rules for padding the shorter filter at both its ends follow immediately and they correspond with the Matlab Wavelet Toolbox [25] behavior.

In the following text, only the first two of the mentioned cases are considered — case 3 is of no practical interest [22]. Two nonnegative variables l_0 and r_0 are defined, denoting the number of zeros to be added from the left and the right end, respectively. Denoting the effective length of the shorter filter by $\underline{m}$, the following naturally holds: $m = l_0 + \underline{m} + r_0$.

In the case 1 (odd $m, \underline{m}$) the extensions are chosen so that $l_0 = r_0 - 2$, which leads to

$$l_0 = \frac{m - \underline{m}}{2} - 1, \quad r_0 = \frac{m - \underline{m}}{2} + 1. \tag{1.17}$$

In the case 2 (even $m, \underline{m}$) the extensions l_0, r_0 are equal, which induces

$$l_0 = r_0 = \frac{m - \underline{m}}{2}. \tag{1.18}$$

Whenever a particular wavelet filter is mentioned in the paper, its abbreviated labeling is taken over from [25].

Example 1: The biorthogonal filter bank `bior2.2` comprises the analyzing low-pass filter $\boldsymbol{h} = (h[0], \ldots, h[4])$ of length $m = \mathrm{len}(\boldsymbol{h}) = 5$ and the high-pass filter $\boldsymbol{g} = (g[0], g[1], g[2])$ of effective length $\underline{m} = \mathrm{len}(\boldsymbol{g}) = 3$. This corresponds to case 1, and according to (1.17), the extensions to be used are $l_0 = 0$ a $r_0 = 2$. Thus, the resultant padded high-pass filter is $(g[0], g[1], g[2], 0, 0)$.

Example 2: The biorthogonal filter bank `bior1.5`: the analyzing low-pass filter $\boldsymbol{h} = (h[1], \ldots, h[10])$ of length $m = \mathrm{len}(\boldsymbol{h}) = 10$, the high-pass filter $\boldsymbol{g} = (g[0], g[1])$ of only the effective length $\underline{m} = \mathrm{len}(\boldsymbol{g}) = 2$. Case 2 should be used now, and according to (1.18), the final extensions are $l_0 = r_0 = 4$. Thus the padded filter takes the form $(0, 0, 0, 0, g[0], g[1], 0, 0, 0, 0)$.

From now on, $\boldsymbol{h}$, $\boldsymbol{g}$, $\tilde{\boldsymbol{h}}$, and $\tilde{\boldsymbol{g}}$ will denote filters already extended to have an equal length m.

Remark 3: When filters of odd lengths are considered, there is one difference between the Matlab Wavelet Toolbox [25] and the process just described. The Wavelet Toolbox inserts an extra zero at the beginning of both the filters to make their lengths be even.

1.2 Discrete Wavelet Transform of Finite Length Signal

A question that immediately comes to mind when working with time-limited signals: Since the fundamental part of DWT is convolution and since convolution is known to exhibit "boundary artifacts", how should one compute the wavelet coefficients located "near the boundaries"?

Although this is not the main focus of this work, a summary of possible methods which answer the above question is presented in this section. Let us say in advance that all of the approaches suffer some shortcoming [18, 19, 25–28]. In this part of text, just a single level of the wavelet decomposition $J = 1$ is assumed (without loss of generality).

1. *Using special border filters.* In this case, the signal samples in the neighborhood of the borders are reconstructed using special filters. The signal is not extended in any way.
2. *Assuming periodicity.* The signal is considered to be one period of an infinite-length periodic signal. If, in addition, the signal length is even, then the total number of coefficients produced at the first level of decomposition is equal to the original number of samples.
3. *Defining samples outside of the original domain.* The idea here is that the samples beyond the signal domain are extrapolated using a more or less suitable and/or a more or less computationally demanding method. It is convenient to divide the possibilities into several groups:
   (a) *Mirroring.* The edge-samples are "mirrored" symmetrically. Such an approach brings "discontinuities" of the signals first difference. If symmetric filters (only the biorthogonal filters can be symmetric) are used, it is possible to make the DWT representation non-redundant (non-expansive).
   (b) *Point-symmetric extension.* [29] Using point-symmetry one can get rid of the discontinuity mentioned.
   (c) *Smooth extension using polynomials.* The method tries to "guess" samples outside of the signal domain using a polynomial of a specified order

(kth order polynomial preserves the "continuity" of k-th derivative).

(d) *Extending with zeros.* This is the simplest method — the signal is considered to be zero beyond its original borders.

4. *Using samples from the neighboring segment.* This approach is natural when the signal to be processed is in fact a time-limited portion cut from a longer signal. For example, in real-time speech processing the buffer holds 256 samples. This type of method reasonably uses samples directly from its neighbor(s) to extend the borders.

   In this sense, such a method could be considered a special case of group 3. Nevertheless, it is listed separately because in the case of the decomposition depth being $J > 1$, the recursive nature of the DWT makes the necessary extension length greater when compared with the other methods. Such situation requires more detailed treatment and modification of DWT and forms one of the goals of this dissertation.

5. *Cutting off.* The goal of this naive approach is to keep the wavelet representation non-redundant. The DWT computation is performed using any of the above methods and then the "border" coefficients are simply discarded. Therefore the reconstruction cannot be exact near the borders any more.

Each of the stated methods suffers at least one shortcoming from the following list:

- the necessity of having special border filters (which is not effective algorithmically),
- the deviation from (bi)orthogonality of the transform,
- inexact reconstruction from the transform coefficients,
- redundancy (expansivity) of the wavelet representation,
- possible errors at the "other end" due to periodicity.

Hence, in choosing a method, one always has to make a compromise.

We find the extension methods given under item 3 (and possibly 4) to be the most natural and the most generally utilizable in practice; such methods have only one drawback — expansivity — which means that the wavelet representation of a signal will have the total number of coefficients a bit higher than number of the input samples. As mentioned in 3a, there exist special situations when expansivity does not appear — this is typical of image processing with biorthogonal filters, for example [13]. Because of its universality, the generally expansive case 3 is considered exclusively.

1.2.1 Mallat's algorithm for finite length signals

Since the Mallat's algorithm application on finite-length signals may not be clear at a first glance, it is established in the following text.

Algorithm 4:[Decomposing pyramid algorithm DWT] Let $\boldsymbol{x}$ be a discrete input signal of length len($\boldsymbol{x}$), the pair of wavelet decomposition filters with length m is defined as $\boldsymbol{g}$ and $\boldsymbol{h}$, J is a positive integer denoting the decomposition depth. Also, the type of boundary treatment has to be known:

1. Denote the input signal $\boldsymbol{x}$ by $\boldsymbol{a}^{(0)}$ and set $j = 0$.
2. One decomposition step:
   (a) *Extending the input vector.* Extend $\boldsymbol{a}^{(j)}$ 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 $\boldsymbol{g}$.
   (c) *Cropping.* Take just its central part from the result, so that the remaining "tails" on both the left and the right sides have the same length $(m-1)$ samples.
   (d) *Downsampling.* Downsample the resultant vector. Denote the result by $\boldsymbol{d}^{(j+1)}$ and store it. Than repeat items b) d), now with filter $\boldsymbol{h}$, denoting and storing the result as $\boldsymbol{a}^{(j+1)}$.
3. Increase j by one. If it now holds $j < J$, return to item 2, in the other cases the algorithm ends.

After Algorithm 4 finishes, the wavelet coefficients are stored in $J+1$ vectors (of different lengths) $\boldsymbol{a}^{(J)}, \boldsymbol{d}^{(J)}, \boldsymbol{d}^{(J-1)}, \ldots, \boldsymbol{d}^{(1)}$.

Algorithm 5:[Reconstruction pyramid algorithm DWT]

Given are: pair of wavelet reconstruction filters of length m – the highpass filter $\tilde{\boldsymbol{g}}$ and the lowpass filter $\tilde{\boldsymbol{h}}$, number of signal samples in the time domain len($\boldsymbol{x}$) and most importantly the $J+1$ vectors of wavelet coefficients $\boldsymbol{a}^{(J)}, \boldsymbol{d}^{(J)}, \boldsymbol{d}^{(J-1)}, \ldots, \boldsymbol{d}^{(1)}$ which are the result of the alg. 4.

1. Set $j := J$.
2. One level of decomposition:
   (a) *Upsampling.* Perform the even type upsampling of the vectors $\boldsymbol{a}^{(j)}$ and $\boldsymbol{d}^{(j)}$.
   (b) *Filtration.* Filter upsampled, vectors i.e. perform a convolution with reconstruction filters $\tilde{\boldsymbol{h}}$ and $\tilde{\boldsymbol{g}}$ respectively.
   (c) *Sum.* Add up outcomes of both filtrations.
   (d) *Cropp off.* From the sum, keep just the "middle" part which length is equal to the length of vector $\boldsymbol{d}^{(j-1)}$ skipping $m-1$ samples from the beginning. When $j = 1$ consider the length of the non existing vector $\boldsymbol{d}^{(0)}$ to be len($\boldsymbol{x}$).

Denote the resulting vector as $\boldsymbol{a}^{(j-1)}$.

3. Decrease j by one. If $j > 0$, than go to step 2., else the algorithm ends.

The result of the algorithm (the reconstructed signal) is in $\boldsymbol{a}^{(0)}$ after the algorithm ends.

1.2.2 Noble Multirate Identity

According to [30], the order of the FIR filters and the downsamplers/upsamplers can be interchanged assuming FIR filter impulse response resampling. The property is called *noble multirate identity* and it is shown in fig. 1.2. Using the property, the iterated filter bank can be transformed into a non-iterated filter bank. An example of such operation for $J = 3$ is shown in figures 1.4a and 1.4b for the analyzing filter bank and in figures 1.5a and 1.5b for the reconstruction filter bank. The amplitude frequency response and impulse responses of the analyzing multirate identity filter bank are shown in fig. 1.4c and 1.5c respectively.

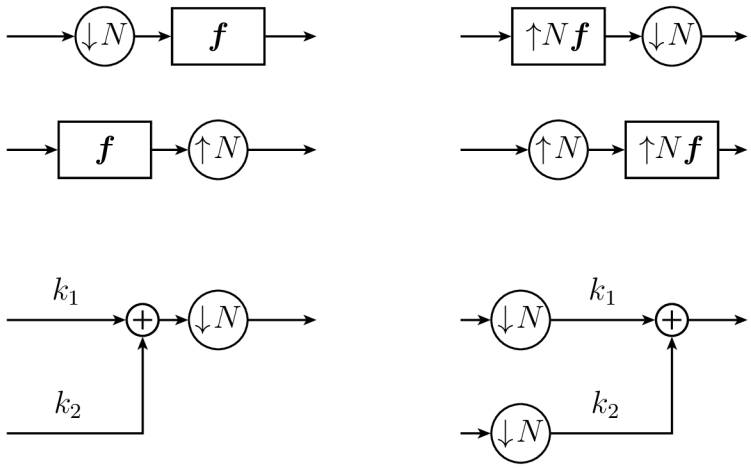


Figure 1.2: Multirate noble identities, commuting operations.

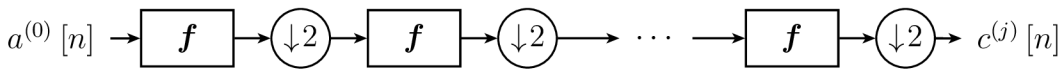


Figure 1.3: Path trough an iterated filter bank.

For the purposes of the thesis, it is crucial to derive lengths of the impulse responses of the multirate identity filter bank. Given the length of the wavelet filters m (possibly zero padded in the biorthogonal filter bank case) and given the path trough the tree-structured analyzing iterated filter bank (from $\boldsymbol{a}^{(0)}$ to $\boldsymbol{c}^{(j)}$)

where arbitrary basic filter $\boldsymbol{h}, \boldsymbol{g}$ is denoted as $\boldsymbol{f}_1$ ($\text{len}(\boldsymbol{f}_1) = m$) and accordingly its 2^i-times upsampled version as $\boldsymbol{f}_i$ and assuming the odd-type of upsampling it can be written

$$\text{len}(\boldsymbol{f}_i) = \text{len}(\uparrow 2^i \boldsymbol{f}_1) \tag{1.19}$$

$$\text{len}(\uparrow N\boldsymbol{f}_1) = N \cdot \text{len}(\boldsymbol{f}_1) - (N-1) = N \cdot m - (N-1) \tag{1.20}$$

and the length of convolution of two impulse responses is equal to,

$$\text{len}(\boldsymbol{f}_i * \boldsymbol{f}_j) = \text{len}(\boldsymbol{f}_i) + \text{len}(\boldsymbol{f}_j) - 1 \tag{1.21}$$

then the length of the resulting single identical filter can be written as

$$\text{len}(\boldsymbol{f}_1 * \cdots * \boldsymbol{f}_j) = m + 2 \cdot m - 1 - 1 + \cdots + 2^J \cdot m - (2^J - 1) - 1 \tag{1.22}$$

which can be rewritten to

$$\text{len}(\boldsymbol{f}_1 * \cdots * \boldsymbol{f}_j) = (2^j - 1)m - (2^j - 2). \tag{1.23}$$

1.2.3 Lifting Scheme

The lifting scheme representation of the wavelet filter bank was introduced by Sweldens in [31] and according to [24], every wavelet filter bank can be decomposed (factored) into elementary lifting steps. In addition, lifting scheme brings yet another way of designing wavelets using custom combinations of these elementary lifting steps. Every transform by the lifting scheme can be inverted and it is performed by a mere reversion of the lifting steps. The computation itself can be done in-place (no external memory needed) and the computation cost can be reduced compared to convolution. The factors are not unique so a considerable effort was devoted to finding effective ones [24, 32, 33] because not every factorization is more effective than the original filter bank. The most famous is the CDF9/7 wavelet factorization, employed in the JPEG2000 standard [13]. Again, the factorization process is not the aim of this work and the already known factors will be used.

Another feature of the lifting scheme is that rounding the results of predict and update operation allows transformation which maps integers to integers, usable especially for lossless data compression [31].

The LWT can also be generalized to non-translation invariant grids and allows adaptivity of subsequent lifting steps [34]. However, these extensions are not concerned in this work.

1.3 Multidimensional Discrete Wavelet Transform

There are two ways of extending the discrete wavelet transform to multiple dimensions [35]. There is the anisotropic and isotropic multidimensional wavelet transform. The anisotropic transform consists of J level DWT of rows and then J level DWT of columns (the roles of rows and columns are interchangeable), but this is not preferred in practice. The isotropic version of the multidimensional WT is used almost exclusively. This can be seen as the multidimensional separable **orthogonal** basis is being built using a tensor product of one-dimensional subspaces. For two-dimensional signals the following equation holds

$$\begin{aligned}\mathcal{V}^{(j-1)} \otimes \mathcal{V}^{(j-1)} &= (\mathcal{V}^{(j)} \oplus^{\perp} \mathcal{W}^{(j)}) \otimes (\mathcal{V}^{(j)} \oplus^{\perp} \mathcal{W}^{(j)}) \\ &= \left(\mathcal{V}^{(j)} \otimes \mathcal{V}^{(j)}\right) \oplus \left(\mathcal{V}^{(j)} \otimes \mathcal{W}^{(j)}\right) \oplus \left(\mathcal{W}^{(j)} \otimes \mathcal{V}^{(j)}\right) \oplus \left(\mathcal{W}^{(j)} \otimes \mathcal{W}^{(j)}\right).\end{aligned} \tag{1.24}$$

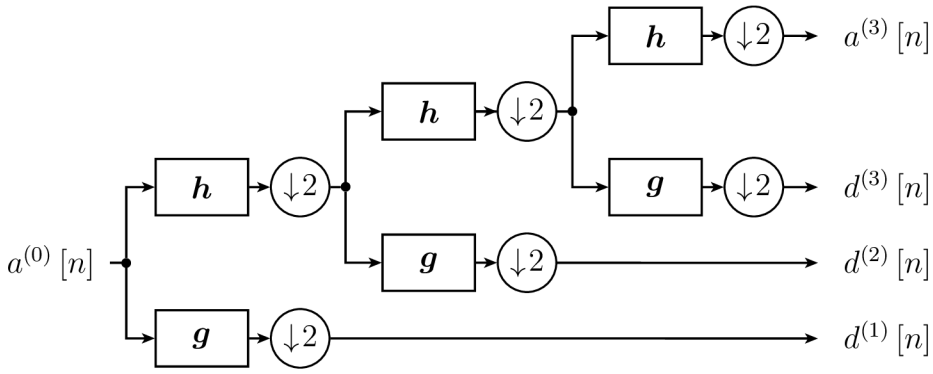
The approximation subspace is denoted as $\mathcal{V}^{(j)} \otimes \mathcal{V}^{(j)}$ and there are another three detail subspaces: horizontal, vertical and diagonal detail spaces

$$\mathcal{W}_{\mathrm{H}}^{(j)} = \mathcal{V}^{(j)} \otimes \mathcal{W}^{(j)}, \quad \mathcal{W}_{\mathrm{V}}^{(j)} = \mathcal{W}^{(j)} \otimes \mathcal{V}^{(j)}, \quad \text{and } \mathcal{W}_{\mathrm{D}}^{(j)} = \mathcal{W}^{(j)} \otimes \mathcal{W}^{(j)}. \tag{1.25}$$

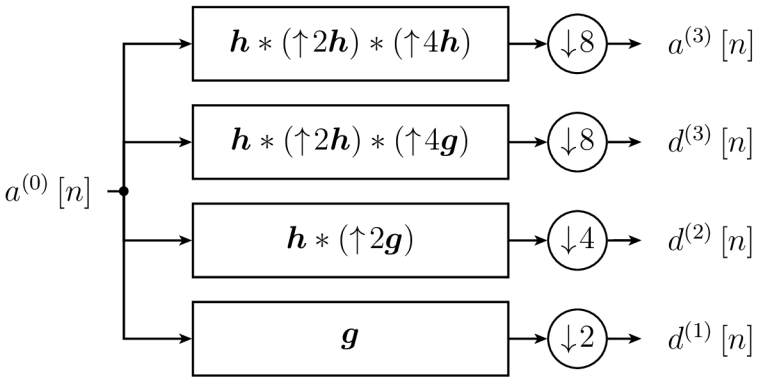
Again, the level j approximation subspaces are nested and the union of detail spaces at level j is its orthogonal complement to the coarse subspace at level one less.

By extending the equation (1.24) to even more dimensions, one can conclude that in D dimensions, there are $2^D - 1$ detail subspaces in addition to the approximation subspace, resulting into total of $J(2^D - 1) + 1$ subspaces. The multidimensional basis vectors are also tensor products of the respective one-dimensional basis vectors and they are separable with respect to the individual dimensions. Therefore, each level of the transform can be done one-dimension at a time using multiple fast wavelet transforms. In case of two-dimensional signals, first the rows, then the columns are processed (or vice versa) as shown in fig. 1.6 (left).

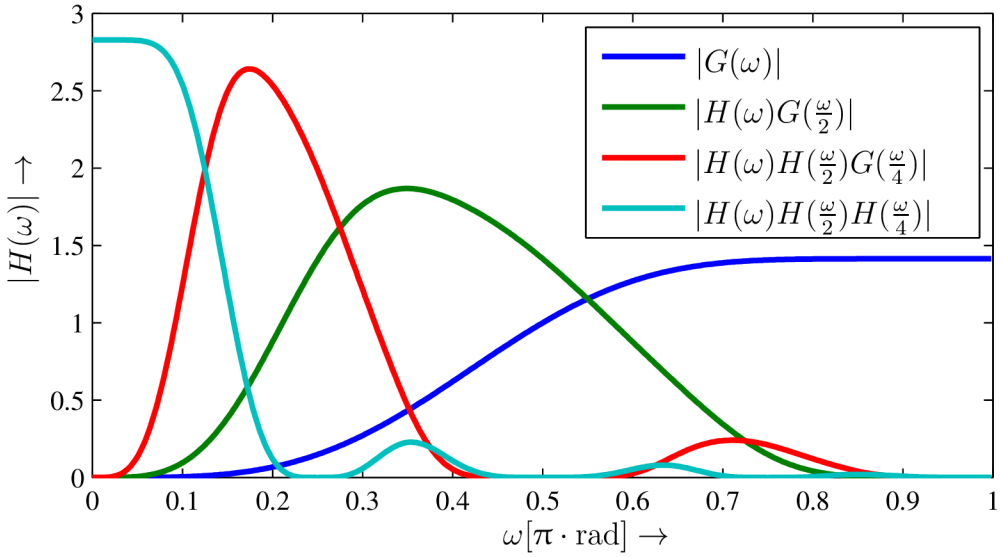
The isotropic multidimensional transform results in the *non-standard* division of spectra [19], see idealized separation of frequency bands for $D = 2$ and $J = 3$ in fig. 1.6 (right). In fig. 1.7 (right), there is a concrete example of the wavelet representation of the Lena image using level $J = 3$ and `CDF9/7` wavelet with symmetric boundary handling.



(a)

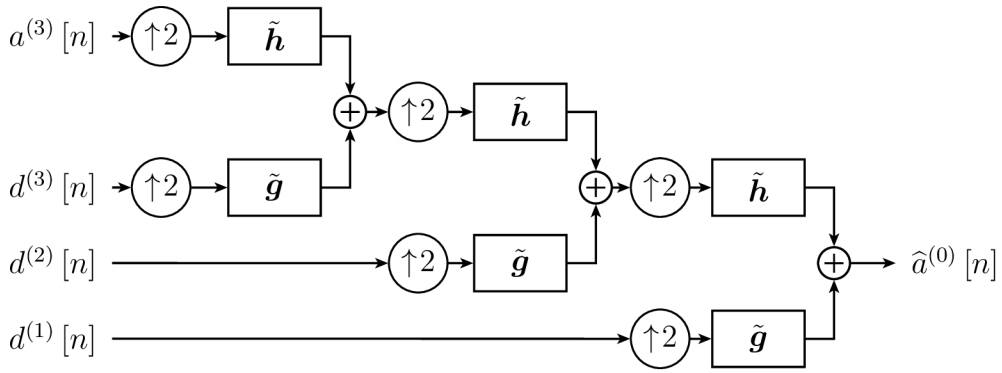


(b)

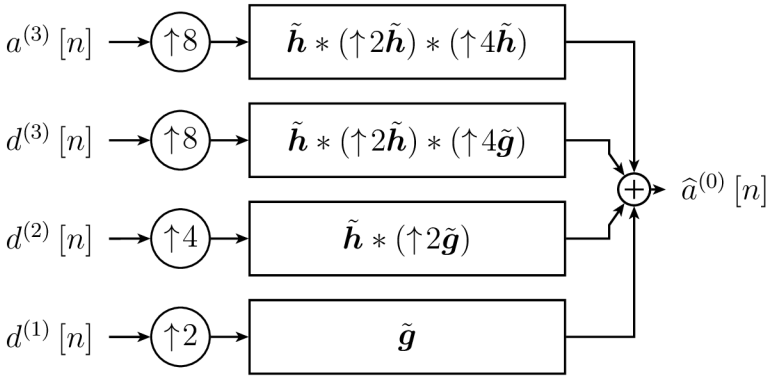


(c)

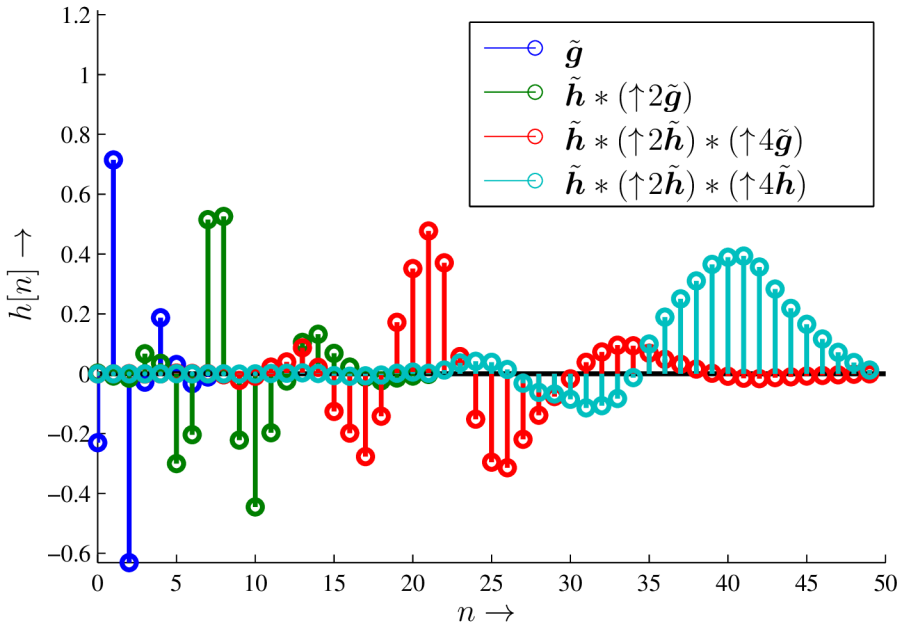
Figure 1.4: Iterated filter bank for pyramidal algorithm DWT with $J = 3$. (a) Analysing iterated filter bank according to fast DWT. (b) Noble multirate identity of the iterated filter bank. (c) Module frequency response of the noble multirate identity.



(a)



(b)



(c)

Figure 1.5: Reconstruction iterated filter bank for pyramidal algorithm DWT with $J = 3$.(a) Reconstructing iterated filter bank according to fast DWT.(b) Noble multirate identity of the iterated filter bank.(c) Impulse responses of the noble multirate identity.

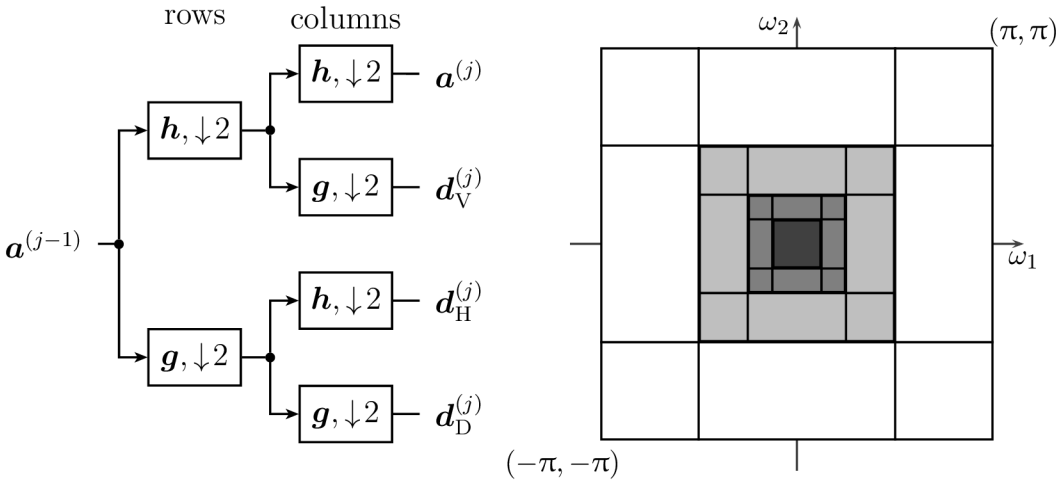


Figure 1.6: (Left) One level of filter bank for a non-standard division of the spectra. (Right) An idealized non-standard division of the spectra for $J = 3$.

Figure 1.7: Two-dimensional separable wavelet decomposition of Lena image, using `CDF9/7` wavelet and $J = 3$. Logarithm of absolute values of coefficients is displayed. The representation is not expansive because of both symmetrical filters are symmetrical boundary extensions.

2 MOTIVATION AND STATE-OF-THE-ART

Often in practice, there are situations when the input signal cannot be loaded and processed at once or it is simply not yet known. The input signal is therefore loaded one segment at a time, transformed to the wavelet domain, where all desirable coefficient processing takes place and than transformed back to the original domain. The importance of the border treatment is magnified for the border artifacts can become a great issue. This chapter summarizes approaches to the segmented wavelet processing.

Firstly, the shortcomings of the so-called "naive" approach to the segmentwise computation of DWT will be shown. In this approach, no segment overlap is exploited and the segments are transformed using common border extension techniques independently. The perfect reconstruction is achieved if the wavelet coefficients are not subject to any kind of processing. Doing so, the artifacts at the borders rise up after the reconstruction when compared to the whole signal reconstruction. Fig. 2.2 shows such situation at the 20th row of pixels taken form the `Lena` image. The setup is as follows: 4 level decomposition is used with `db4` wavelet, the wavelet coefficients are hard-thresholded with $\lambda = 150$ i.e. all coefficients with absolute value less than λ are zeroed. Sorted wavelet coefficients before and after thresholding are shown in fig. 2.1.

In addition, 2^J-shift invariant property of the DWT restricts segment division lines to be multiples of 2^J, otherwise additional inaccuracies can be introduced provided a standard implementation of DWT is used. The example in fig. 2.2 satisfies this criterion.

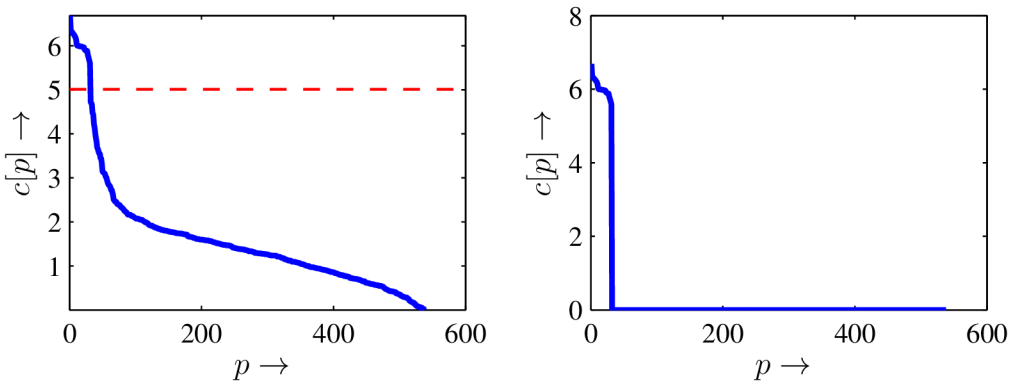


Figure 2.1: (Left) Log of the sorted absolute values of wavelet (both approximation and detail) coefficients and the threshold $\lambda = 150$. (Right) Values smaller than threshold are set to zeros.

The border artifacts are clearly visible in fig. 2.2.

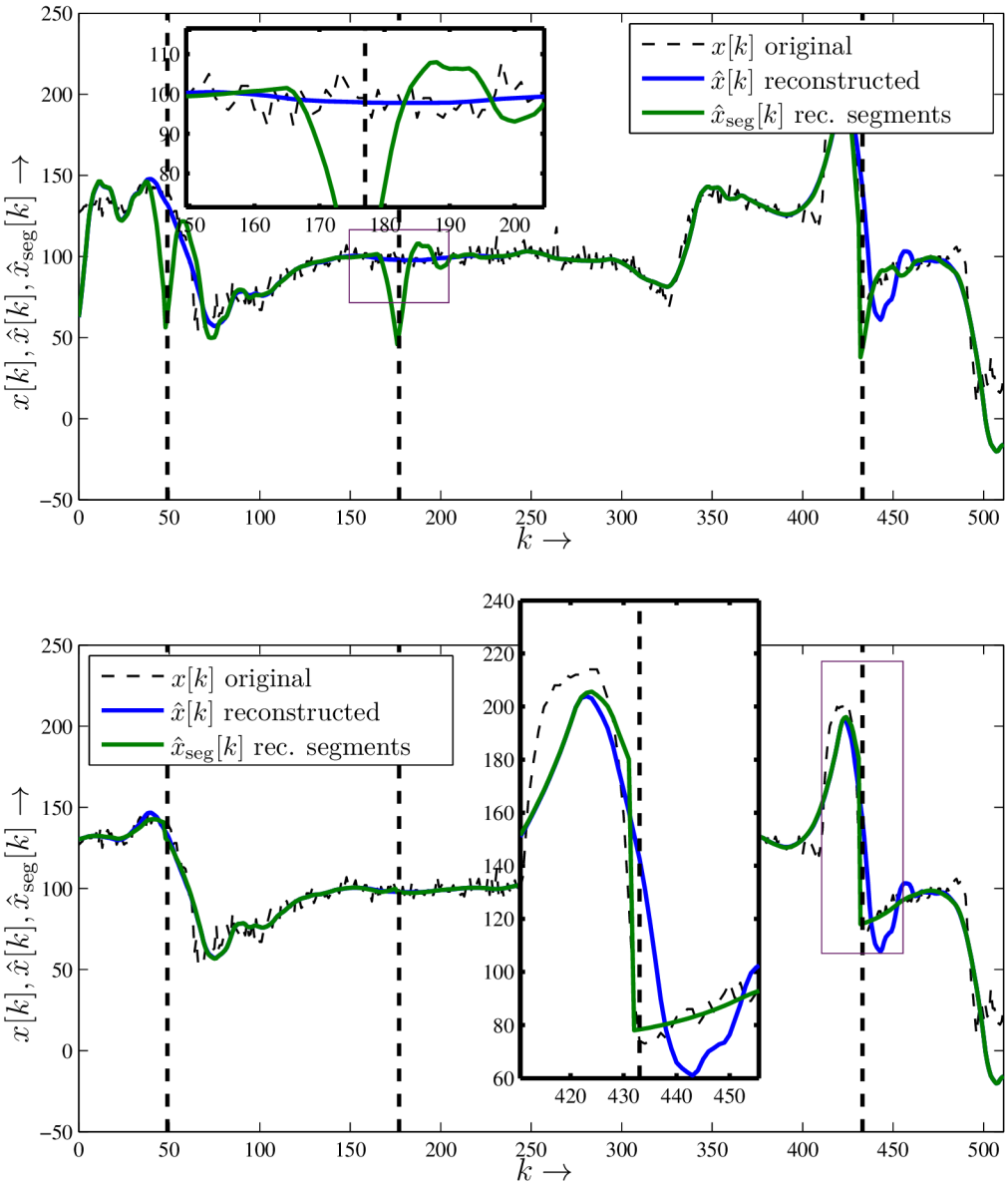


Figure 2.2: Artifacts at the borders of the segments. The dividing lines are at indexes $k = 48, 176, 432$. The graphs show intensities $x[k]$ of pixels from the 20th row of the grayscale `Lena` image. The signal $x[k]$ was transformed to the wavelet domain (level 4, wavelet `db4`) by the DWT algorithm, wavelet coefficients were hard-thresholded with $\lambda = 150$ and than used for reconstruction. The reconstructed signal $\hat{x}_{\text{seg}}[k]$ was obtained by processing coefficients belonging to individual segments, whereas $\hat{x}[k]$ by processing the whole input signal. Samples beyond segment boundaries were assumed to be zeros (above) and symmetrically mirrored (below). The border artifacts are clearly visible, although the symmetrical extension performs better in this situation.

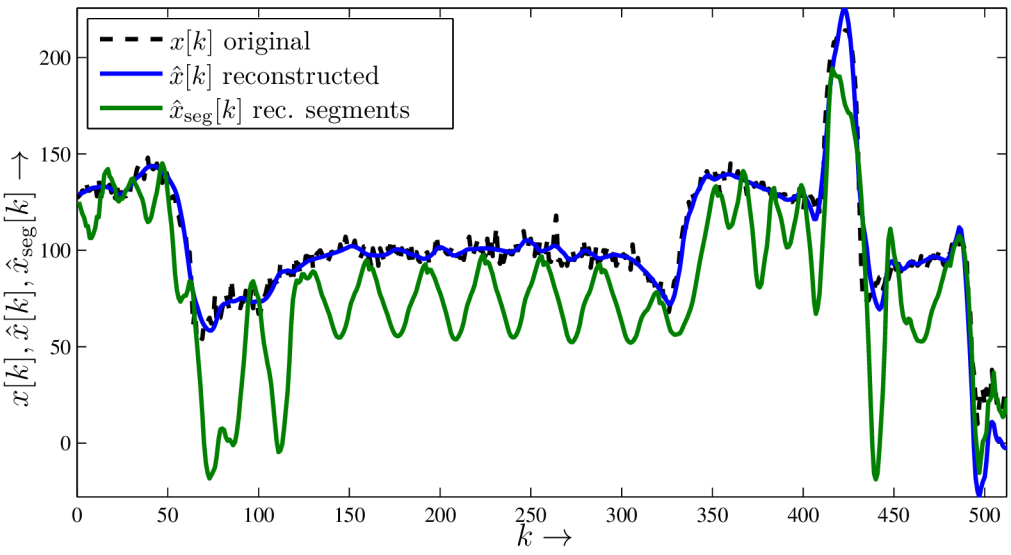


Figure 2.3: Reconstructed signal degradation using general windowing with overlap. The setup was the same as in fig. 2.2, 64 sample triangle window with 50% overlap was used.

The next possible approach, adopted from the short-time Discrete Fourier Transform, is based on signal windowing and overlapping the resultant segments. However, even when invertible (BUPU – bounded uniform partition of unity) windowing is used, severe problems are introduced provided the wavelet coefficients are subject to nonlinear processing (see fig. 2.3) or even to linear processing, which is not carried out coefficient-wise, not to mention considerable potential numerical errors at the window tails. Such approach is discussed in [36].

The state-of-the-art methods which can be found in the literature and which treat the border problem differently will be discussed in the following text. However, most of the methods seems to be 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 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 [37] could be 96 samples). The methods can be divided into two classes according to their purpose (and set of drawbacks). In the first class, there are methods for *real-time* wavelet transform which tend to allow small errors and in the second class, there are methods for *parallel* computation of the wavelet transform of images which calculate the wavelet transform exactly, but they are usually tailored to the specific wavelet filter or just to wavelet analysis. Moreover, the calculations are synchronized at each level of decomposition for the

purpose of data exchange or partially calculated coefficients completion.

In [38], the border error-free method for the wavelet packet transform in the audio coder setting (nonlinear wavelet coefficient processing) is introduced. The idea is transferable to the wavelet transform and, for the wavelet analysis, it is based on reusing the last $m-2$ approximation coefficients at each level j from the previous segment, achieving correct wavelet coefficients. However, the synthesis process is not derived for arbitrarily filter lengths and its description is somewhat confusing and therefore it is not clear whether the reconstruction is meant to be exact. Further, the method clearly works only with segments with length equal to a power of two, and it is restricted to the consecutive order of the segments.

The paper [39] describes a framework for linear time-domain digital audio effects performed directly in the wavelet domain. The *shift invariant* wavelet transform is employed using signal circular shifts. The segment lengths are again restricted to a power of two. The border-end effect treatment method is built upon [38] but it reuses the whole previous segment so that the input segment size is doubled. The reconstruction segment length is preserved. This approach is somewhat "ad-hoc" and can fail for more demanding combinations of filter lengths m and depths of decomposition J or it can introduce a considerable redundancy of computations when m and J are of small values especially for multidimensional signals.

Another attempt for real-time nonlinear wavelet processing (thresholding for denoising) was introduced in [40]. The extensible moving window with a constant step is employed but common border extension techniques are used.

The paper [41] performs rather general segmented computation of the wavelet packet analysis (forward transform only) with arbitrary number of channels using segment overlap. Although it is not stated explicitly, the segment length restriction is lessened to a multiple of 2^J, where J is the depth of the deepest branch of the wavelet packet decomposition (depth of decomposition in the DWT case). Authors claim that the boundary distortion was removed but from the results, it is clear that it is not true for some combinations of J and m. Moreover, the overlaps seem unnecessary high when compared to the further described SegDWT algorithm.

Authors of [42] bring an interesting approach to the segmented computation of the forward DWT using a lifting scheme. They use postprocessing of the partially transformed wavelet coefficients near the boundaries. No prior overlaps are used but after the forward transform of two adjacent segments, the ending coefficients of the first one and the beginning coefficients of the latter one are exchanged and they are subjected to the postprocessing to achieve correct values. The method not seems to restrict the segment lengths but adaptation of the method to the real-time setting assuming requirement for the wavelet coefficient processing would be difficult, not to mention the lack of the inverse transform.

The methods for parallel computation of the forward wavelet transform tailored to multiprocessor architectures with a message passing interprocessor communication were presented in [43] and later in [44] for the lifting scheme. The methods are based on exchanging samples from neighboring segments as needed after one level of the decomposition is calculated. The main focus of the papers is enumeration and optimization of the message sizes. Again, the inverse transform is omitted and the methods are not easily transferable to a real-time setting.

The paper [45] deals with another parallelization of the 2D-DWT using CUDA architecture, but the segmented approach is not considered here. Rows and columns of the image are taken as a whole.

Another approach to parallelization of lifting scheme 2D-DWT using CUDA is taken in [46]. To use devices' memory effectively, the sliding window with overlap is used when processing columns of the image. However, only one level of the transform is done in each sliding window run.

To the author's best knowledge, there is but one algorithm which allows to perform *exact* wavelet analysis *and* synthesis with a segment at the same time provided equality of coefficients and reconstruction is preserved compared to the whole signal wavelet analysis and synthesis – the SegDWT algorithm [10]. It employs sophisticated segment overlaps for a correct wavelet coefficient synchronization and an exact reconstruction (as if the signal had not been segmented). The segment length is arbitrary as well as the depth of the decomposition and filter lengths.

However, there is one more thing: neither of the described methods allow segments of varying sizes. It does not seem to be an issue for one-dimensional real-time signal processing but it can become an issue in the case of the parallel execution when segments of equal length prevent an effective load balancing between processing units.

2.1 SegDWT Algorithm

Since the SegDWT algorithm is the cornerstone of the thesis, it will be described in detail. The algorithm processes the signal segment-by-segment and it comprise of analysis (forward) and synthesis (inverse) parts and both of them consist of several steps:

- Analysis (forward) part:
  - Extension of the actual segment.
  - Application of the (modified) Mallat's algorithm.
  - Removal of redundant coefficients.

  - The result consists of vectors of "full" wavelet coefficients, which are ready to be processed.
- Synthesis (inverse) part:
  - Zero padding of the input wavelet coefficients vectors.
  - Application of the inverse Mallat's algorithm.
  - Addition of the overlap from the previously reconstructed segment.

The analysis part is in principle similar to the overlap-save algorithm (OLS) for the linear convolution, while the synthesis part to the overlap-add (OLA).

Overlap methods for linear convolution are well known in conjunction with fast convolution in spectral domain using FFT (circular convolution). Despite the fact that the fast convolution is not used in the SegDWT (the reasoning is given in sec. 7.1.2), the principles are valid even in the time domain. First, the linear convolution process of one segment is depicted in fig. 2.4. The well known formula for the linear convolution of two finite-length signals $\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$ is

$$y[n] = \sum_{k=0}^{m-1} h[k]x[n-k], \tag{2.1}$$

for $x[n]$ being the input signal segment of length s, $h[k]$ being the impulse response of length m and $y[n]$ denotes the output signal of length $s+m-1$ for $n = 0, \ldots, s-1+m-1$.

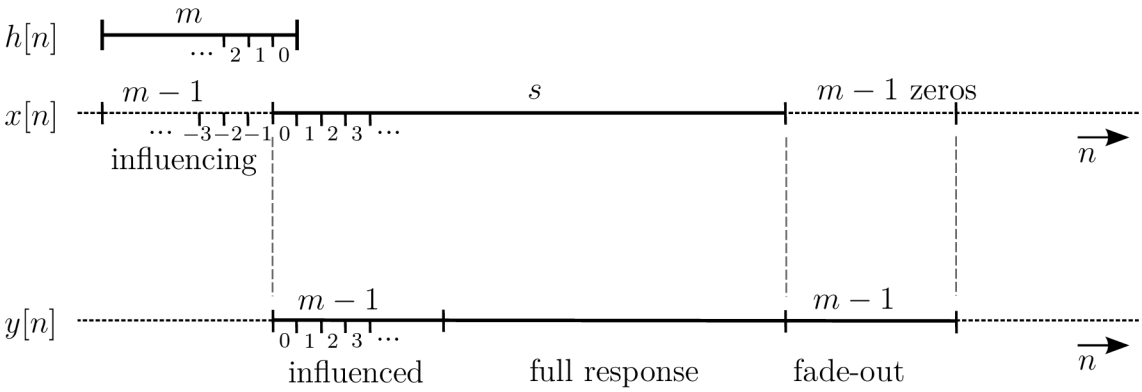


Figure 2.4: Segment convolution in detail. Segment of length s from input signal $x[n]$ is linearly convolved with impulse response $h[n]$ of length m. Redrawn from [47] and modified.

The **OLS** algorithm reuses last $m-1$ samples (influencing samples) from a previous segment and it does not calculate $m-1$ "fade-out" samples. $m-1$ samples from the beginning are discarded afterwards for they are used only to make the $m-1$ influenced samples into full response (fig. 2.5). Usage of the algorithm is shown at fig. 2.6.

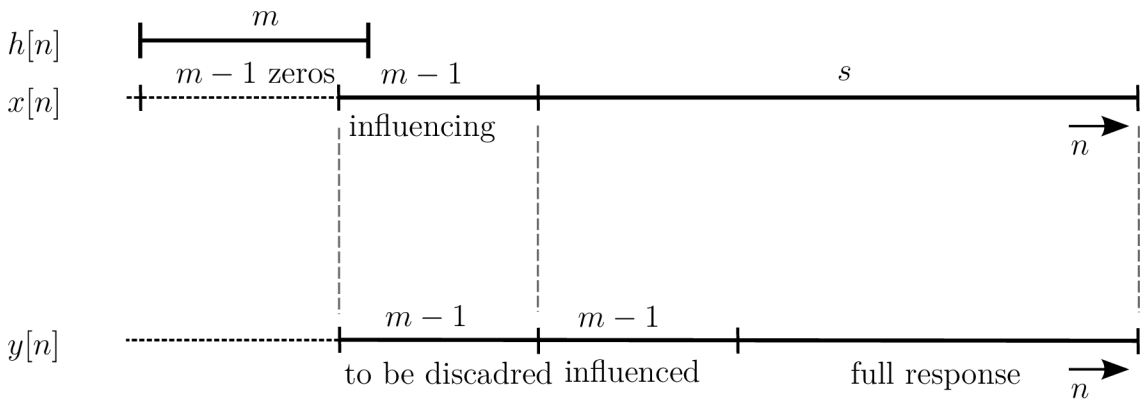


Figure 2.5: OLS Segment convolution in detail.

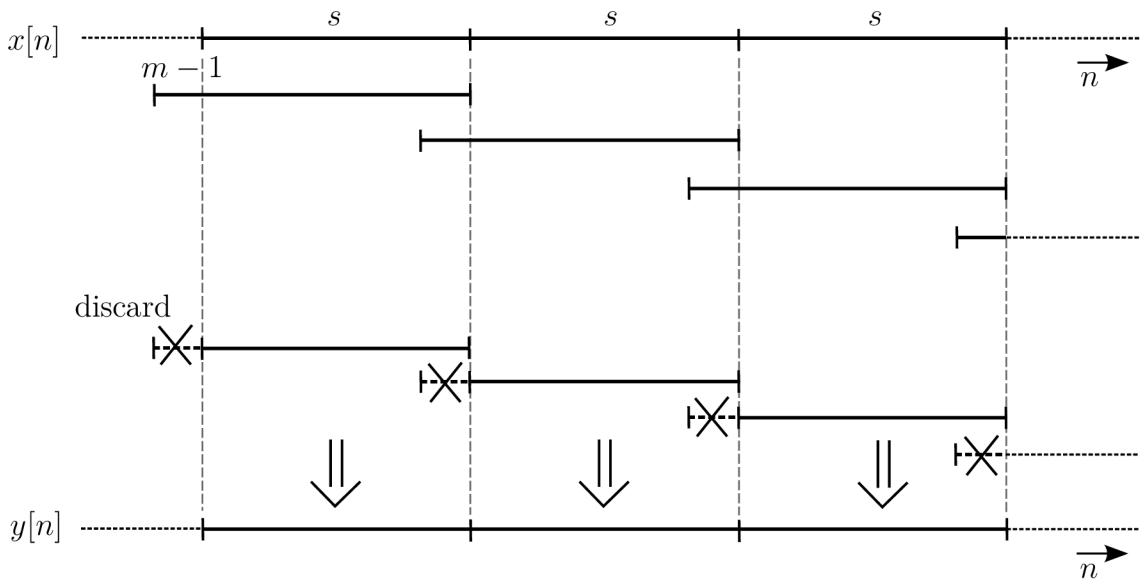


Figure 2.6: Overlap-Save Algorithm in detail. Last $m-1$ samples of each segment are "saved" for the following segment. Redrawn from [47] and modified.

The **OLA** algorithm always uses zero padding i.e. $m-1$ zero samples are appended to the end of each segment. The fade-out samples are then held to be added to the first $m-1$ influenced samples of the following segment (fig. 2.4). Usage of the algorithm is shown at fig. 2.7.

2.1.1 Algorithm description

This section describes the actual SegDWT algorithm as it was presented in [10]. The SegDWT algorithm was developed for FIR orthogonal filter banks, but FIR biorthogonal filters can also be used if zero padded to the same length according to section 1.1.

The one-dimensional input signal $\boldsymbol{x}$ is divided into $N \geq 1$ segments of equal

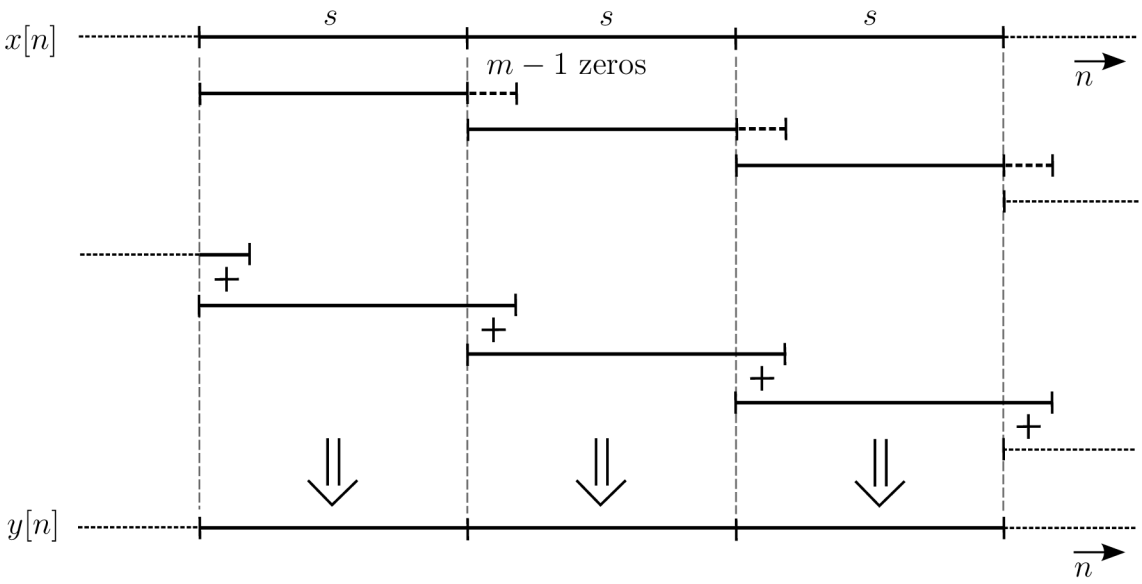


Figure 2.7: Overlap-Add algorithm in detail. $m-1$ zero samples are appended to the end of each segment. After convolution, these samples represent overlap, which has to be added to the beginning of the next segment. Redrawn from [47] and modified.

length s. The last one can be shorter than s. To achieve a correct follow-up of two sets of wavelet coefficients at decomposition level j it is necessary that the two consecutive segments to be properly extended. It has been shown that the two consecutive segments must have

$$r(j) = (2^j - 1)(m - 1) \tag{2.2}$$

input samples in common after they were extended. This extension has to be divided into the right extension of the first segment (of length R) and the left extension of the following segment (of length L) so that $r(J) = R + L$, however $R, L \geq 0$ cannot be chosen arbitrarily. The minimum suitable right extension of the segment n for $n = 0, 2, \ldots, N-2$ is

$${}^{n}R_{\min} = 2^J \left\lceil \frac{(n+1)s}{2^J} \right\rceil - (n+1)s, \tag{2.3}$$

and the maximum left extension of segment $(n+1)$ is

$${}^{n+1}L_{\max} = r(J) - {}^{n}R_{\min}. \tag{2.4}$$

The algorithm works such that it reads (receives) individual segments of the input signal, it 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.

For simplicity, the whole signal border extension method is assumed to be zero padding, but the transition to different treatments is straightforward. The algorithm is stated as follows:

Algorithm 6:[SegDWT analysis v. 1.0] Let the wavelet filters $\boldsymbol{g}, \boldsymbol{h}$ of length m, decomposition level J and boundary treatment be given. The input signal $\boldsymbol{x}$ is divided into N segments of equal length $s \geq 2^J$ and the segments are denoted by ${}^0\boldsymbol{x}, {}^2\boldsymbol{x}, {}^3\boldsymbol{x}, \ldots, {}^{N-1}\boldsymbol{x}$.

1. Set $n = 0$.
2. Read the first segment, ${}^0\boldsymbol{x}$, label it as "current" and extend it from left by $r(J)$ zero samples.
3. **If** the current segment is also the last one ($n = N - 1$) at the same time, compute DWT of this segment using Algorithm 4 and finish.
4. Load $(n + 1)$ segment and label it as "next".
5. **If** the next segment is the last one:
   (a) Combine the current ${}^n\boldsymbol{x}$ and the next segment ${}^{n+1}\boldsymbol{x}$, set this new segment as current (the current becomes the last one).
   (b) Extend the current segment by $r(J)$ zero samples from the right.
   (c) Calculate DWT of depth J from the extended current segment using the Algorithm 4.

   Else
   (d) Determine ${}^{n+1}L_{\max}$ for the next segment and ${}^nR_{\min}$ for current segment using formulas (2.3) and (2.4).
   (e) Extend current segment from the right by ${}^nR_{\min}$ samples from the next segment. Extend the next segment from the left by ${}^{n+1}L_{\max}$ samples from the current segment.
   (f) Calculate the DWT of depth J from the extended current segment using the algorithm 4 with omitting step 2(a).
6. Modify the vectors containing the wavelet coefficients by trimming off a certain number of redundant coefficients from the left side, specifically: at the level j, $j = 1, 2, \ldots, J - 1$ trim off $r(J - j)$ coefficients.
7. **If** the current segment is the last segment, trim off the vectors in the same manner as in the previous step $r(J - j)$ but this time from the right.
8. Store the result as ${}^n\boldsymbol{a}^{(J)}, {}^n\boldsymbol{d}^{(J)}, {}^n\boldsymbol{d}^{(J-1)}, \ldots, {}^n\boldsymbol{d}^{(1)}$.
9. **If** the current segment is not the last one, set the next segment as current, increase n by 1 and go to item 4.

The output of Algorithm 6 is $N(J + 1)$ vectors of wavelet coefficients

$$\left\{ {}^i\boldsymbol{a}^{(J)}, {}^i\boldsymbol{d}^{(J)}, {}^i\boldsymbol{d}^{(J-1)}, \ldots, {}^i\boldsymbol{d}^{(1)} \right\}_{i=0}^{N-1} \tag{2.5}$$

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

Blocks of wavelet coefficients produced segment-by-segment by the forward part of the SegDWT constitute the input for the inverse algorithm. Analogue to the forward case, we use the boolean flag *last*, which becomes true if the very last segment is to be processed.

In addition to that, the signal parity is kept (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 SegDWT 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.

Algorithm 7:[SegDWT synthesis v. 1.0] Let the decomposition depth J be given, as well as wavelet reconstruction filters $\tilde{\boldsymbol{g}}$ and $\tilde{\boldsymbol{h}}$ of lengths m, and coefficients ${}^{n}\boldsymbol{a}^{(J)}$,${}^{n}\boldsymbol{d}^{(J)}$,${}^{n}\boldsymbol{d}^{(J-1)}$, ..., ${}^{n}\boldsymbol{d}^{(1)}$ for all n.

1. Set $n = 0$. Set $last = 0$.
2. **If** $last = 1$, then the Algorithm ends.
3. Read the n block of coefficients and update "*last*".
4. Extend the detail coefficients: at the level j, $j = 1, \ldots, J-1$, append $r(J-j)$ zero coefficients from the left side.
5. Compute the inverse transform of depth J using Algorithm 5 with omitting the cropping part.
6. **If** $n \neq 0$, 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.

The analysis and synthesis parts of the SegDWT algorithm can be both used on the actual segment consecutively thus forming a universal algorithm for any kind of wavelet coefficient processing task in real-time. The algorithm usage in this setup is shown in fig. 2.8 and fig. 2.9.

2.1.2 Algorithm Remarks

Extensions of the first ($n = 0$) and the last ($n = N - 1$) segments are treated differently and their values are

$$ {}^{0}L_{\max} = {}^{N-1}R_{\min} = r(J). \tag{2.6} $$

Given the actual segment ${}^{n}\boldsymbol{x}$ and its extended version ${}^{n}\boldsymbol{x}_{\text{ext}}$, the length of the coefficient vectors ${}^{n}\boldsymbol{c}_{\text{ext}}^{(j)}$ at levels $j = 1, \ldots, J$ before trimming is given by

$$ \text{len}({}^{n}\boldsymbol{c}_{\text{ext}}^{(j)}) = {}^{n}N_{\text{ext}}^{(j)} = \left\lfloor \text{len}({}^{n}\boldsymbol{x}_{\text{ext}})2^{-j} + (2^{-j} - 1)(m - 1) \right\rfloor, \tag{2.7} $$

where $\text{len}({}^{n}\boldsymbol{x}_{\text{ext}}) = {}^{n}L_{\max} + s + {}^{n}R_{\min}$, and m denotes the length of the wavelet filters. However, first

$$ N_{\text{disc}}^{(j)} = r(J - j) \tag{2.8} $$

coefficients at each level $j < J$ are calculated redundantly and they are discarded according to the algorithm description. In addition, the same number of coefficients of the last segment are discarded from the right. Therefore, the number of coefficients after discarding the redundant ones is

$$ {}^{n}N_{\text{coef}}^{(j)} = {}^{n}N_{\text{ext}}^{(j)} - {}^{n}N_{\text{disc}}^{(j)}, \tag{2.9} $$

except for the last segment which will have

$$ {}^{N-1}N_{\text{coef}}^{(j)} = {}^{N-1}N_{\text{ext}}^{(j)} - 2 \cdot {}^{N-1}N_{\text{disc}}^{(j)}, \tag{2.10} $$

coefficients remaining.

In the real-time setting, the algorithm delay is

- $r(J)$ samples if $(s \bmod 2^J) = 0$ and therefore ${}^{n}R_{\min} = 0$ for each n,
- $s + r(J)$ samples in all other cases, for the following segment have to be waited for.

Another remark from [10] regards the fact that the extensions are periodic with respect to the segment number.

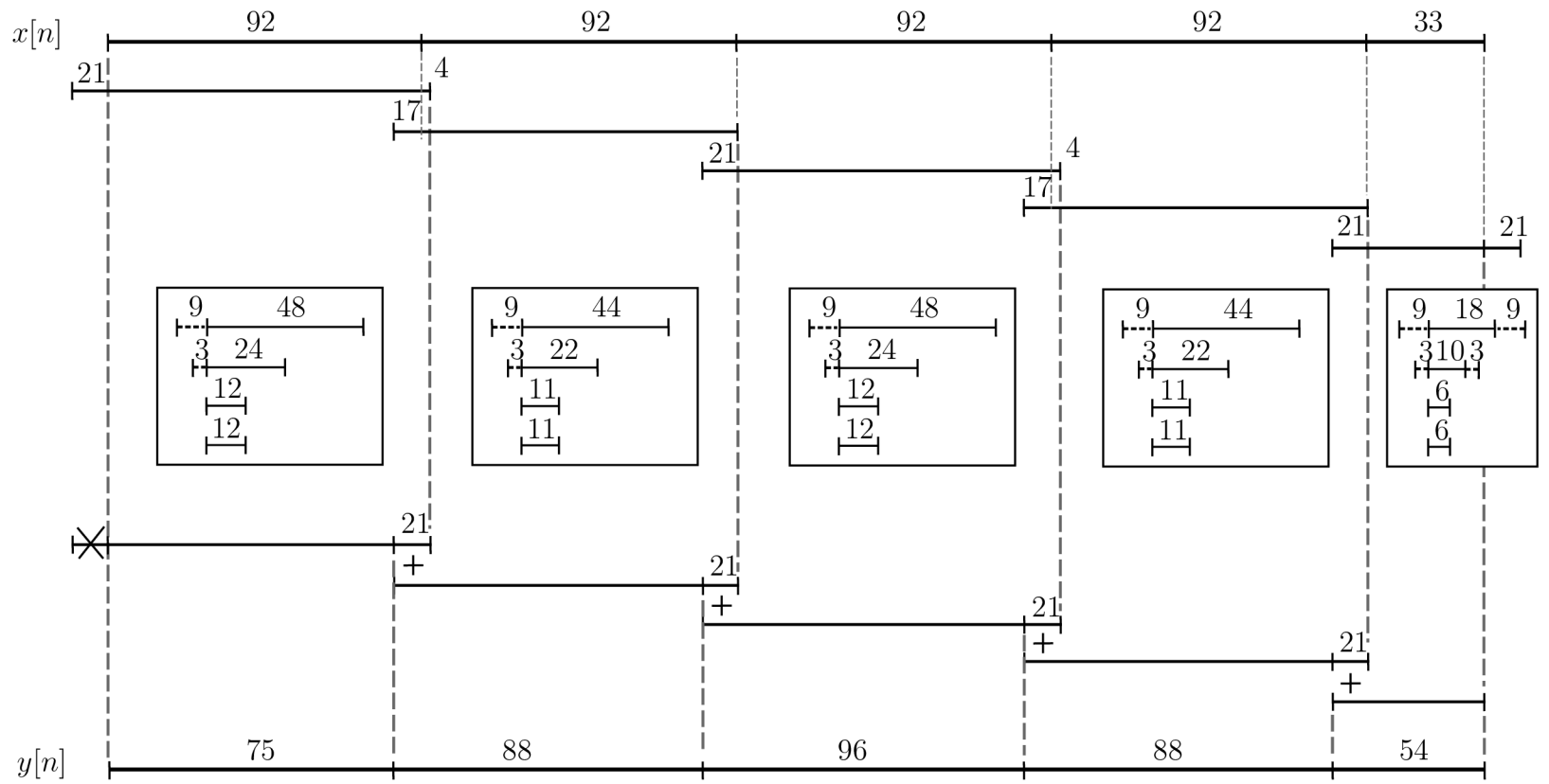


Figure 2.8: SegDWT algorithm demonstration example. The input signal $x[n]$ of length 401 is divided into 4 segments of length 92 and one of length 33, therefore $n = 0, \ldots, 3$. $J = 3$, $m = 4$ (e.g. `db2`) which, according to (2.2), leads to $r(3) = (2^3 - 1)(4 - 1) = 21$. Individual segments are extended from neighbors according to (2.3) and (2.4) e.g. ${}^0R_{\min} = 2^3 \left\lceil \frac{92}{2^3} \right\rceil - 92 = 4$ and ${}^1L_{\max} = 21 - 4 = 17$. Using modified DWT on the extended segments, the wavelet coefficients are obtained (in rectangular boxes), from which the initial $r(J - j)$ redundant samples are discarded (this only applies to the detail wavelet coefficients at level $j < J$ since $r(0) = 0$). At this point, wavelet coefficients can be processed in any way as they are identical to the whole signal wavelet transform. Prior to the inverse transform, the previously discarded samples are appended back but as zero samples. After the inverse DWT, the last $r(J)$ samples of each segment form overlap.

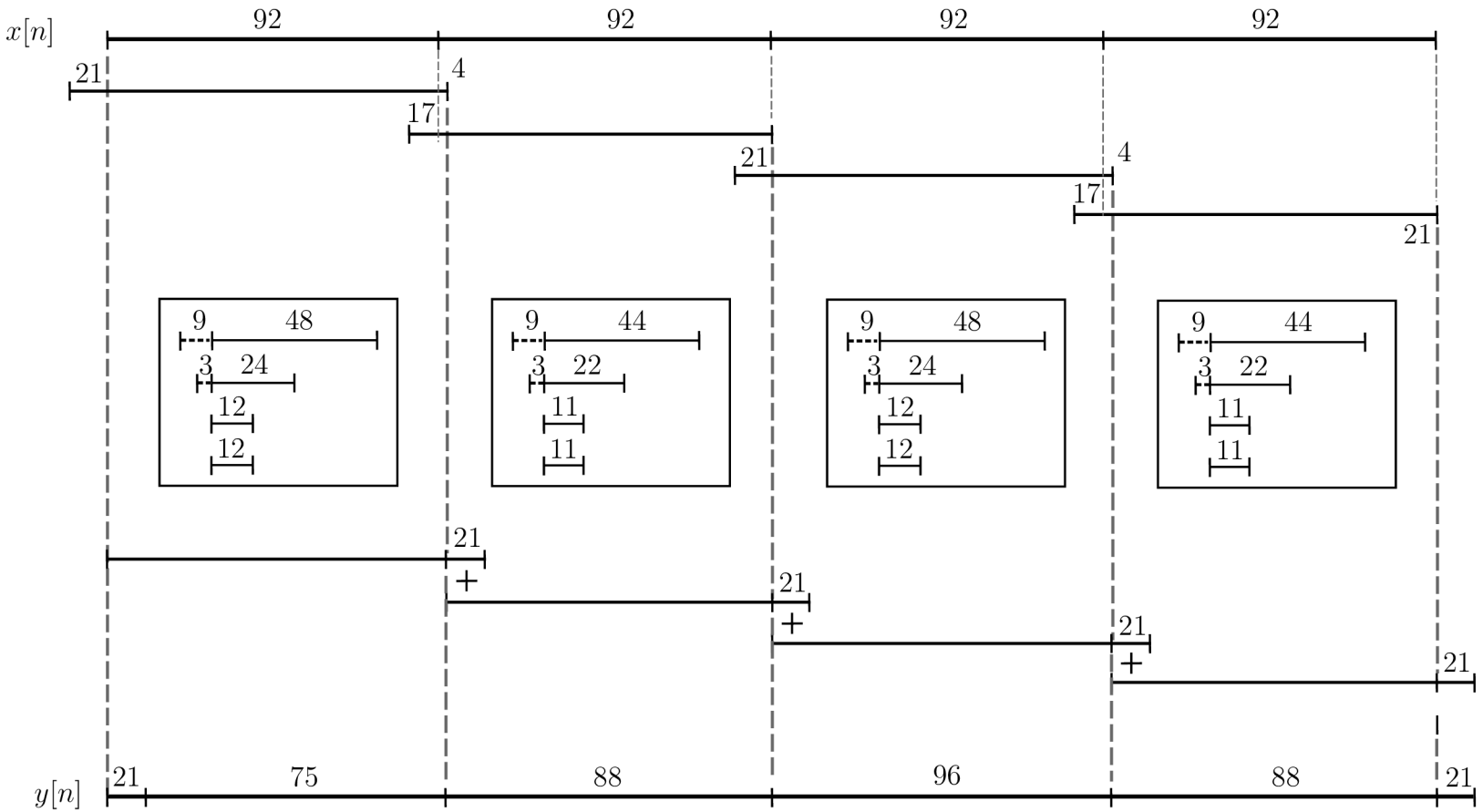


Figure 2.9: SegDWT algorithm example in the real-time setup. The input signal $x[n]$ is processed by segments of length $s = 92$. The length of the wavelet filters is $m = 4$ and the depth of decomposition is $J = 3$. This setup leads to $r(J) = 21$. Note that the reconstructed signal is delayed by the $r(J)$ samples; the first $r(J)$ samples of the reconstructed signal can be viewed as the "reconstruction warmup" and should be set to zero.

3 THESIS OBJECTIVES

The advantages of the segmentwise computation of DWT were discussed in chapter 2. Algorithms for such computations can already be found in the recent literature, however, except for one, they were developed for a concrete application or/and are restricted to one type of wavelet filter and most of them lack perfect recovery. The special case (SegDWT in [10]) was formulated more universally but just for the one-dimensional case and with the assumption of equal segment lengths. Direct usage of the algorithm for multidimensional signals seems to be too restricting to be usable in practice. Thus the first objective of this thesis deals with modifications of the original algorithm.

The following list summarizes the main drawbacks and restrictions of the original algorithm design and proposes modifications to achieve maximal generality:

- *The left extension ${}^{n}L_{\max}$ is chosen to be as high as possible to maximize re-usage of the received samples* due to the original purpose of the algorithm for real time processing of the acoustic signals. The extension lengths can clearly be stated more universally, therefore formulas for the other extreme ${}^{n}L_{\min}$ and all intermediate values will be derived.
- *The algorithm considers only segments of equal size.* This is inconvenient because it allows only square (cube) segments in multidimensional signals and prevents a dynamic splitting of segments. It will be shown that the lengths of both the right and the left extensions of the n and $(n+1)$ segment, respectively, depend only on the position of their dividing line which in turn allows an arbitrary rectangular (box) segment shape for multidimensional signals.
- *The extensions are unnecessarily long.* A more detailed analysis of the SegDWT algorithm reveals the fact that the even type of subsampling indirectly increases lengths of extensions. It will be shown that a small modification can save up to $2^J - 1$ samples of extensions. The number of the saved samples increases even more with increasing number of signal dimensions.
- *Another restricting factor is the need of the right extensions itself.* The algorithm analysis shows that the minimum right extension ${}^{n}R_{\min}$ is employed just to align the right border of the segment to a multiple of 2^J, thus ${}^{n}R_{\min} = 0$ when the dividing line index is a multiple of 2^J. This restriction can be also lifted by encompassing the (nonzero) right extension to the left one provided there is another modification of the algorithm. This is clearly beneficial when causality is a need (e.g. audio, video signals). There is a workaround proposed in the original algorithm, but it increases the processing delay by a whole segment duration.

Having the generally stated SegDWT algorithm, the next goal is to tailor it to

the concrete usage exploiting prior information about the processed signal while optimizing some parameters of the algorithm. Since the original algorithm employs a overlap-save for the analysis and a overlap-add for the synthesis, the new versions of the SegDWT are:

- *Overlap-save SegDWT analysis with overlaps in wavelet domain* In case of consecutive order of segments, the memory requirements can be reduced using overlaps directly in the wavelet domain (approx. coefficients at levels $j = 0, \ldots, J-1$).
- *Overlap-Add SegDWT analysis and Overlap-Save SegDWT synthesis* In some situations, it can be beneficial to use complementary methods i.e. Overlap-Add for analysis and Overlap-Save for synthesis. Especially where a parallel processing of more segments is concerned, the overlap-add approach creates a so-called "race condition" [48] i.e. two parallel writes to one memory location can overwrite each other and result in errors.
- *Region of Interest wavelet coefficient processing* Combining Overlap-Save type of SegDWT for both analysis and synthesis brings the possibility of processing arbitrary segment truly independently in a sense that the current segment samples are fully reconstructed in opposition to the incomplete reconstruction of the last $r(J)$ samples when using OLA type SegDWT for synthesis, provided the equality of wavelet coefficients with the appropriate parts of the whole signal wavelet transform.

All the proposed modifications are presented in chapter 4.

Bearing the proposed modifications in mind, the second objective of the thesis, the multidimensional extensions via separability property, are relatively simple and can be stated universally for arbitrary dimension number which is done in chapter 6.

The lifting scheme forms an alternative to the wavelet transform computation and can also be conducted segmentwise. Since the lifting scheme is more complex than the plain two channel filter bank, the segmentwise algorithm for LWT is not as straightforward as in SegDWT case. Therefore, the chapter 5 describes the development of several algorithms, which, in the end, produces desired left end right extensions.

The last objective of the thesis is to verify the proposed algorithms in real-life applications.

4 SEGMENTED DISCRETE WAVELET TRANSFORM

This chapter[1] contains all the modifications introduced in chapter 3. The purpose of these modifications is to increase generality of the algorithm, since the original algorithm is not directly usable for multidimensional signals. The desired properties are: an arbitrary order of segments to be processed, the independence of calculations so they can be carried out in parallel, custom extension lengths manipulations and an effective exploitation of 2^J-shift invariance.

Section 4.1 builds algorithm with the maximally general properties. The generality comes at a cost of slightly more complicated formulas for the segment extensions lengths and there can be some redundant computations while sections 4.2 and 4.3 present modifications that lead to optimization in some sense while sacrificing other properties.

All further presented modifications were implemented in Matlab and the codes can be found on the accompanied DVD and on the SegDWT algorithm webpage [49].

4.1 SegDWT Analysis and Proposed Extensions

Prior to the description of the modifications, a detailed analysis of the original algorithm is needed. The following text follows section 2.1 and discusses details and remarks not yet described. First, the input samples and the wavelet coefficients

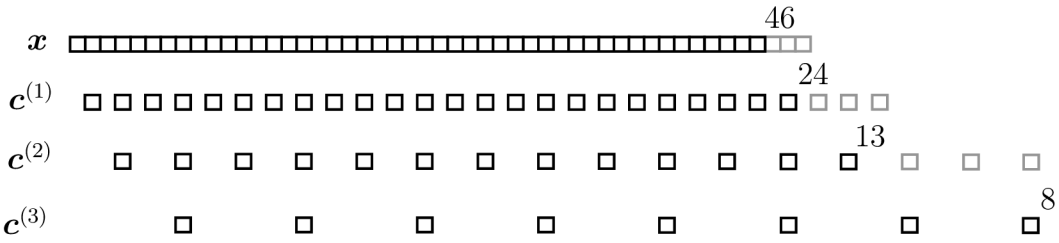


Figure 4.1: The input samples and the wavelet coefficients alignment of the input signal of length 46 using a wavelet with filter lengths $m = 4$ and depth of decomposition $J = 3$. Gray coefficients denote "range" of the impulse response during the linear convolution.

alignment of the whole input signal $\boldsymbol{x}$ need to be established. The even downsampling and the expansivity property result in the coefficient alignment shown in

[1]The research in this chapter was conducted jointly with Mgr. Pavel Rajmic, Ph.D. Publications related this this chapter are [1–3].

fig. 4.1.

The number of coefficients $N_{\text{coef}}^{(j)}$ at level j of the input signal of length s when using filters of length m can be easily derived recursively, using a number of coefficients at the previous level. In [2], we derived a non-recursive formula

$$N_{\text{coef}}^{(j)} = \left\lfloor 2^{-j}s + (1 - 2^{-j})(m-1) \right\rfloor . \tag{4.1}$$

There are both left and right extension of the segments employed in the SegDWT algorithm (recall (2.3) for ${}^{n}R_{\min}$ and (2.4) ${}^{n+1}L_{\max}$).

The purpose of the right extension is to align the end of each segment to be integer multiple of 2^J, which results in the correct alignment of vectors of wavelet coefficients and to the unification of all consecutive calculations.

The purpose of the left extension is to provide enough samples from the preceding segment(s) to "fully" (meaning as if the whole input signal was available) calculate the wavelet coefficients at the topmost level of decomposition. Together, both extension provide $r(J)$ (from (2.2)) samples needed for the first coefficient at the topmost decomposition level in the current segment to be calculated fully.

It is clear that like this the lengths of the extensions can vary from segment to segment, and that the respective lengths are thus *induced*, in contrast to the STFT-type classical windowing where the overlap lengths are *fixed*.

Forward SegDWT As it was stated, after the extension of the segment, the Mallat's algorithm (see sec. 1.2.1) is employed but without step 2a, *Extending the input vector*. Alternatively, it can be seen as using OLS type of convolution in each iteration of Mallat's algorithm, assuming influencing samples (see fig. 2.4) to be already provided by the means of the segment's left extension. Since the OLS convolution in addition does not calculate "fade-out", the outcome of the OLS convolution is shorter by $m-1$ samples (from the beginning) prior to the downsampling. After downsampling, the number of coefficients is equal to (2.7).

The detailed depiction of the forward SegDWT at the segment transition is in fig. 4.2. In the figure, ${}^{n+1}S$ denotes the index of the leftmost sample of the $n+1$ segment in a global point of view, prior to the extensions. Clearly, ${}^{n+1}S = (n+1)s$ assuming ${}^{0}S = 0$ and equal length of segments. This denotation will be more convenient in the rest of this chapter.

Inverse SegDWT The reconstructed segment length and position is equal to the length and position of the one analyzed after extensions.

Prior to the reconstruction, $r(J-j)$ zero coefficients are appended to the beginning of the coefficient vector at level j (see fig. 4.3). In contrast to the forward

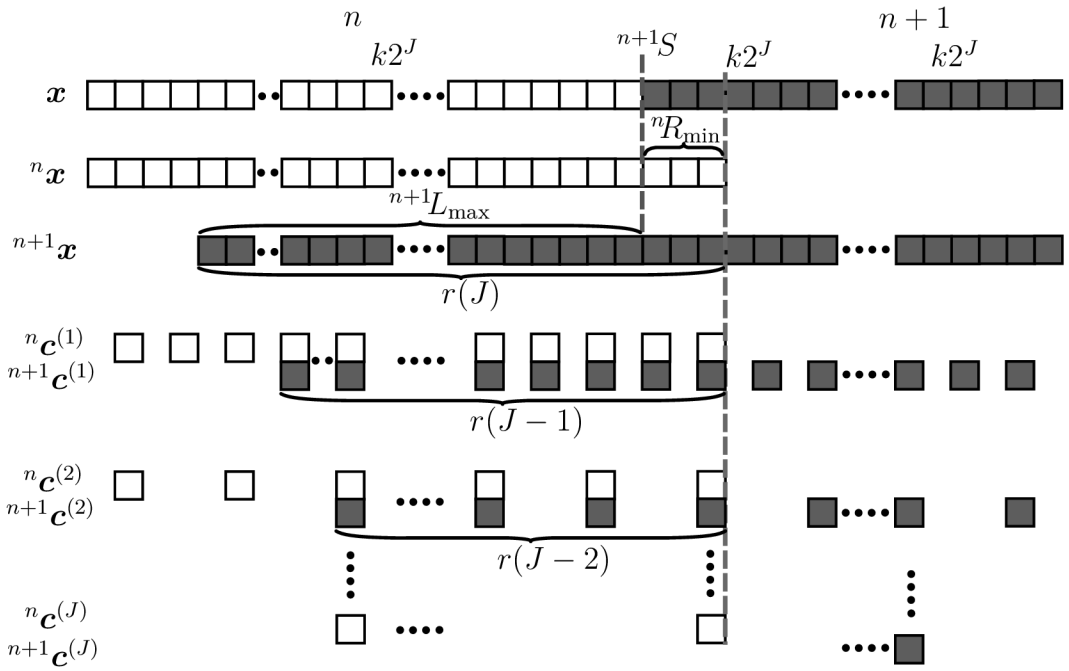


Figure 4.2: Transition between two consecutive segments n and $n+1$ at index $^{n+1}S = (n+1)s$ in the time-domain signal $\boldsymbol{x}$. The segments are extended by $^{n}R_{\min}$ and $^{n+1}L_{\max}$ samples accordingly. The wavelet coefficients $\boldsymbol{c}^{(j)}$ belonging to respective segments are shown. The coefficients belonging to the segment $(n+1)$ lying to the left from the dividing line marked as $r(J-1), r(J-2), \ldots$ are calculated redundantly and they ought to be discarded after the computations are complete. Also the wavelet coefficients of the n segment are aligned to the dividing line due to the right extension.

SegDWT, the non-shortened (linear or OLA) version of convolution is employed after the even upsampling of the coefficient vector. This means that the length of the intermediate reconstruction vector grows between steps of the reconstruction and results in the overlap of $r(J)$ samples reaching to the neighboring segment.

Another remark, considering the inverse transform, regards the fact that after the reconstruction of the whole signal, there are additional and redundant $r(J)$ samples at the beginning. Additional samples at the beginning bring ambiguity to the indexing of the reconstructed signal which can be viewed in two ways:

- The reconstruction is delayed by $r(J)$ samples. Hence the indexing of the input signal and the reconstruction does not match and also last the $r(J)$ samples of each reconstructed segment form overlap to the next one. This view is natural in the real-time setup.
- The first $r(J)$ samples of the reconstruction are not included in the indexing. The indexing of both the input signal and the reconstruction matches and this

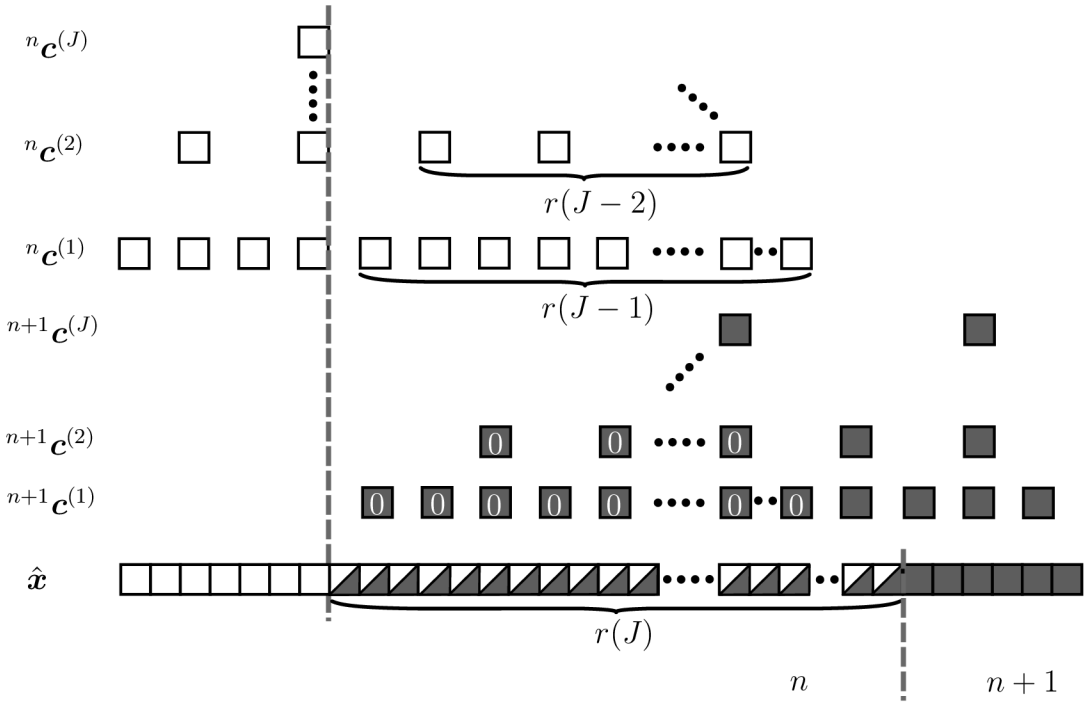


Figure 4.3: The inverse SegDWT in detail. The initial zero padding of the detail coefficients at levels $j = 1, \ldots, J-1$ causes a relative shift of individual coefficient vectors. This padding in turn causes the proclaimed delay of the reconstruction by $r(J)$ samples.

time, on the contrary, the first $r(J)$ samples of each segment overlap to the previous segment.

The second view will be considered in the following text to prevent an ambiguity. When necessary, the value according to the first view will be placed in brackets.

Examples of both the forward and the inverse SegDWT are shown at fig. B.6a and fig. B.6c respectively.

Noble multirate identity and the SegDWT algorithm There is a nice connection between the noble multirate identity representation of the iterated filter bank (see sec. 1.2.2) and the SegDWT remarks. By supplying $j = J$ to (1.23) one can get length of the filter identical to the longest branch of the iterated filter bank:

$$\operatorname{len}(\boldsymbol{f}_1 * \cdots * \boldsymbol{f}_J) = (2^J - 1)m - (2^J - 2). \tag{4.2}$$

After a simple manipulation, the formula changes to

$$(2^J - 1)(m - 1) + 1, \tag{4.3}$$

and thus the necessary filter overlap is exactly equal to the $r(J)$ from (2.2).

4.1.1 Enstensions tradeof

As was mentioned, the original algorithm allows constant-length segments only and uses minimum right and maximum left extensions of two consecutive segments. Other possibilities were not derived in the original work and therefore this section completes them. Following ideas and the mathematic style of [10], the following notions are built upon the Theorem 8.11 (from [10]) which can be re-written as:
Theorem 8: *Let the segment n is given, whose length including its left extension is ${}^{n}l$. Then the left extension of the next segment ${}^{n+1}L$ can be computed by the formula:*

$$ {}^{n+1}L = {}^{n}l - 2^J i \quad \text{for integer } i \in \left[\frac{{}^{n}l - r(J)}{2^J}, \frac{{}^{n}l}{2^J} \right]. \tag{4.4} $$

And for the right extension of n-th segment the following holds:

$$ {}^{n}R = r(J) - {}^{n+1}L. \tag{4.5} $$

The maximum left extension ${}^{n+1}L_{\max}$ is naturally reached for the lowest $i = \left\lceil \frac{{}^{n}l - r(J)}{2^J} \right\rceil$ which is also done in the original algorithm, and the minimum left extension ${}^{n+1}L_{\min}$ can be obtained when taking the other extreme, the highest $i = \left\lfloor \frac{{}^{n}l}{2^J} \right\rfloor$. $R_{\min}$ (for $L_{\max}$) is already known (2.3) and ${}^{n}R_{\max}$ (for ${}^{n+1}L_{\min}$) can be written as

$$ {}^{n}R_{\max} = 2^J \left\lfloor \frac{(n+1)s + r(J)}{2^J} \right\rfloor - ns, \tag{4.6} $$

(The proof is the same as in Theorem 8.14 in [10]). We can rewrite formula (4.4) as

$$ {}^{n+1}L = {}^{n}l - 2^J \left(\left\lceil \frac{{}^{n}l - r(J)}{2^J} \right\rceil + k \right) \text{ where } k \in \mathbb{N}^0, \tag{4.7} $$

satisfying ${}^{n+1}L \geq {}^{n+1}L_{\min}$ at the same time. Having compared formulas (2.3), (4.6) and (4.7), we can write the right extension of segment n as:

$$ {}^{n}R = 2^J \left\lceil \frac{(n+1)s}{2^J} \right\rceil - (n+1)s + 2^J k, \quad \text{where } k \in \mathbb{N}^0, \tag{4.8} $$

or alternatively as

$$ {}^{n}R = {}^{n}R_{\min} + 2^J k \quad \text{where } k \in \mathbb{N}^0, \tag{4.9} $$

satisfying ${}^{n}R \leq {}^{n}R_{\max}$ at the same time. Using this formula and (4.5),(2.2) one can "trade-off" the multiples of 2^J samples between extensions up to defined values ${}^{n}R_{\max}$ and ${}^{n+1}L_{\min}$. Note that the resulting coefficients after wavelet analysis are traded too: it makes $k2^{J-j}$ coefficients at level j and the formula (2.7) includes them.

Examples of the modification for both forward and inverse SegDWT are shown in fig. B.6b and fig. B.6d respectively.

4.1.2 Segments of different sizes

This modification appears to be the most important one. It lifts the constant segment length constriction which impacts the algorithm significantly. Namely:

- The extensions depend on ${}^{n}S$ (index of segment's first sample from the global point of view) rather than on segment number and segment length (for example ${}^{n}L \to {}^{{}^{n}S}L$ but ${}^{n}R \to {}^{{}^{n+1}S}R$ and $r(J) = {}^{{}^{n}S}L + {}^{{}^{n}S}R$)
- The extensions become 2^J-periodic with respect to a sequence of ${}^{n}S$ increasing by one.
- Dynamic splitting of segments is allowed.
- The segments do not have to be processed in the consecutive order.
- The algorithm can be further developed just for the general case of the two consecutive segments.
- Each transition between segments can be treated individually.

The following theorem describes the presented modification.

Theorem 9: *The right extension of the segment n ($n = 0, 1, 2, \ldots, N-2$) and the left extension of the segment $(n+1)$ are given by the length of the portion of the signal starting at the beginning and ending at the end of the segment n (the number of already processed samples or the index of the segment's first sample from the global point of view) ${}^{n}S$, and the following holds*

$$ {}^{{}^{n}S}R = 2^J \left\lceil \frac{{}^{n}S}{2^J} \right\rceil - {}^{n}S + k2^J \text{ for } k \in \mathbb{N}^0, \text{ while } {}^{{}^{n}S}R \leq {}^{{}^{n}S}R_{\max}, \tag{4.10} $$

at the same time, or alternatively

$$ {}^{{}^{n}S}R = {}^{{}^{n}S}R_{\min} + k2^J \text{ for } k \in \mathbb{N}^0, \text{ while } {}^{{}^{n}S}R \leq {}^{{}^{n}S}R_{\max} \tag{4.11} $$

The extensions are not dependent on the number of previous segments neither on their lengths. The proof A can be found in appendix A.

This result is graphically shown in Fig 4.4.

4.1.3 Extension length reduction

As it was stated previously, globally the even type of up-/down-sampling is considered. In practice, in the dyadic case, it means discarding every first sample after every convolution. This is a legitimate operation for the wavelet transform of the whole signal, since it is a commonly accepted convention. In a segmentwise case, the left extensions have to be long enough to allow discarding first sample in each level after convolution which is clearly a waste of computational resources. Switching from the even to the odd type of up-/down-sampling, the left extension is reduced by $2^J - 1$ samples. The change is done just for the segmentwise computation purposes and the even type of up-/down-sampling is preserved globally. Since right

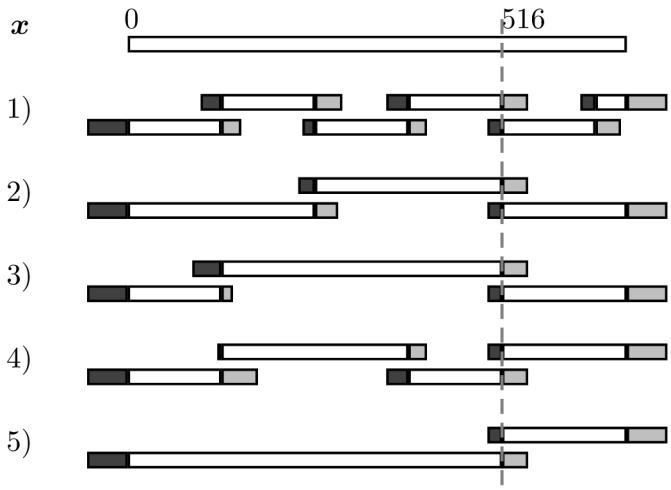


Figure 4.4: Figure presenting Theorem 9. Five cases of division of the input signal $\boldsymbol{x}$ are shown. There is always a division of a pair of segments between samples 515 and 516, but the divisions of the preceding part differs from case to case. Nevertheless, the lengths of the related extensions of the neighboring segments are equal in all cases.

extensions are not affected by this change, the left extension reduction cuts $r(J)$ down to

$$r_{\mathrm{red}}(J) = (2^J - 1)(m-2). \tag{4.12}$$

The rest of the algorithm remains the same except for the $r(J)$ substituted with $r_{\mathrm{red}}(J)$ which exhibits in several places: in the number of discarded coefficients after forward transform $r_{\mathrm{red}}(J-j)$ (and in number of the zero coefficients that are appended back prior to the inverse transform) and thus the total number of the coefficients containing the redundant ones (from (2.7)) is

$${}^{n}N_{\mathrm{ext}}^{(j)} = \left\lfloor \mathrm{len}({}^{n}\boldsymbol{x}_{\mathrm{ext}})2^{-j} + (2^{-j}-1)(m-2) \right\rfloor. \tag{4.13}$$

The length of the segment overlap after the reconstruction is also equal to $r_{\mathrm{red}}(J)$. In fact, the real segment overlap (meaning the number of nonzero samples within the $r(J)$ ones at the beginning of the reconstructed segment) was always $r_{\mathrm{red}}(J)$ since the even upsampling adds a zero to the beginning at each level prior to the convolution, which is propagated through the iterations of the synthesis filter bank. There is but one exception to this rule: the right extension of the last segment remains $r(J)$ and therefore also the number of the discarded coefficients from the right remains $r(J-j)$.

Examples of the modification for both forward and inverse SegDWT are shown in fig. B.7a and fig. B.7c respectively.

4.1.4 Right extension removal

The original algorithm with its possible nonzero right extension(s) have two disadvantages

- It violates causality and makes it hard to be used directly in the real-time setup.
- The right extension aligning the right border to the next multiple of 2^J causes the reconstructed segment to be aligned as well. This means that the segment borders of the input signal and the reconstructed signal do not match.

Regarding the first drawback, the original algorithm employs a delay of the processing by whole next segment, but this additional delay could be unacceptable. Another approach could be the employment of the "negative" right extension in a sense that the right border of the segment would be aligned with the lesser multiple of 2^J and the remaining samples would be encompassed into the left extension of the following segment. However, these approaches are both "workarounds" and do not solve the second drawback.

Regardless to this, the idea of the "negative" extension is worth describing for the coefficient alignment is not impaired and the algorithm complexity does not increase. It is clear that the number of the remaining samples after aligning with the lesser multiple of 2^J is $({}^nS \bmod 2^J)$ and thus the right extension is a negative number

$${}^{nS}R_{\text{neg}} = -({}^nS \bmod 2^J), \tag{4.14}$$

which leads to the necessary left extension

$${}^{nS}L_{\text{neg}} = r_{\text{red}}(J) + ({}^nS \bmod 2^J). \tag{4.15}$$

The negative right extension is also a special case of the algorithm modification described in sec. 4.1.1, where $k = -1$ in (4.9).

To attack the second drawback, it is necessary to modify the algorithm so that no 2^J-alignment is needed and therefore neither is the right extension. In this setup, considerable modifications of the main formulas need to be done.

First, the formulas for $r(J)$ and ${}^{nS}L$ become the same (since the right extension is always zero)

$${}^{nS}L_{\text{noright}} = r_{\text{red}}(J) + ({}^nS \bmod 2^J) \tag{4.16}$$

If $({}^nS \bmod 2^J) = 0$, the right segment border is already aligned; the extension has to be increased accordingly if it is not. Note that the worst case is $({}^nS \bmod 2^J) = 2^J - 1$ which leads us back to ${}^{nS}L_{\text{noright}} = r(J)$.

Second, since the segment's right border is no longer aligned, the number of coefficients which belong to actual segment changes at each level of decomposition.

Let this value be denoted as ${}^{nS}N^{(j)}_{\text{coef}}$ (similarly as in (2.9)). It is derived using indexes of coefficients belonging to the given segment (starting with ${}^{n}S$, for $n = 0, \ldots, N-1$). The number of coefficients, or the index of the first coefficient belonging to the given segment starting with zero at level j in the segment starting at index ${}^{n}S = {}^{n}S^{(0)}$ is given by

$$ {}^{n}S^{(j)} = \left\lfloor \frac{{}^{n}S^{(0)}}{2^j} \right\rfloor, \tag{4.17} $$

but the last segment demands a different treatment due to the expansivity of the DWT. Let ${}^{N}S^{(0)} = \text{len}(\boldsymbol{x})$ denote a non-existing segment following the last one, than for other j using (4.1) we can write

$$ {}^{N}S^{(j)} = \left\lfloor \frac{{}^{N}S^{(j-1)} + m - 1}{2} \right\rfloor = \left\lfloor {}^{N}S 2^{-j} + (1 - 2^{-j})(m-1) \right\rfloor. \tag{4.18} $$

The number of coefficients is then

$$ {}^{nS}N^{(j)}_{\text{coef}} = {}^{n+1}S^{(j)} - {}^{n}S^{(j)} \tag{4.19} $$

in the segment starting with ${}^{n}S$. The number of the detail coefficients at level j that have to be discarded from the beginning of the coefficient vectors after forward transform is equal to

$$ {}^{nS}N^{(j)}_{\text{disc}} = r_{\text{red}}(J - j) + \left\lfloor \frac{({}^{n}S \bmod 2^J)}{2^j} \right\rfloor, \tag{4.20} $$

therefore prior to the coefficient discarding there are

$$ {}^{nS}N^{(j)}_{\text{ext}} = {}^{nS}N^{(j)}_{\text{disc}} + {}^{nS}N^{(j)}_{\text{coef}} \tag{4.21} $$

coefficients.

And, lastly, the calculated segment overlap after reconstruction is ${}^{nS}L_{\text{noright}}$.

Examples of modification for both the forward and the inverse SegDWT are shown in fig. B.7b and fig. B.7d respectively. An additional example of the algorithm modification usage at concrete signal is shown in fig. B.2 and B.3.

Segment length limitation It makes sense to define the minimum segment length $s_{\min}$ allowing the SegDWT algorithm to be carried out as it was described so far. It turns out, that the SegDWT algorithm needs the segment to contain at least one coefficient at the topmost level J, which limits the $s_{\min}$ to be

$$ s_{\min} \geq 2^J. \tag{4.22} $$

On one hand, this limitation can seem restricting, but on the other hand the necessary (and induced) extensions are approximately $m-1$ times longer than the minimal allowed segment length and the shorter the segments the higher the computational overhead.

New SegDWT formulation At this point, the algorithm is considered to be "maximally" general. Since the original algorithm description (alg. 6 and alg. 7) is somewhat obsolete now, the description of the new SegDWT algorithm follows. For simplicity, at least two segments are to be processed $N > 1$ and $s_{\min} \geq r(J)$ is assumed to limit the extensions to reach just to the directly adjacent segments. Also the reconstructed signal $\hat{\boldsymbol{x}}$ is delayed by $r(J)$ samples (for it is maximal possible ${}^{nS}L_{\text{noright}}$). The example of this setup is shown in fig. 4.6 (without the initial delay in fig. 4.5).

Algorithm 10:[SegDWT analysis v. 1.1 – no right extensions]
Let be given: $\boldsymbol{g}, \boldsymbol{h}$ of length m, J, input signal $\boldsymbol{x}$ divided into $N > 1$ segments ${}^{0}\boldsymbol{x}, {}^{1}\boldsymbol{x}, {}^{2}\boldsymbol{x}, \ldots, {}^{N-1}\boldsymbol{x}$. Zero padding is considered whenever segment extension reaches outside of the input signal $\boldsymbol{x}$ support.
One segment analysis:
For $n = 0, \ldots, N-1$ repeat:

1. Read segment ${}^{n}\boldsymbol{x}$ and extend it from left by ${}^{nS}L_{\text{noright}}$ samples from the previous segment.
2. **If** $n = N-1$, the current segment is the last one, extend it from the right by $r(J)$ zero samples.
3. Calculate the DWT of depth J from the extended current segment using the algorithm 4 omitting step 2(a) (OLS-type convolution with odd type downsampling).
4. Modify the vectors containing the wavelet coefficients by trimming off a certain number of redundant coefficients from the left side, specifically: at the level j, $j = 1, 2, \ldots, J-1$ trim off ${}^{nS}N_{\text{disc}}^{(j)}$ coefficients.
5. **If** $n = N-1$, trim off the vectors in the same manner as in the previous step but this time the number of trimmed coefficients is $r(J-j)$ and the trimming is performed from the right.
6. Store the result as ${}^{n}\boldsymbol{a}^{(J)}, {}^{n}\boldsymbol{d}^{(J)}, {}^{n}\boldsymbol{d}^{(J-1)}, \ldots, {}^{n}\boldsymbol{d}^{(1)}$.

Algorithm 11:[SegDWT synthesis v. 1.1 – no right extensions]
Let be given: wavelet reconstruction filters $\tilde{\boldsymbol{g}}$ and $\tilde{\boldsymbol{h}}$ of lengths m, J, ${}^{n}S$ for all segments ${}^{0}\hat{\boldsymbol{x}}, {}^{1}\hat{\boldsymbol{x}}, {}^{2}\hat{\boldsymbol{x}}, \ldots, {}^{N-1}\hat{\boldsymbol{x}}$ to be reconstructed. The reconstructed segments do not have to be equal to the analyzed ones if respective coefficients are available. For storing the overlap, a buffer of length $r(J)$ is used.
One segment synthesis:
For $n = 0, \ldots, N-1$ repeat:

1. Read respective coefficient vectors ${}^{n}\boldsymbol{a}^{(J)}, {}^{n}\boldsymbol{d}^{(J)}, {}^{n}\boldsymbol{d}^{(J-1)}, \ldots, {}^{n}\boldsymbol{d}^{(1)}$ according to the reconstructed segment ${}^{n}\hat{\boldsymbol{x}}$.

2. Extend the detail coefficients: at the level j, $j = 1, \ldots, J-1$, append ${}^{nS}N^{(j)}_{\text{disc}}$ zero coefficients from the left side.
3. Compute the inverse transform of depth J using Algorithm 5 omitting the cropping part (OLA-type convolution, odd upsampling).
4. **If** $n \neq 0$, add samples from the buffer to the current inverted segment:
   (a) Add last ${}^{nS}L_{\text{noright}}$ samples from the buffer to the first respective samples of the current segment.
   (b) Append remaining $r(J) - {}^{nS}L_{\text{noright}}$ first samples from the buffer from the left side of the current segment.
5. **If** $n \neq N-1$, store $r(J)$ last samples of ${}^{n}\hat{\boldsymbol{x}}$ in the buffer, append the remaining samples to the output.
   Else Append the whole inversion to the output.
6. The output (a segment of a time-domain signal) is now complete and prepared to be "sent".

4.2 Exploiting consecutive order of segments

One of the assumptions one can benefit from when seeking a way to optimize the SegDWT algorithm is the consecutive order of the segments (i.e. the real-time case). It can be safely presumed that both unprocessed and processed previous wavelet coefficients are available. The necessary overlaps for the SegDWT analysis can then be derived directly in wavelet domain and thus reducing the computational and mainly the memory complexity (from exponential to linear dependence on J). The SegDWT synthesis with overlaps in the wavelet domain does not bring such computational reduction and the overlap handling becomes complex enough not to be beneficial in the practical implementation.

4.2.1 SegDWT analysis with overlaps in wavelet domain

The algorithm clearly performs redundant computations whenever segments are analyzed (forward transformed) in the consecutive order i.e. in real-time setting. Even though it might not be apparent at the first glance, there are ${}^{nS}N^{(j)}_{\text{disc}}$ (4.20) foremost approximation coefficients at levels $j = 1, \ldots, J-1$ calculated redundantly for they were already calculated during the transformation of the previous segment. Since approximation coefficients of the previous segments can be buffered, the left extension of the actual segment can be carried out directly at each level of decomposition as if OLS convolution (fig. 2.5) was done at each level but the number of *influencing* samples may vary. The length of the extension of approximation coefficient at levels

$j = 0, \ldots, J-1$ is $m-1$ or $m-2$. The exact value depends on the position of the dividing line between segments (index ${}^{n}S$) so that

$$ {}^{n_S}L^{(j)}_{\text{noright}} = m - 2 + ({}^{n}S^{(j)} \bmod 2) = m - 2 + \left(\left\lfloor \frac{{}^{n}S}{2^j} \right\rfloor \bmod 2 \right), \tag{4.23} $$

while the OLS convolution and the odd type subsampling are preserved and no additional coefficient discarding is necessary except for the last segment, which is extended from the right side by $r(J)$ samples like in the original algorithm, and an appropriate coefficient discarding from the right side takes place.

Example of the algorithm is shown in fig. B.9.

4.3 Complementary methods and offline processing

So far, the modifications have not impaired causality (except for the possible 2^J samples exchange between extensions) and the overlap-save SegDWT analysis ensured that the received coefficients were fully calculated (i.e. no overlap-add type overlaps in the wavelet domain after the forward transform). Also the purpose of the algorithm so far was to perform the analysis, to allow possible coefficient modifications and then to perform the synthesis of the given segment (with some overlap in the time or the wavelet domain). The algorithm however can be also used for segmentwise wavelet analysis only when the output comprises of vectors of wavelet coefficients of the whole signal and, similarly, it can be used for the segmentwise synthesis, when a whole signal wavelet domain representation of a signal is available. Therefore the causality restriction can be removed but on the other hand, the requirement for the arbitrary order of the segment processing may rise up.

Regarding the off-line segmented processing, the idea of a single-level DWT at a time comes to a play. Since the entire signal (or the whole signal wavelet transform) is available, the single-level segmented DWT (or inverse DWT) can be done at a time and the division into segments at each level of decomposition can be treated individually. Conveniently, the necessary overlaps from sec. 4.2 can be reused.

The complementary methods, presented further, form alternatives in both real- and non-real-time cases. The methods are derived for arbitrary segment lengths (and dividing lines), but the formulas are greatly simplified, when 2^J segment border alignment is held, which can be easily done in the offline case.

4.3.1 Overlap-Add for SegDWT analysis

There are several changes when compared to the OLS analysis, which is employed in the original algorithm:

- Extensions to neighboring segments are no longer used and the OLA type of convolution is employed. However, when calculating coefficients at level $j \geq 1$ the downsampling type is chosen with respect to ${}^{n}S^{(j-1)}$. An odd downsampling is used when ${}^{n}S^{(j-1)}$ is odd and vice versa.
- After the wavelet analysis, the number of the wavelet coefficients at each level j produced this way can be calculated using formula (4.19) but taking the current segment as the last one ($n+1 = N$). This mean, that there are

$$ {}^{n_S}R^{(j)}_{\mathrm{OLA}} = \left\lfloor \frac{r(j) + ({}^{n}S \bmod 2^J)}{2^j} \right\rfloor - \left\lfloor \frac{({}^{n}S \bmod 2^J)}{2^j} \right\rfloor \tag{4.24} $$

  more coefficients than there would be if the current segment were not considered as last. These coefficients form overlaps to the respective vectors of coefficients belonging to the following segment. Again, the formula is greatly simplified when $({}^{n}S \bmod 2^J) = 0$:

$$ {}^{n_S}R^{(j)}_{\mathrm{OLA}} = \left\lfloor \frac{r(j)}{2^j} \right\rfloor . \tag{4.25} $$

  An example is shown in fig. B.8a.
  The values of these coefficients are not yet fully calculated and thus cannot be processed non-linearly and used for the inverse transform. Linear operations are allowed (multiplication, equal value coefficient shift) as long as as the machine precision is not an issue (the values of the tailing coefficients at the higher levels are calculated using the incomplete ones from the lower levels).
  On the other hand, carefully choosing and processing the complete wavelet coefficients is allowed as well as the inverse transform using just these coefficients. However, the idea will not be developed further because it can become cumbersome to deal with in practice especially when the overlap lengths (projected to the time domain) become comparable to segment lengths.

4.3.2 Overlap-Save for SegDWT synthesis

After the segment n reconstruction using the OLA synthesis (like in the original algorithm), there are ${}^{n+1S}L_{\mathrm{noright}}$ last samples, which are not fully calculated. The goal of OLS synthesis is to fully reconstruct all segment samples which were analyzed (prior to extension) with no overlap. Contrary to the convention this means that the coefficient vectors belonging to actual segment have to be extended from the *right*

side using coefficients belonging to the following segment and thus violating causality. The number of these coefficients is the same as in (4.24). No zero coefficients are appended and OLS type of convolution is used. Again, when calculating coefficients at level j, the upsampling type depends on the ${}^{n}S^{(j)}$ in the similar manner like in the previous section: the odd type upsampling is used when ${}^{n}S^{(j)}$ is odd and vice versa.

An example is shown in fig. B.8c.

4.4 Region of Interest SegDWT

In the following paragraphs, a new promising combination of the OLS methods for analysis and synthesis is discussed. Recall that the original algorithm uses overlap-save for analysis and overlap-add for synthesis. Because of using the OLS analysis and synthesis it is possible to process (analyze and reconstruct) an arbitrarily chosen segment while no overlap after reconstruction is needed.

This approach also answers the question stated in [50] asking which wavelet coefficients participate in exact reconstruction of the arbitrarily chosen rectangle ROI. The number of such affected coefficients is clearly higher than the OLS analysis SegDWT algorithm produces.

OLS analysis and OLS synthesis In this setup, the wavelet coefficients extension necessary for the synthesis OLS needs to be calculated as part of the analysis step. Naturally it requires an additional right extension of the analyzed segment in turn and therefore the causality is violated. The length of the right extension in the input samples is given by the number of coefficients in the topmost level (J). Supplying $j = J$ into (4.24) results in

$$ {}^{nS}R^{(J)}_{\mathrm{OLS}} = \left\lfloor \frac{r(J) + ({}^{n}S \bmod 2^{J})}{2^{J}} \right\rfloor, \tag{4.26} $$

which represents number of the additional coefficients at level J. Mapping them back to the right extension at level $j = 0$ result into

$$ {}^{nS}R^{(0)}_{\mathrm{OLS}} = \left\lfloor \frac{r(J) + ({}^{n}S \bmod 2^{J})}{2^{J}} \right\rfloor 2^{J} - ({}^{n}S \bmod 2^{J}). \tag{4.27} $$

Again, the formula is greatly simplified when $({}^{n}S \bmod 2^{J}) = 0$:

$$ {}^{nS}R^{(0)}_{\mathrm{OLS}} = \left\lfloor \frac{r(J)}{2^{J}} \right\rfloor 2^{J}. \tag{4.28} $$

The processed segment n is therefore extended from both sides by ${}^{nS}L_{\mathrm{noright}}$ samples from the left and by ${}^{n+1S}R_{\mathrm{OLS}}$ from the right.

The extended segment is then OLS-analyzed as the original algorithm is, but an additional right coefficient vector cropping is necessary. The number of the cropped coefficients is given by ${}^{n_S}N^{(j)}_{\text{coef}}$ from (4.19) (as if the segment was the last one) subtracted from the total number of the calculated coefficients after cropping them from the left side

$$ {}^{n_S}N^{(j)}_{\text{discright}} = {}^{n_S}N^{(j)}_{\text{extright}} - {}^{n_S}N^{(j)}_{\text{coef}} \tag{4.29} $$

where

$$ {}^{n_S}N^{(j)}_{\text{extright}} = \left\lfloor \frac{{}^{n}S + {}^{n_S} + {}^{n+1S}R^{(j)}_{\text{OLS}}}{2^j} \right\rfloor - {}^{n}S^{(j)}. \tag{4.30} $$

Knowing the lengths of the wavelet coefficient vectors, the analysis is done as described in sec. 4.1.4, the synthesis as described in sec. 4.3.2.

An example is shown in fig. B.8b and B.8d.

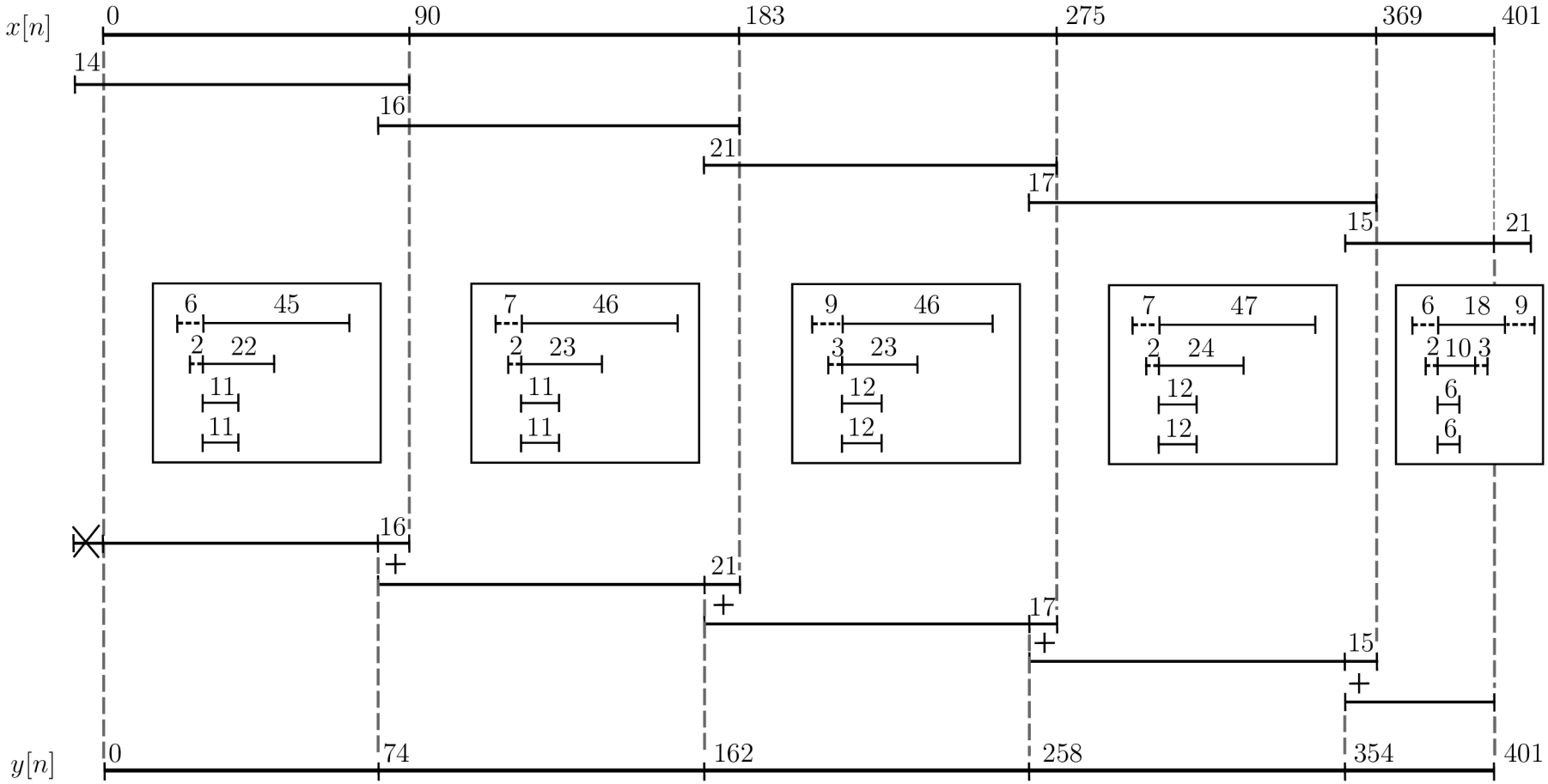


Figure 4.5: SegDWT algorithm modification demonstration example. The setup is like in the example in fig. 2.8 ($J = 3$, $m = 4$) but the lengths of the segments vary slightly. Modifications from section 4.1.4 regarding the removal of the right extension are considered. The segment extensions are calculated using eq. (4.16) and the number of discarded (and appended back as zeros during inverse transform) using eq. (4.20). Note the ends of the analyzed and the reconstructed segment is to be aligned.

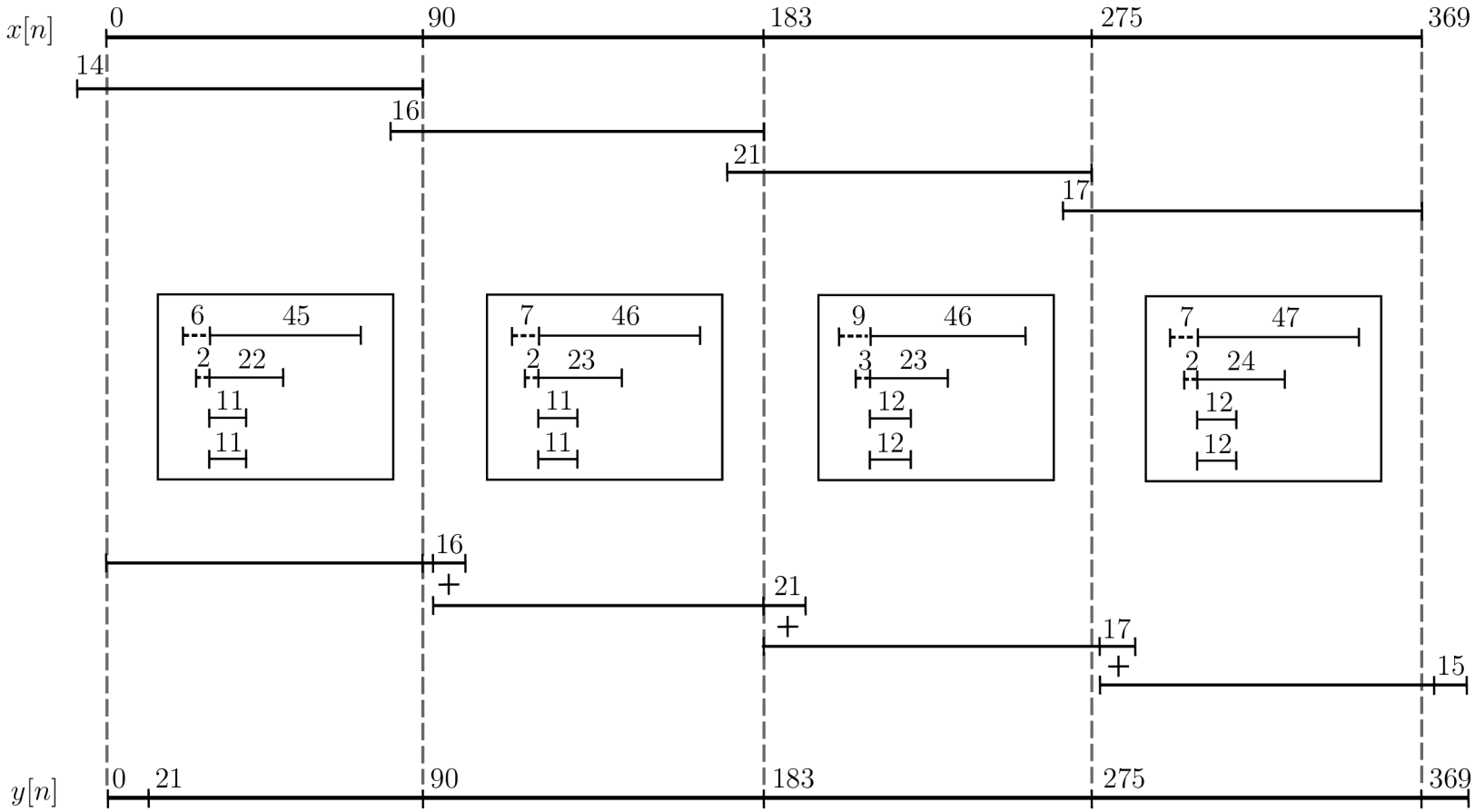


Figure 4.6: SegDWT algorithm modification example in the real-time setup. The setting and the extension lengths are the same as in fig. 4.5, but the reconstruction is delayed by $r(3) = 21$. This is equal to the longest possible segment overlap and therefore some reconstructed segments are shifted accordingly.

5 SEGMENTED LIFTING WAVELET TRANSFORM

In this chapter, a novel algorithm of a segmented computation of a LWT is proposed, called Segmented Lifting Wavelet transform – SegLWT[1]. The main idea is similar to the SegDWT algorithm. The forward transform:

1. Read a segment from the input, calculate its proper left and right extensions and read samples according to them.
2. Perform the LWT of the extended segment.
3. Crop the redundant samples in each level of decomposition from both sides.

Repeat these steps until all samples are processed. The fair generality of the algorithm lies in choice of the segment lengths, which are not restricted to the power of two and can be chosen arbitrarily (up to some minimal length) and can even differ from each other. The inverse transform is similar:

1. Read the corresponding sets of coefficients and extend them from both sides with zeros.
2. Perform the inverse LWT.
3. Place the segment to the correct position (within the output), add overlaps to the neighboring segments.

Again, repeat these steps until all samples are processed. The reconstructed signal does not suffer from the border distortion as it would when no overlaps were exploited.

The LWT differ from the DWT especially in the possibility of the calculations to be carried out in-place and in the number of elementary operations which is reduced by about a half. Also the lifting scheme provides a richer family of wavelets, since every wavelet filter bank can be transformed into the lifting scheme and in addition the invertible lifting transform can be designed directly as the lifting scheme does not have an equivalent wavelet filter bank representation.

Preliminaries Let us revise the notation: the $\boldsymbol{x}$ is input signal, the $\hat{\boldsymbol{x}}$ reconstructed signal, $\boldsymbol{a}^{(j)}, \boldsymbol{d}^{(j)}$ where $j = 0, 1, 2, \ldots, J$ are approximation (coarse) resp. detail (fine) coefficients and $\boldsymbol{a}^{(0)} = \boldsymbol{x}$. Symbol ${}^{n}\boldsymbol{x}$, $n = 0, 1, 2, \ldots$ denotes the segment n, ${}^{n}\boldsymbol{a}^{(j)}, {}^{n}\boldsymbol{d}^{(j)}$ denote approximation and detail coefficients of segment n accordingly. The length of the signal is given by $\mathrm{len}(\boldsymbol{x})$. Equivalently, every signal (or set of coefficients) can be understood as column vectors in $\mathbb{R}^N$, where N is the signal length. To address a particular sample (coefficient), square brackets are used, e.g. $x[n]$, $n = 0, 1, 2, \ldots$ or to choose set of samples $x[k]_{k \in \mathcal{I}}$. Clearly, $x[k] \neq {}^{n}x[k]$ for

[1] Publication related to this chapter is [4].

$n > 0$. ${}^nS$ denotes the first index of the segment n in the global indexing $x[{}^nS] = {}^nx[0]$ and ${}^nx[k] = x[{}^nS + k]$ for $k = 0, 1, 2, \ldots, \mathrm{len}({}^n\boldsymbol{x}) - 1$. Further, several operations are used $\lfloor r \rfloor, \lceil r \rceil$ to round $r \in \mathbb{R}$ to the nearest lower resp. greater integer, indicator functions $\mathrm{odd}(n), \mathrm{even}(n)$ return 1 when n is odd, even and 0 otherwise. Similarly, the indicator function $\mathrm{pow2}_p(n)$ returns 1 when $n = 2^p$, and 0 otherwise. The result of the indicator function is negated by bar, i.e. $\overline{\mathrm{odd}(n)} = \mathrm{even}(n)$. The indicator function applied to signals chooses samples at indexes with indicator function equal to 1 e.g. $\boldsymbol{y} = \mathrm{pow2}_p(\boldsymbol{x}) = x[k]_{k \in \mathbb{Z} \wedge \mathrm{pow2}_p(k)=1}$ and stack them together forming a new vector.

Lifting scheme is build upon splitting samples to odd and even ones, therefore $\boldsymbol{x}_\mathrm{E} = \mathrm{even}(\boldsymbol{x}) = x[2n]_{n \in \mathbb{Z}}$, $\boldsymbol{x}_\mathrm{O} = \mathrm{odd}(\boldsymbol{x}) = x[2n+1]_{n \in \mathbb{Z}}$, but to preserve the correct global division of samples even in segments, the starting index must be taken into account i.e. ${}^n\boldsymbol{x}_\mathrm{O} = \mathrm{odd}({}^n\boldsymbol{x}) = x[{}^nS + \overline{\mathrm{odd}({}^nS)} + 2n]_{n \in \mathbb{Z}}$ and ${}^n\boldsymbol{x}_\mathrm{E} = \mathrm{even}({}^n\boldsymbol{x}) = x[{}^nS + \overline{\mathrm{even}({}^nS)} + 2n]_{n \in \mathbb{Z}}$.

The lifting scheme [24, 31] maintains all the coefficients $\boldsymbol{a}^{(j)}, \boldsymbol{d}^{(j)},\ j > 0$ interleaved in vector $\boldsymbol{x}$ when computed in-place, so $\boldsymbol{x}$ also holds the intermediate results. After transforming the whole signal, the coefficients are interleaved in a way that $\boldsymbol{a}^{(j)} = \mathrm{pow2}_j(\boldsymbol{x})$ and $\boldsymbol{d}^{(j)} = \mathrm{pow2}_j(x[k + 2^{j-1}]_{k \in \mathbb{Z}})$. Given segment ${}^n\boldsymbol{x}$ with the first index ${}^nS$, it is crucial to know indexes of all interleaved ${}^n\boldsymbol{a}^{(j)}, {}^n\boldsymbol{d}^{(j)},\ j > 0$ from the global indexing point of view: ${}^n\boldsymbol{a}^{(j)} = \mathrm{pow2}_j(x[k + \left\lceil \frac{{}^nS}{2^j} \right\rceil 2^j]_{k \in \mathbb{Z}})$ and ${}^n\boldsymbol{d}^{(j)} = \mathrm{pow2}_j(x[k + \left\lceil \frac{{}^nS}{2^j} - \frac{1}{2} \right\rceil 2^j + 2^{j-1}]_{k \in \mathbb{Z}})$ as the ${}^nS^{(j)} = \left\lceil \frac{{}^nS}{2^j} \right\rceil$ denotes first approximation coefficient at j level indexing point of view (compare with DWT coefficient indexing alignment (4.17)).

The lifting scheme [24, 31] consists of four steps: split, predict, update and scale, see fig. 5.2 for one stage forward (left) and inverse (right) transform. The inverse transform part contains the same steps as the forward transform part in reverse order and with minus signs. Operators $\boldsymbol{P}, \boldsymbol{U}$ are known to be Laurent polynomials after the Z-transform which can have both positive and negative exponents. For the purposes of our algorithm, we regard $\boldsymbol{P}, \boldsymbol{U}$ as signals (row vectors) containing coefficients of Laurent polynomial supplemented with zeros for missing exponents between the lowest and the highest one. Accordingly we define $\mathrm{supp}(\boldsymbol{P})$ as a set of all exponents between the smallest and the greatest ones.

The lifting steps are the following:

1. **Split** – splits samples to odd and even ones
2. **Predict** – combines several even samples and adds the result to the actual odd sample. This can be described by

$$a_\mathrm{O}^{(j)}[n] \leftarrow a_\mathrm{O}^{(j)}[n] + \sum_{k \in \mathrm{supp}(\boldsymbol{P})} \boldsymbol{P}[k] a_\mathrm{E}^{(j)}[n+k]. \tag{5.1}$$

3. **Update** – combines several odd samples and adds the result to the actual even sample

$$a_{\mathrm{E}}^{(j)}[n] \leftarrow a_{\mathrm{E}}^{(j)}[n] + \sum_{k \in \mathrm{supp}(\boldsymbol{U})} \boldsymbol{U}[k] a_{\mathrm{O}}^{(j)}[n+k]. \tag{5.2}$$

4. **Scale** – scales each sample by a given factor K or $1/K$.

The number of the lifting update and predict steps depends on the actual factorization. Additional levels of decomposition can be achieved when using the scheme iteratively on the output $\boldsymbol{a}^{(j+1)}$. The maximal desired level is denoted as J as in DWT and since the number of samples halves with each iteration (rounded up), the maximal reasonable $J_{\max}$, for which there are at least two samples in $\boldsymbol{a}^{(J-1)}$ (to get at least one sample in each $\boldsymbol{a}^{(J)}$ and $\boldsymbol{d}^{(J)}$).

5.1 Supporting algorithms

5.1.1 Lifting scheme and neighboring segments

From now on, let us consider the neighboring segments n and $(n+1)$. Accordingly, the left extension of $(n+1)$ segment is denoted by ${}^{n+1}L$ and the right extension of segment n is ${}^{n}R$. These extensions depend on the level of decomposition J and on the type of the lifting scheme (lifting steps count, type, values). First, the extensions to compute a single level (from j to $j+1$) LWT need to be established. Since lifting schemes factorizations may differ in number of lifting steps and their Laurent polynomials, we chose an algorithmic approach:

Algorithm 12:*[Determination of extensions for one level analysis]*
Given: lifting scheme, segments n and $(n+1)$, index of the first approximation coefficient of $(n+1)$ segment at level j: ${}^{n+1}S^{(j)}$. To be determined: the right and the left extension from j to $(j+1)$ level ${}^{n}R^{(j)}$ and ${}^{n+1}L^{(j)}$ (for clarity, front superscripts will be omitted in the following text)

Set $R^{(j)} = 0$ and $L^{(j)} = 0$. For all the predict and the update lifting steps taken in the reverse order, repeat:

1. Determine the type ($\boldsymbol{U}$ or $\boldsymbol{P}$) and the maximal ($e_{\max}$) and the minimal ($e_{\min}$) exponent of the current step.
2. **If** the current step is predict $\boldsymbol{P}$:
   (a) **If** $e_{\min} \leq 0$ then $L^{(j)} \leftarrow L^{(j)} - 2e_{\min} + \mathrm{odd}(S^{(j)} - L^{(j)})$
   (b) **If** $e_{\max} > 0$ then $R^{(j)} \leftarrow R^{(j)} + 2e_{\max} - 2 + \mathrm{odd}(S^{(j)} - 1 + R^{(j)})$
3. **If** the current step is update $\boldsymbol{U}$:
   (a) **If** $e_{\min} < 0$ then $L^{(j)} \leftarrow L^{(j)} - 2e_{\min} - 1 - \mathrm{odd}(S^{(j)} - L^{(j)})$
   (b) **If** $e_{\max} \geq 0$ then $R^{(j)} \leftarrow R^{(j)} + 2e_{\max} + 1 - \mathrm{odd}(S^{(j)} - 1 + R^{(j)})$

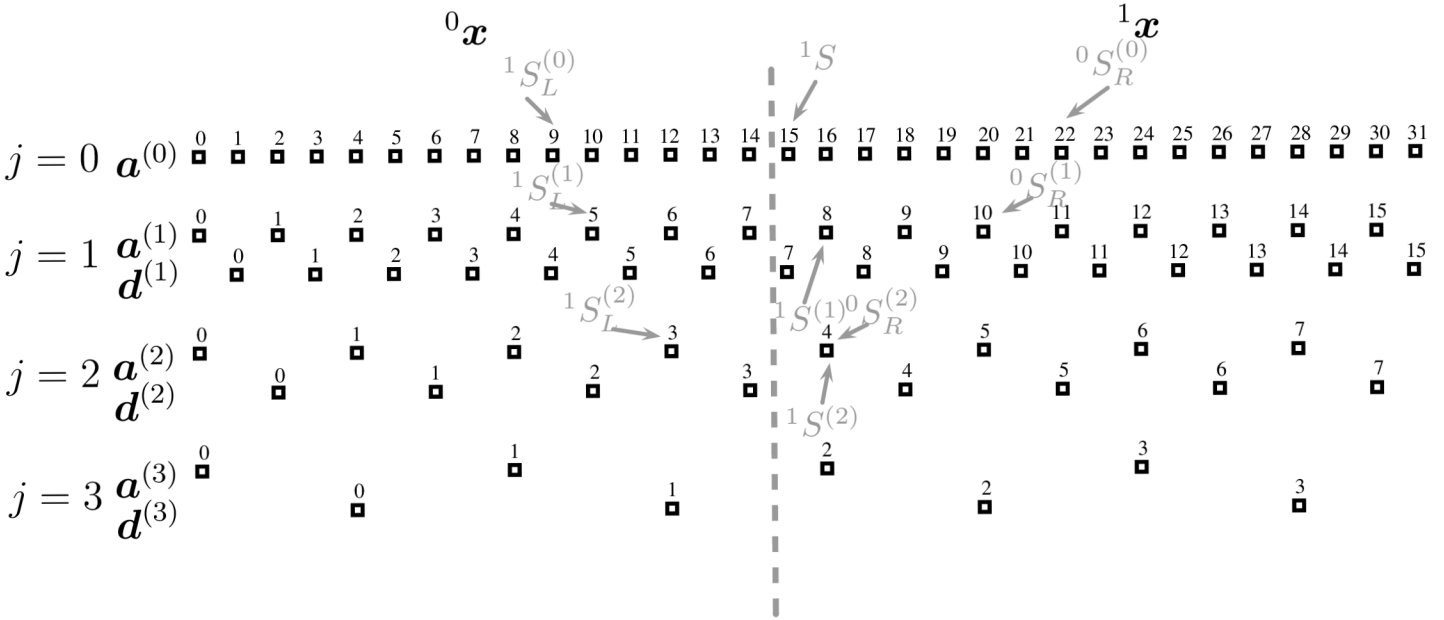


Figure 5.1: Example 1 graphically in detail.

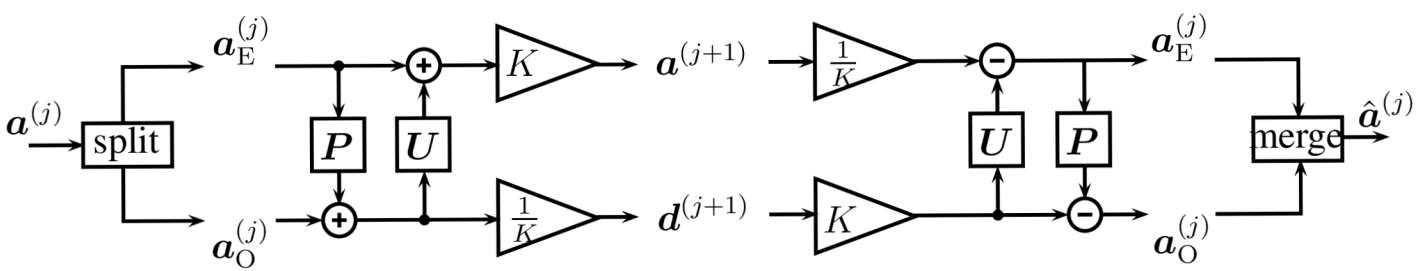


Figure 5.2: One stage of a lifting scheme. Left part: forward lifting, Right part: inverse lifting

Applying this algorithm iteratively as more levels of decomposition are needed, the algorithm for ${}^{n}R$ and ${}^{n+1}L$ can be stated:

Algorithm 13:*[Determination of extensions for J level analysis]*
Given: lifting scheme, segments n and $(n+1)$, index of the first sample of $(n+1)$ segment in the global indexing point of view : ${}^{n+1}S$. To be determined: the right and the left extension ${}^{n}R$ and ${}^{n+1}L$. In addition, let us denote the index of the first approximation coefficient at level j of segment $n+1$ after left extension ${}^{n+1}S_L^{(j)}$and similarly last approximation coefficient of segment n as ${}^{n}S_R^{(j)}$. Again, the front superscripts will be omitted in further description.

Set $S_R^{(J)} = 0$ and $S_L^{(J)} = \infty$. For all levels of the analysis j from $J-1$ up to 0, repeat:

1. Determine $\hat{S}_R^{(j)} = \max(S^{(j)} - 1, 2S_R^{(j+1)})$ and $\hat{S}_L^{(j)} = \min(S^{(j)}, 2S_L^{(j+1)})$.
2. Calculate $S_R^{(j)} = \hat{S}_R^{(j)} + R^{(j)}$ and $S_L^{(j)} = \hat{S}_L^{(j)} - L^{(j)}$ via utilizing Algorithm 12 taking $\hat{S}_L^{(j)}$ as the beginning of the $(n+1)$ for the left extension and $\hat{S}_R^{(j)} + 1$ as the beginning of the $(n+1)$ segment for the right extension.

Then ${}^{n}R = {}^{n}R^{(0)} - {}^{n+1}S + 1$ and ${}^{n+1}L = {}^{n+1}S - {}^{n+1}L^{(0)}$.

The left and the right extension can trade-off integer multiples of 2^J samples the same way as in SegDWT

$$ {}^{n}R \leftarrow {}^{n}R + k2^J, \tag{5.3} $$

$$ {}^{n+1}L \leftarrow {}^{n+1}L - k2^J, \tag{5.4} $$

as long as ${}^{n}R, {}^{n+1}L \geq 0$ holds for chosen integer k.

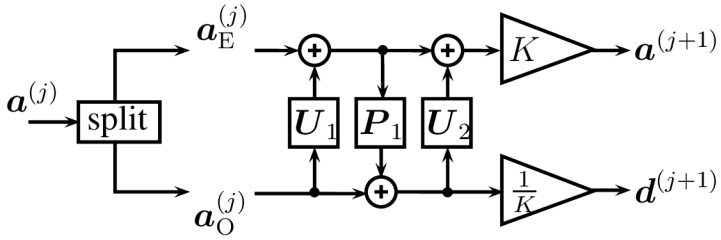


Figure 5.3: Lifting scheme for the wavelet `cdf3.1` with Laurent polynomials $\boldsymbol{U}_1 = -\frac{1}{3}z^{-1}$, $\boldsymbol{P}_1 = -\frac{3}{8}z - \frac{9}{8}$, $\boldsymbol{U}_2 = \frac{4}{9}$, $K \doteq 2.1213$

Example 14: Let us take lifting scheme for wavelet `cdf3.1` see fig. 5.3. The desired depth of the analysis is $J = 3$. The signal $\boldsymbol{x}$, $\mathrm{len}(\boldsymbol{x}) = 32$ is divided into two segments ${}^{0}\boldsymbol{x}$,${}^{1}\boldsymbol{x}$ with ${}^{0}S = 0$ and ${}^{1}S = 15$ and extensions ${}^{0}R$ and ${}^{1}L$ are to be evaluated.

According to Algorithm 13, we start with level 2 ($J - 1 = 2$):

1. $j = 2$: ${}^{1}S^{(2)} = \left\lceil \frac{{}^{1}S}{2^2} \right\rceil = 4$, ${}^{0}\hat{S}_R^{(2)} = 3$, ${}^{1}\hat{S}_L^{(2)} = 4$. Via Algorithm 12, ${}^{0}R^{(2)}$ and ${}^{1}L^{(2)}$ are enumerated:

(a) $\boldsymbol{U}_2$ is the last lifting step, $e_{\min} = e_{\max} = 0$. According to step 3b in Algorithm 12, ${}^0R^{(2)} = 0 + 0 + 1 - 1 = 0$.

(b) $\boldsymbol{P}_1$ is the second lifting step, $e_{\min} = 0, e_{\max} = 1$. According to steps 2a and 2b in Algorithm 12: ${}^1L^{(2)} = 0+0+0 = 0$ and ${}^0R^{(2)} = 0+2-2+1 = 1$.

(c) $\boldsymbol{U}_1$ is the first lifting step, $e_{\min}, e_{\max} = -1$. According to step 3a in Algorithm 12, ${}^1L^{(2)} = 0 + 2 - 1 - 0 = 1$.

So we have ${}^0R^{(2)} = 1, {}^1L^{(2)} = 1$ and ${}^0S_R^{(2)} = 4, {}^1S_L^{(2)} = 3$.

2. $j = 1$: ${}^1S^{(2)} = \left\lceil \frac{{}^1S}{2} \right\rceil = 8$, ${}^0\hat{S}_R^{(1)} = 8$, ${}^1\hat{S}_L^{(1)} = 6$. Extensions ${}^0R^{(1)}, {}^1L^{(1)}$ are defined similarly as in the previous step: ${}^0R^{(1)} = 2, {}^1L^{(1)} = 1$ and ${}^0S_R^{(1)} = 10, {}^1S_L^{(1)} = 5$.
3. $j = 0$: Similarly, ${}^1S^{(0)} = 15$, ${}^0\hat{S}_R^{(0)} = 20$, ${}^1\hat{S}_L^{(0)} = 10$, Extensions ${}^0R^{(0)} = 2, {}^1L^{(0)} = 1$ and ${}^0S_R^{(0)} = 22, {}^1S_L^{(0)} = 9$.

Extensions for `cdf3.1` are ${}^0R = 22 - 15 + 1 = 8$ and ${}^1L = 15 - 9 = 6$. The example is depicted in fig 5.1. Fig. 5.4 shows all (two) possible extensions parametrized by (5.3), (5.4).

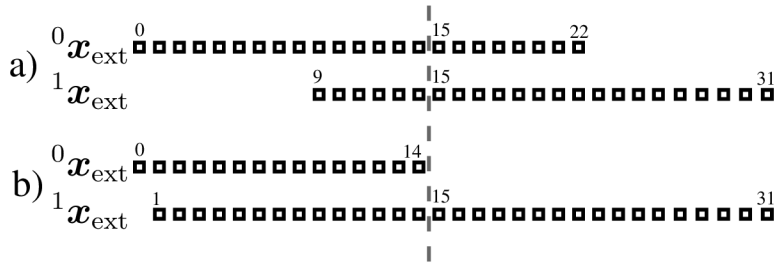


Figure 5.4: Possible extensions derived from Example 14. Equations (5.3), (5.4) were used with $k = 0$ (a) and $k = -1$ (b)

5.2 The Main Algorithm

This section contains a full description of the forward and the inverse segmented wavelet transform via the lifting scheme. When extending segments, one can slide of the support of the input signal. In this case, a zero padding is considered for the sake of simplicity, but the extension for other types of boundary treatment is straightforward.

Algorithm 15:*[SegLWT analysis of one segment]*
Given: segment n, index of the first sample in the segment in the global indexing point of view, ${}^nS$, and the last one $({}^{n+1}S - 1)$, desired level of decomposition J, lifting scheme, parameter k from parametrized extensions (5.3), (5.4) of neighboring segments already transformed. Output: $(J+1)$ interleaved sets of coefficients ${}^n\boldsymbol{a}^{(J)}$ and ${}^n\boldsymbol{d}^{(j)}$ for $j = 1, \ldots, J$.

1. Via Algorithm 13 and formulas (5.3), (5.4), determine left and right extensions of segment n: To get the proper left extension, apply Algorithm 13 to $(n-1)$ and n segments, in (5.4) use k from the previous segment if such exists, choose and store one in other case. To get the right extension apply Algorithm 13 to n and $(n+1)$ segments and in (5.3) use k from the next segment if such exists, otherwise choose and store some k.
2. Perform LWT of the extended segment in-place.
3. Crop the same amount of samples as was (or would be) added from both sides for $k = 0$ in (5.3), (5.4).
4. Optionally: Deinterleave coefficients to get sets ${}^{n}\boldsymbol{a}^{(J)}$ and ${}^{n}\boldsymbol{d}^{(j)}$.

Algorithm 16:*[SegLWT synthesis of one segment]*
Given: Interleaved sets of coefficients ${}^{n}\boldsymbol{a}^{(J)}$ and ${}^{n}\boldsymbol{d}^{(j)}$ for $j = 1, \ldots, J$, global index of the first sample in segment ${}^{n}S$, lifting scheme. Output: segment ${}^{n}\hat{\boldsymbol{x}}$, extended from both sides (containing overlaps to neighboring segments).

1. Extend interleaved coefficients from both sides with zeros as in Algorithm 15, though the extensions are computed differently now. Algorithm 13 is followed, but for several changes. The signs in step 2 are switched when computing ${}^{n}S_{R}^{(j)}$ and ${}^{n+1}S_{L}^{(j)}$, ${}^{n}S_{R}^{(J)}$ and ${}^{n+1}S_{L}^{(J)}$ switch values and also the resulting ${}^{n}R$ and ${}^{n+1}L$ switch values and have an opposing sign in addition.
2. Perform inverse LWT on the zero-extended segments.
3. Add the resulting segment ${}^{n}\hat{\boldsymbol{x}}$ of length $\mathrm{len}({}^{n}\hat{\boldsymbol{x}}) = {}^{n}L + \mathrm{len}({}^{n}\boldsymbol{x}) + {}^{n}R$ to the output signal starting at the global index ${}^{n}S_{L}^{(0)} = {}^{n}S - {}^{n}L$.

5.2.1 SegLWT algorithm in real-time

According to (5.3) the minimum right extension can be ${}^{n}R_{\min} = ({}^{n}R \bmod 2^{J}) \in \{0, \ldots, 2^{J}-1\}$. It depends on the value of ${}^{n}S$ but also on the concrete structure of the lifting scheme. In this section, a modification of the algorithm is proposed not requiring a right extension in a sense that none of the samples from the following segment are needed. This notion greatly simplifies the practical use of the proposed algorithm.

The idea is similar to the notion of the "negative" right extension from eq. (4.14). The situation is still the same (two neighboring segments n and $(n+1)$ as in section 5.1.1) but in addition, we are not allowed to use any samples from $(n+1)$ when working with segment n. Let us assume the ${}^{n}R_{\min} > 0$. The restriction for eq. (5.3), (5.4) $({}^{n}R, {}^{n+1}L) \geq 0$ can be overcome assuming a slight modification of the proposed algorithm. In the discussed case, we are interested in the case ${}^{n}R < 0$ which can be interpreted as that there are $2^{J} - {}^{n}R_{\min}$ samples at the end of the n segment, which

are not processed at the same time as segment n. These samples are encompassed in the left extension of the $(n+1)$ segment. This left extension is then given by

$$ {}^{n+1}L_{\text{noright}} = {}^{n+1}L + \left\lceil \frac{{}^{n}R}{2^J} \right\rceil 2^J \tag{5.5} $$

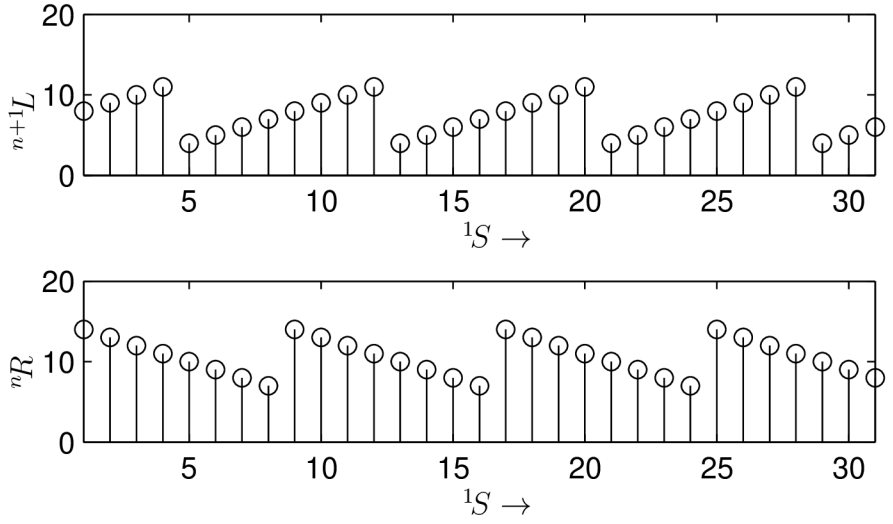
Example 17: Let us assume the same setup as in Example 1: ${}^1S = 15$, $J = 3$, ${}^0R = 8$ and thus ${}^0R_{\min} = ({}^0R \bmod 2^3) = 0$ see fig. 5.4 b). Extensions for every possible ${}^1S = 1, \ldots, 31$ are shown in the fig. 5.5. Let us leave aside the impracticality of the extending segments so that after the extension, one or both of them ale longer than the whole signal (clearly there are limitations). Periodicity $2^J = 8$ of the extensions are clearly visible, and ${}^0R_{\min} = 0$ for ${}^1S = p2^J - 1$ for some integer p, however the indexes for which the ${}^0R_{\min} = 0$ differ for different lifting schemes, only the periodicity remains.

After performing LWT of the actual segment n, ${}^nL$ samples are discarded from the left and ${}^nR$ from the right side. The resulting coefficients can be processed (e.g. thresholded) prior to the inverse LWT.

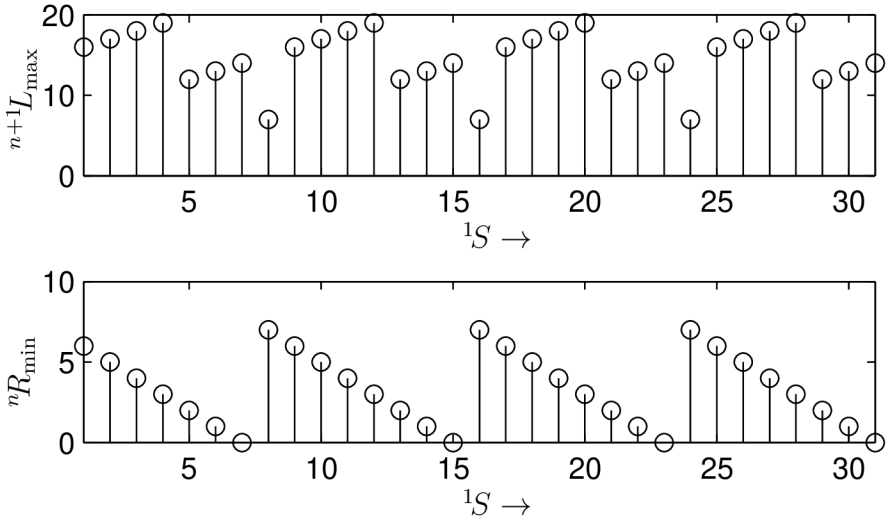
Inverse SegLWT in real-time setting After the coefficient cropping, the index ${}^nS$ actually changes (to ${}^nS_{\text{inv}}$). Also values ${}^nR_{\text{inv}}$ and ${}^{n+1}L_{\text{inv}}$ have to be recalculated using

$$ {}^{n+1}S_{\text{inv}} = {}^{n+1}S - \left\lceil \frac{{}^{n}R}{2^J} \right\rceil 2^J \tag{5.6} $$

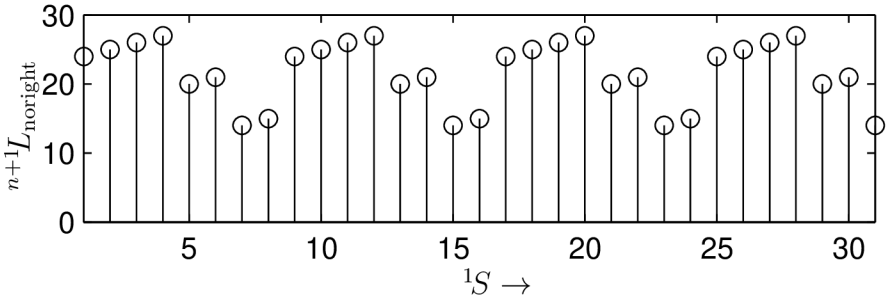
Then, according to the alg. 16, ${}^n\hat{\boldsymbol{x}}$ contain overlaps both to the next and the to the previous segment. So the last ${}^nR_{\text{inv}} + {}^{n+1}L_{\text{inv}}$ samples should be held for they will be added to the first respective samples in the following segment. The real-time SegLWT is depicted in fig. 5.6 for two arbitrary segments assuming $\text{len}({}^n\boldsymbol{x}) \gg {}^nR + {}^{n+1}L$. It is shown that the process exhibits delay of ${}^{n+1}S - {}^{n+1}S_{\text{inv}} + {}^{n+1}L_{\text{inv}}$ samples.



(a)



(b)



(c)

Figure 5.5: Extensions from Example , $^1S = 1, \ldots, 31$, (a) Left (^nR) and right (^{n+1}L) extensions using alg. 15 (b) Maximum left extensions $^{n+1}L_{\max}$ and minimum right extensions $^nR_{\min}$. (c) Left extensions ($^{n+1}L_{\text{noright}}$) only using formula (5.5).

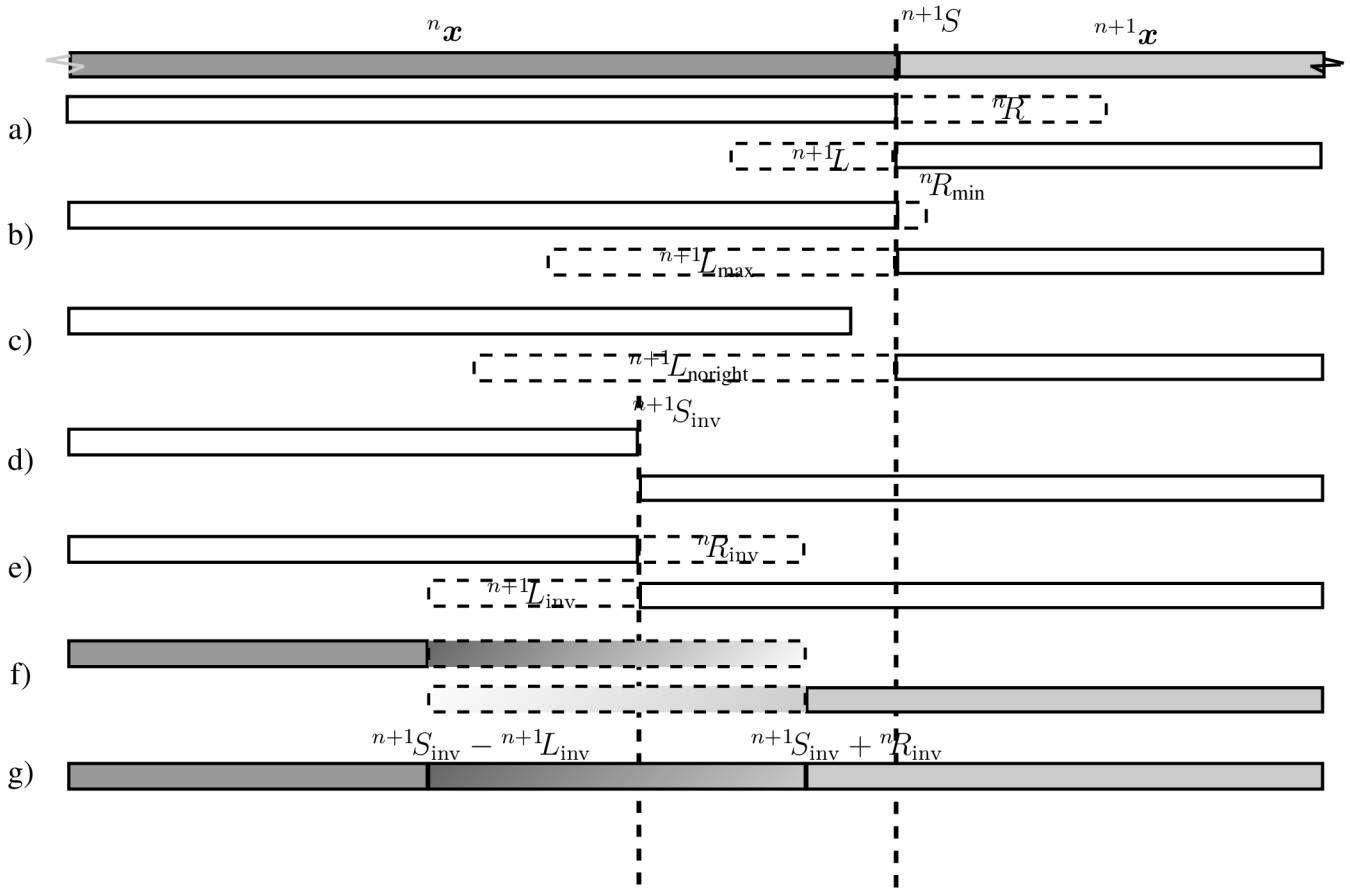


Figure 5.6: Depiction of the real-time SLWT for two segments ${}^{n}\boldsymbol{x}$ and ${}^{n+1}\boldsymbol{x}$ divided at index ${}^{n+1}S$. a) Extensions ${}^{n}R, {}^{n+1}L$ calculated using alg. 15. b) Extensions ${}^{n}R_{\min}, {}^{n+1}L_{\max}$ calculated using eq. (5.3), (5.4). c) Only left extension ${}^{n+1}L_{\text{noright}}$ using (5.5). d) The segments contain interleaved wavelet coefficients after cropping. ${}^{n}R$ samples were discarded from the n-th segment from the right, ${}^{n+1}L$ samples were discarded from segment $(n+1)$ from the left. e) The segments extended by ${}^{n}R_{\text{inv}}$, ${}^{n+1}L_{\text{inv}}$ zeros respectively. f) The segments after inverse LWT. The last ${}^{n}R_{\text{inv}} + {}^{n+1}L_{\text{inv}}$ samples of segment n are "overlap", so they are added to the same number of the first samples of the segment $(n+1)$. g) The segments after adding the overlap.

6 MULTIDIMENSIONAL EXTENSIONS

In this chapter, the extension of the segmented algorithm to the multidimensional isotropic discrete wavelet transform is presented. The separability property ensures that the principles from the previous chapters are valid even for multidimensional signals. For simplicity, the consecutive order of segments along the time dimension (if present as one of the dimensions) will not be exploited. The starting point is the maximally general algorithm from sec. 4.1 with a left extension only. The input of the multidimensional SegDWT is then an arbitrary box-shaped segment since the segment dimensions can differ in each direction. Since the segment order in each dimension cannot be defined unambiguously, the $\boldsymbol{S} = (S[0], S[1], \ldots, S[D-1])$, denote left, upper, (near,...) corner of the segment and $\boldsymbol{s} = (s[0], s[1], \ldots, s[D-1])$ its dimensions, where D denotes the number of the input signal dimensions. Accordingly, the ${}^{\boldsymbol{S}}\boldsymbol{L} = ({}^{S[0]}L_{\text{noright}}, {}^{S[1]}L_{\text{noright}}, \ldots, {}^{S[D-1]}L_{\text{noright}})$ denote left, top (front,...) extension (according to (4.16)) of the segment starting with $\boldsymbol{S}$. Extended segment dimensions are then $\boldsymbol{s}_{\text{ext}} = (L[0]+s[0], L[1]+s[1], \ldots, L[D-1]+s[D-1])$. One level isotropic DWT of one segment consists of a multiple one-dimensional DWT in each direction at a time. After that, there are additional $2^D - 1$ detail coefficient vectors in addition to one approximation coefficient vector sharing the same dimensions. The dimensions at level j can be derived similarly as in (4.19) taking a direction at a time.

In addition to the new SegDWT formulation (alg. 10 and alg. 11), the two presented modifications seem to be suitable for the multidimensional signals: Overlap-Save for SegDWT synthesis and subsequently the ROI SegDWT.

An example of the two-dimensional SegDWT algorithm usage on the real image is in fig. B.1.

7 APPLICATIONS

The chapter introduces two applications using SegDWT algorithm serving as a proof-of-concept.

The first application is VST effect plugin that allows custom real-time wavelet coefficient processing. The VST (Virtual Studio Technology) is a product of the Steinberg company which provides an interface for integrating software audio synthesizer and effect plugins with audio editors and recording systems. To run a plugin, a host application is needed to feed the plugin with audio data and processing the output. Since VST technology is nowadays a standard in the digital audio processing field, there are many such host application available. We used two host applications for the (succesful) testing of the created plugin module. The first one was *DSOUND GT-Player (EDU) Express*, version 2.6 Feb 17 2006. This host is simple enough and great for debugging, etc. It is no longer supported, but it is downloadable from the archive [51]. The second host was *Cubase 4* (EDU), version 4.5.2 Build 274.

The second application is an implementation of a parallel computation of DWT of images. The actual parallelization is done by the means of the Intel Threading Building Blocks (TBB) library framework [48]. There are other mechanisms for computation parallelization but the TBB became popular enough to be incorporated in the widely used computer vision OpenCV library [52].

7.1 VST plugin for Real-Time Wavelet Audio Processing

The implementation started from the template by Dr. J. Schimmel, which is accessible from URL [51]. The template is designed for creating VST plugin modules compatible with VST 2.4 specification.

The VST plugin uses "SegDWT" library which was created in the C++ language. The library consists of the `SegDWT.h` and `SegDWT.lib` files, whose source codes are available at [49]. The library processes single precision data types only. Both the forward and the inverse transforms are implemented in the `FloatSegDWT` class. Wavelet filters and wavelet coefficient processor are injected into the class by means of the `FloatWfilter` and `IWaveletCoeffProcessor` type objects, respectively. The main public functions of the class are summarized in Listing 7.1.

Listing 7.1: Important functions from `FloatSegDWT` class.

```
class FloatSegDWT{
...
public:
```

```
FloatSegDWT(FloatWfilter::Type waveletType, int newJ=1);
FloatSegDWT(FloatWfilter* newWavelet, int newJ=1);

void forwardOLS(float* in,int inLen,float* out[],int outLen[],
                unsigned long Sn=0);

void inverseOLA(float* in[], int inLen[],float* out, int outLen,
                unsigned long Sn=0);

void process(float* in, float* out, int inLen,
             unsigned long Sn=0);

void setWaveletProcessor(IWaveletCoeffProcessor* procesor_);
...
};
```

The class instance can be created using two constructors. The object containing the (four) wavelet filters can be either supplied directly by a pointer or created in the constructor according to the enumeration data type `FloatWfilter::Type` value. Function `forwardOLS` takes the input array `in` and calculates wavelet the coefficient arrays, which have to be allocated beforehand. Value `Sn` identifies the index of the first sample from the global indexing point of view. Function `inverseOLA` is complementary to `forwardOLS`. Function `process` initially calls `forwardOLS`, then processes the function of the `IWaveletCoeffProcessor` object and, lastly, the `inverseOLA`.

The storage of the overlaps is handled internally. Routines for allocating memory for arrays of wavelet coefficients are included as well.

The compiled VST plugin module is accessible through URL [49] in the ready-to-use form of a DLL file (∼1.2 MB). It suffices to copy the file to the plugin directory of the VST host software before the host is run.

The graphical user interface (GUI) is a simple, minimal one and consists of two parts, see Fig. 7.1. The left part of the plugin GUI appears always the same. It allows the user to set the global gain *after* the signal synthesis — `Gain`, choosing the wavelet filter — `Wavelet`, the depth of decomposition — `Depth`, and the method of processing the wavelet coefficients — `Process`. Wavelet filter names and filters were adopted from the Matlab Wavelet toolbox. The depth of decomposition J is limited by the size of the input buffer s_{buf} (which is controlled by the host application) so that $2^J \leq s_{\text{buf}}$, and at the same time, its maximum is set to $J = 10$.

The right part of the GUI depends on the selected `Process`. There are wavelet coefficient processors bundled with the plugin by default, however they serve mainly to "prove" the proposed algorithm. (Of course, if no modification was done to the

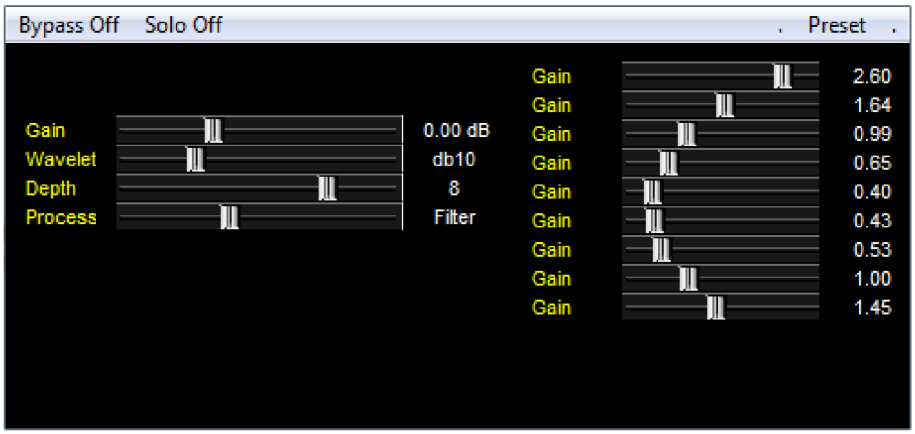


Figure 7.1: VST plugin GUI when "Filter" is selected for processing the coefficients.

coefficients, the output signal would be equal to the input signal up to numerical errors!) The controls at the right hand side allow setting parameters for the respective processors. The bundled processors are:

- *Default* — simply copies the wavelet coefficients and leaves them intact. This is incorporated to verify the perfect reconstruction.
- *Filter* — allows a multiplication of the wavelet coefficients by the specified values. Each decomposition level is assigned its own value.
- *Hard Thr* — hard-thresholds each subband with a specified value, i.e., the coefficients in absolute values smaller than the threshold are set to zero.
- *Random* — each coefficient in each subband is randomly perturbed. The extend of the scattering is controlled by the specified parameters.

The number of sliders is $J + 1$ in all these cases, each of them linked with the respective decomposition level. The depths go from the highest-frequency details to the approximation coefficients when taken from the top to the bottom.

The delay of the output in comparison to the input is always equal to $r(J)$ regardless of the buffer size.

However, the limit of the CPU performance can be reached on some computers when a demanding combination of parameters is set. For example, $J = 10$, wavelet `db10` (Daubechies 10 with $m = 20$), which leads to $r(J) = 19456$ samples of the left extension which have to be processed in addition to the actual segment samples, whose minimal length is restricted to $s_{\text{buf}} \geq 1024$.

This paragraph clarifies how to add your own real-time wavelet coefficient processor into the VST (2.4) plugin, extend and adapt it to your specific needs. The custom processor can be inserted into the plugin (or, more precisely, into the SegDWT library) by means of the Template pattern paradigm. To do this, create a class inherited from the interface called `IWaveletCoeffProcessor` which implements all its vir-

tual functions, see Listing 7.2. In the `setUserVariables` function in `vst_temp.cpp` file, dynamically create the instance of your processor, create the instance of the structure `ProcessorInfo` and fill the respective variables. Then append the structure object to the `processorList` vector.

After compiling and running the plugin in a host application, your processor should be accessible by means of the `Process` slider, at a position corresponding to its index in the `processorList` vector. To demonstrate how to access the individual wavelet coefficients, we display `DefaultProcessor process` function implementation in Listing 7.3.

7.1.1 Convolution and down/upsampling

The convolution and the down/upsampling are realized in the time domain. The standard two-direction cyclic buffer is exploited and the convolution and downsampling are done together in a single step, for both the filters simultaneously, according to the formulas

$$a^{(j+1)}[n] = \sum_{k=0}^{m-1} a^{(j)}[2n-k+m-1]\, h[k], \tag{7.1}$$

$$d^{(j+1)}[n] = \sum_{k=0}^{m-1} a^{(j)}[2n-k+m-1]\, g[k], \tag{7.2}$$

for $n = 0, \ldots, {}^{n}S^{(j)}_{\text{seg}} - 1$, j increasing from zero to $J-1$, the formulas have to be modified for the first and the last segments—they have to be treated slightly differently. However, the last segment cannot be identified properly in the VST live streaming audio setup.

The described process is equivalent to the "full" linear convolution followed by cropping $m-1$ samples from both sides, followed by the odd downsampling. This way, half the operations are saved.

In a similar manner, the upsampling and the convolution in the inverse DWT are done together in a single step, for both filters simultaneously, according to the

Listing 7.2: Structure of `IWaveletCoeffProcessor` interface.

```
class IWaveletCoeffProcessor{
public:
 virtual void process(float** in,float** out,int* coefLens,int J)=0;

 virtual void setParams(float* params,int paramLen) = 0;

 virtual void getParams(float* params,int paramLen) = 0;
};
```

Listing 7.3: Demonstration of accessing wavelet coefficients

```
void DefaultProcessor::process(float** in,float** out,
                               int* coefLens,int J){
  // temporary vaiables
  float* inSubband;float* outSubband;int jTemp=0; int coefLen=0;

  for(int j=1;j<=J;j++){
  // initiation of temp. variables for j-level detail coefficients
     jTemp = j-1;
     coefLen = coefLens[jTemp];
     inSubband = in[jTemp];
     outSubband = out[jTemp];

     // iteration over j-level detail coeff.
     for(int i =0;i<coefLen;i++){
     /**PLACE FOR j-th level i-th DETAIL COEFFICIENT PROCESSING**/
       outSubband[i] = inSubband[i];
     /***************END*************************************/
     }
  }
  // initiation of temp. variables for J-level approximation coeff.
  jTemp=J; coefLen=coefLens[J]; inSubband=in[J]; outSubband=out[J];

  for(int i =0;i<coefLen;i++){
  /**PLACE FOR i-th APPROXIMATION COEFFICIENT PROCESSING**/
    outSubband[i] = inSubband[i];
  /*********END*************************************/
  }
}
```

formula

$$
\begin{aligned}
a^{(j)}[n] = & \sum_{k=0}^{\left\lfloor \frac{m-1+(n \bmod 2)}{2} \right\rfloor} a^{(j+1)} \left[\left\lfloor \frac{n}{2} \right\rfloor - k \right] \tilde{h}\left[2k + (n \bmod 2)\right] \\
& + \sum_{k=0}^{\left\lfloor \frac{m-1+(n \bmod 2)}{2} \right\rfloor} d^{(j+1)} \left[\left\lfloor \frac{n}{2} \right\rfloor - k \right] \tilde{g}\left[2k + (n \bmod 2)\right]
\end{aligned}
\tag{7.3}
$$

$n = 0, \ldots, {}^{n_S}N_{\mathrm{seg}}^{(j)} - 1$ and $j = J-1, \ldots, 0$.

Again, the number of operations is reduced in comparison to the equivalent calculation consisting of upsampling both $\boldsymbol{a}^{(j+1)}$ and $\boldsymbol{d}^{(j+1)}$, followed by a linear convolution and the sum of the outcomes.

7.1.2 Fast convolution via FFT is not faster

Although it may seem tempting to perform the convolution and the resampling in the frequency domain using FFT, so far our tests have shown that this approach brings only a negligible performance increase and only in some extreme situations. In the rest of cases, the FFT approach performs worse. Moreover, the frequency domain filtering and the resampling bring, apart from the segment size constrictions, complications with implementation, and require a considerable revision of the SegDWT algorithm. The fact that the FFT approach does not perform so well in such situations is caused mainly by the short length of filters the wavelet filter bank comprises (i.e. $m \leq 20$) and by the relatively short segments, even after they had been extended $s_{\mathrm{ext}} = r_{\mathrm{noright}}(J) + s_{\mathrm{buf}}$.

We compared our implementation of the DWT analysis (forward transform only) in the time domain with frequency domain implementation using FFTW [53] 3.3.1 default 32bit dll binary distribution using Intel C++ compiler 12.0.1 with the `\03` optimization parameter. The tests were run 101 times and the median was taken as the result, which is plotted in Fig. 7.2. The testing machine was running Windows 7 Professional 64bit on Intel(R) Core(TM) i7 CPU 960 3.2GHz. We can conclude that the FFT implementation starts being beneficial for $J \geq 10$ and $m \geq 17$, since the segment length after the (maximal) extension depends on J exponentially and on m linearly, and it will be $s_{\mathrm{ext}} = 16368 + s_{\mathrm{buf}}$.

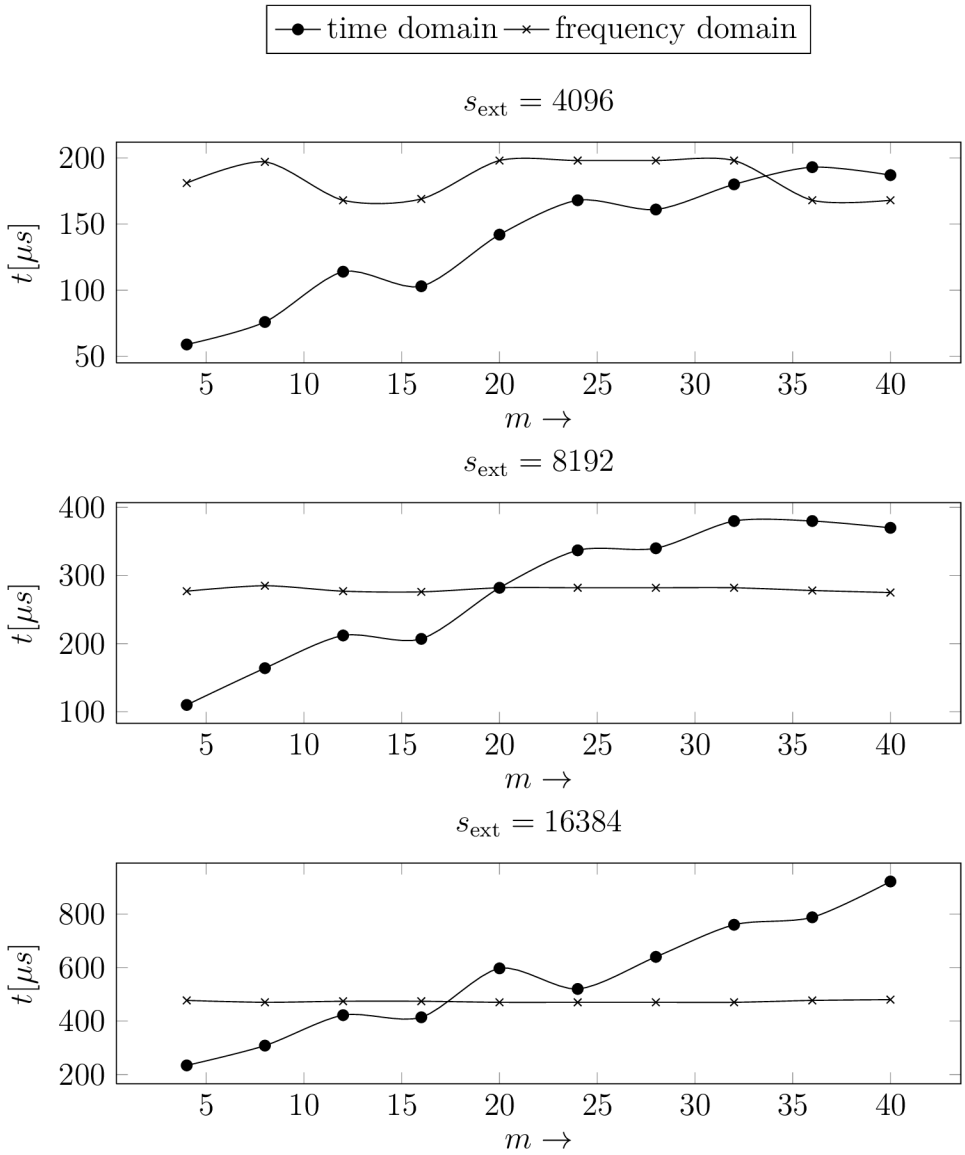


Figure 7.2: Comparison between time domain an frequency domain forward DWT implementations for different sequence lengths. Since the relative differences were not affected by the choice of the depth of decomposition, $J = 6$ was used.

7.2 Parallel 2D Wavelet Transform Library

The parallel implementation of the SegDWT was done in C++ and it is distributed as a static library available at [49]. To introduce parallelism into the implementation the Intel Threading Building Blocks (TBB) library was used. TBB provides algorithms and concepts which enable to fully exploit the possibilities of the multicore processors. It also provides automatic task scheduler and the automatic thread management. For more detailed information on using TBB in image processing see [5] or [48].

A basic concept of TBB is that tasks are recursively divided into smaller parts and then they are processed in parallel. When processing an image, the initial task range spans the whole image. This range is then recursively split into halves and when the new range is small enough, it is processed. TBB manages this splitting automatically and even allows task stealing to achieve a load balancing between working threads. A programmer can control the size of the smallest range by parameter `grainsize`, but if TBB decides not to split the range any more, the `grainsize` does not have to be reached so `grainsize` is only a coarse value that define size up to which range will not be split. Hence, the size of the segment to be processed is not known beforehand. At this point the new algorithm enters and after each division the extensions of the two affected segments are computed. According to the Theorem 9 these extensions are not affected by any other division of ranges.

At this point a very important fact needs to be highlighted. Every division of the range brings a redundancy of the computation. Obviously the number of redundant rows or columns of the input pixels is up to $r(J)$, which is depends on the length of the filters m and the depth of the wavelet decomposition J, see (2.2). So it is advisable to use as few divisions as possible, but at the same time it is important to effectively exploit all available threads to achieve the desired speedup.

7.2.1 Testing

Via testing, we would like to establish the optimum `grainsize` for a given $r(J)$ to reach the highest speedup possible. The serial version, to which parallel versions in different setups are compared to, is computed as if the whole image was one segment, so there are extensions by $r(J)$ only at the image borders. For testing purposes we used system running Intel C2Q Q9550 (4 cores). All data types were single precision 32-bit floating points. The compiler associated with Microsoft Visual Studio 2008 was used with the `\O2` optimization parameter. A median from 10 runs was taken as a result.

Firstly, we performed several tests with 4096×4096 px image for a fixed $r(5)$

and varying `grainsize`. It can be seen in Fig. 7.3 that the speedup is relatively independent on the choice of `grainsize`, but with the increasing $r(J)$ a bigger `grainsize` is needed. Using that, the speedup for a different $r(J)$ with an almost optimum choice of `grainsize` $\sim 2r(J)$ is shown in Fig. 7.4.

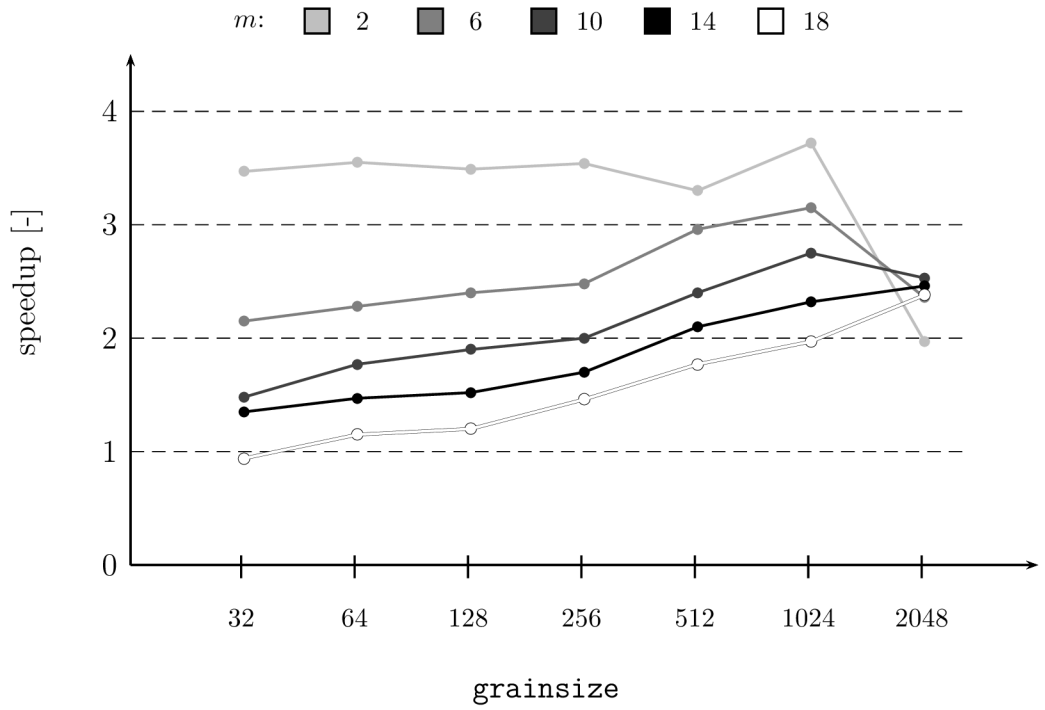


Figure 7.3: Speedup for increasing `grainsize` for different values of $r(5)$ (31, 155, 279, 403, 527 corresponding to filters with $m = 2, 6, 10, 14, 18$)

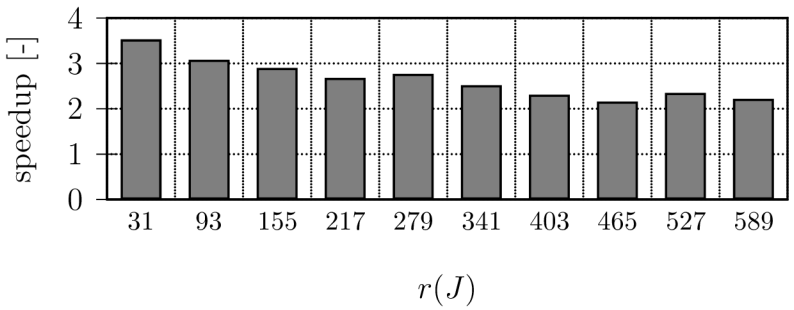


Figure 7.4: Speedup for increasing $r(J)$ for $J = 5$ and $m = 2, \ldots, 20$

We obtained another interesting result through Intel parallel Universe Portal. It is a web service where Intel offers computing resources on a 8-Core Intel Xeon@2.80GHz with hyperthreading (effectively 16 cores) (the service is suspended now). The tests were performed in the following setup: image 4096×4096 px, $m = 10$, $J = 4$ leading

to $r(J) = 135$, `grainsize` was set to 512 in both directions. In Fig. 7.5 the scalability of the speedup can be seen – that means the performance is increases with an increasing number of cores.

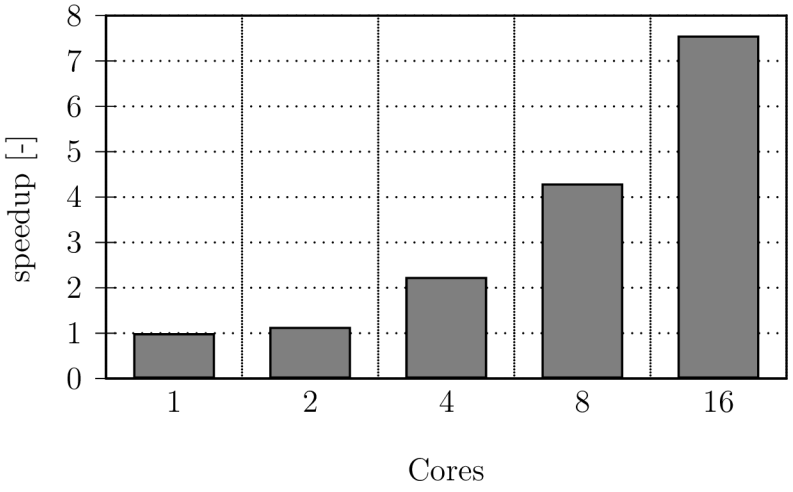


Figure 7.5: Speedup on Intel Parallel Universe

7.2.2 Software

The described implementation is distributed as a static library for 32bit OS Windows freely under the GPLv2 license. The library and the source codes are accessible from [49]. To use the library, it is necessary to include the header file `segDTWT.h` from the `include` directory and set the linker to include `segDTWT2D.lib` from the `lib` directory. The functions, which perform parallel forward and inverse wavelet transforms are:

```
segDTWTfwd_32f_C1(float* i_data, int i_widthStep, float* subbands[],
                  int widthSteps[], int levels, Size size,
                  separableWavelet* w);

segDTWTinv_32f_C1(float* subbands[],int widthSteps[], float* o_data,
                  int o_widthStep, int levels, Size size,
                  separableWavelet* w);
```

`float* i_data` – pointer to the beginning of the image

`int i_widthStep` – distance between two consecutive rows of the image in memory in bytes

`float* subbands[]` – array of pointers to the output subbands. For details see below.

`int widthSteps[]` – array of the distances between two consecutive rows in bytes in EACH individual subband

`int levels` – depth of decomposition, the J variable

`Size size` – dimensions of the input image

```
typedef struct
  int width;
  int height;
Size;
```

`separableWavelet* w` – object defining the wavelet filters

Prior to the calling of this functions one must have `separableWavelet` object prepared and also the memory for the output subbands needs to be allocated. One of the constructors for `separableWavelet` class accepts the wavelet name and the file name, where the wavelet filters definitions are located.

```
separableWavelet(string name="default",
             const char* file = "wavelets.dat")
```

The `wavelets.dat` file is distributed with the library and it contains wavelet filters defined in MatLab wavelet toolbox, but more filters can be added keeping the prescribed format.

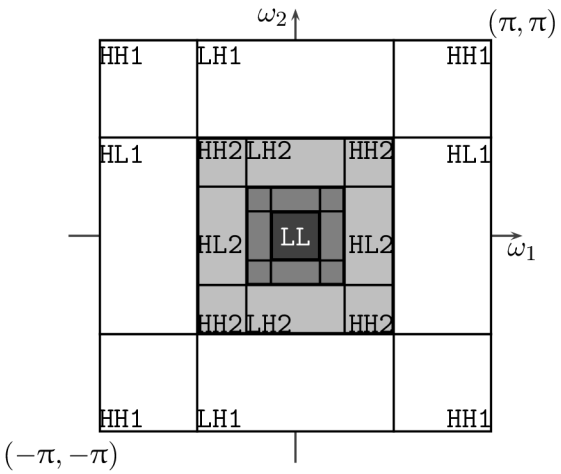


Figure 7.6: Subbands labeling

The `subbands[]` is an array of pointers to the output subbands. The pointers are stored in the following order: `subbands[0]=HL1,[1]=HH1,[2]=LH1,[3]=HL2,[4]=HH2,[5]=LH2 ...[last]=LL` assuming the subband labeling as depicted in fig. 7.6. The memory allocation can by done by means of the `allocateSubbands` function:

```
allocateSubbands(float* subbands[], int widthSteps[], int level,
                 int filter_length_L, int filter_length_H,
                 Size size);
```

`float* subbands[]` – array of pointers to be filled with pointers to the subbands

`int widthSteps[]` – array of the distances between two consecutive rows in bytes in EACH individual subband

`int filter_length_L/H` – length of the low- high-pass filter

8 CONCLUSION

The presented thesis was devoted to the generalization of a SegDWT algorithm to one- and multi-dimensional signals. The crucial shortcomings of the original algorithm were identified and removed. In addition, several optimizations of the algorithm were proposed. In particular, the necessity of the right extension of segments was removed and the possibility of segments of different lengths was introduced.

Several novel ideas were incorporated into the SegDWT algorithm including moving the overlaps from the time domain to the wavelet domain and thus reducing memory complexity. Further, the idea of the ROI SegDWT was introduced. For a chosen segment it identifies wavelet coefficients which participate on its full reconstruction and it further determines the left and the right extension of the analyzed segment which are necessary for the concerned coefficients calculation.

The presented novel SegLWT algorithm extends ideas of the SegDWT to the lifting scheme. The main motivation was a possible reduction of the overlaps. It turned out that the overlap lengths were strongly dependent on the actual filter bank factorization (lifting scheme) which is not unique. At best, the required overlap lengths are comparable to the SegDWT ones. However, the lifting scheme computational advantages (in-place, math operation number reduction) still remain.

All of the proposed modifications were confirmed in the Matlab simulations. The code with demos is accessible on the accompanied DVD and on the SegDWT algorithm webpage [49].

Two proof-of-concept applications were created to confirm the SegDWT algorithm design and its usability in the practice. That is the VST plugin for the real-time wavelet processing of the audio signals, which provide mechanism for custom (user-defined) wavelet coefficient processing. During the playback, no disturbing border artifacts occurs even after strong non-linear modifications of the wavelet coefficients. Also the VST technology buffer sizes are host application specific, not limited to the powers of two and can even vary during the playback. The SegDWT algorithm presented in this thesis is designed to cope with this behavior. The second application is the exploitation of SegDWT algorithm for the parallel implementation of the forward and the inverse DWT of the images using Intel Threading Building Blocks library. The advantage of the SegDWT usage in this type of application is its ability to let the library split the pixels blocks even during the computation execution for load balancing. It was shown that the achieved speedup is scalable and it is proportional to the number of the working threads of the CPU. On the other hand, the optimum minimal pixel block dimensions seem to be $\sim 2r(J)$. To retain the speedup scalability the images have to be large enough.

The SegDWT algorithm is not limited to the DWT only. Any tree structured

separable filter bank based transforms can be calculated segmentwise using SegDWT e.g. framelets [54], complex wavelet transform ℂWT [55], wavelet packets and others.

Hereafter the text sketches ideas for a future research. The segmentwise computations can be extended even to the non-iterated FIR filter bank structures for the noble identity filter bank representation has such structure. The only limiting factors seem to be the lengths of the impulse responses (directly defining the necessary overlaps) and the subsampling factor. Another possible extension of the SegDWT could be its adaptation to non-separable wavelet-type transforms [34] of the images and the multidimensional signals. A particular challenge lies in dealing with non-separable subsampling patterns.

Author's Selected References

[1] Z. Prusa, P. Rajmic, and J. Maly, "Segmentwise computation of 2D forward discrete wavelet transform," in *Proceedings of 33rd International Conference on Telecommunications and Signal Processing TSP2010*, Baden near Vienna, Austria, 2010, pp. 186–191.

[2] P. Rajmic and Z. Prusa, "Discrete wavelet transform of finite signals: Detailed study of the algorithm (in czech)," *Elektrorevue – Internetový časopis*, vol. 113, pp. 1–18, 2010. [Online]. Available: http://elektrorevue.cz/files/200000574-a3ef3a4e91

[3] P. Rajmic, Z. Prusa, and R. Konczi, "VST plugin module performing wavelet transform in real-time," in *Proceedings of 15th international conference on digital audio effects DAFx12*, 2012.

[4] Z. Prusa and P. Rajmic, "Segmented computation of wavelet transform via lifting schemes," in *Proceedings of 34th International Conference on Telecommunications and Signal Processing TSP2011*, Budapest, Hungary, 2011, pp. 433–437.

[5] Z. Prusa and J. Maly, "Parallel image processing using intel libraries," in *Proceedings of 32nd International Conference on Telecommunications and Signal Processing TSP2009*, Dunakiliti, Hungary, 2009, pp. 92–95.

[6] J. Maly, O. Venard, and Z. Prusa, "Color image compression based on SPIHT with separate channel processing," in *Proceedings of 32nd International Conference on Telecommunications and Signal Processing TSP2009*, Dunakiliti, Hungary, 2009, pp. 9–13.

[7] Z. Prusa and J. Maly, "Color image compression using SPIHT and its implementation (in czech)," *Elektrorevue – Internetový časopis (http://www.elektrorevue.cz)*, vol. 2009, no. 17, pp. 1–11, 2009. [Online]. Available: http://www.elektrorevue.cz/files/200000298-66c1367bb0/

[8] Z. Prusa and P. Rajmic, "Parallel implementation of 2D forward discrete wavelet transform on multicore CPUs," *Elektrorevue – Internetový časopis (http://www.elektrorevue.cz)*, vol. 1, no. 3, pp. 14–20, 2010. [Online]. Available: http://www.elektrorevue.cz/files/200000692-b1906b28a5/

[9] ——, "Real–time lifting wavelet transform algorithm," *Elektrorevue – Internetový časopis (http://www.elektrorevue.cz)*, vol. 2, no. 3, pp. 53–59, 2011. [Online]. Available: http://www.elektrorevue.cz/files/200000769-36aba37a58/

Other References

[10] P. Rajmic, "Exploitation of the wavelet transform and mathematical statistics for separation signals and noise, (in Czech)," Ph.D. dissertation, Brno University of Technology, Brno, 2004. [Online]. Available: http://www.utko.feec.vutbr.cz/~rajmic/phd_thesis/rajmic-dizertacni_prace-revize_23_10_2007.pdf

[11] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[12] A. Said and W. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.

[13] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image compression fundamentals, standards, and practice.* USA: Kluwer Academic Publishers, 2002.

[14] D. Donoho, "De-noising by soft-thresholding," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613 –627, 1995.

[15] J. Z. Wang and G. Wiederhold, "Wavemark: Digital image watermarking using daubechies' wavelets and error correcting coding," in *Proceedings of the SPIE International Symposium on Voice, Video and Data Communications*, 1998.

[16] R. N. Strickland and H. I. Hahn, "Wavelet transform methods for object detection and recovery," *IEEE Transactions on Image Processing*, vol. 6, no. 5, pp. 724–35, 1997.

[17] I. Drori and D. Lischinski, "Fast multiresolution image operations in the wavelet domain," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 9, no. 3, pp. 395 – 411, july-sept. 2003.

[18] G. Strang and T. Nguyen, *Wavelets and Filter Banks.* Wellesley College, 1996.

[19] R. Vargic, *Wavelets and filter banks (in Slovak).* Bratislava: STU in Bratislava, 2004.

[20] S. Mallat, *A Wavelet tour of signal processing, The Sparse way*, 3rd ed. Boston: Academic Press, 2008.

[21] A. N. Akansu and R. A. Haddad, *Multiresolution Signal Decomposition.* Academic Press, 2001.

[22] M. Vetterli and C. Herley, "Wavelets and filter banks: theory and design," *Signal Processing, IEEE Transactions on*, vol. 40, no. 9, pp. 2207 –2232, sep 1992.

[23] I. Daubechies, *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[24] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245–267, 1998.

[25] M. Misiti, Y. Misiti, G. Oppenheim, and P. J.-M., *Wavelet Toolbox User's Guide*, The Mathworks, Inc., 2000.

[26] C. Taswell and K. C. McGill, "Algorithm 735: Wavelet transform algorithms for finite-duration discrete-time signals," *ACM Trans. Math. Softw.*, vol. 20, no. 3, pp. 398–412, Sep. 1994.

[27] A. Cohen, I. Daubechies, and P. Vial, "Wavelets on the interval and fast wavelet transforms," *Applied and Computational Harmonic Analysis*, vol. 1, no. 1, pp. 54–81, 1993.

[28] P. Rajmic and J. Maly, "Boundary effects in the wavelet transform of finite discrete signals," in *Proceedings of the conference TSP2007*, Electrical Engineering Society. Brno: Brno University of Technology, 2007, pp. 81–84.

[29] I. Kharitonenko, X. Zhang, and S. Twelves, "A wavelet transform with point-symmetric extension at tile boundaries," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1357–1364, 2002.

[30] P. P. Vaidyanathan, *Multirate systems and filter banks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[31] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM J. Math. Anal.*, vol. 29, no. 2, pp. 511–546, 1997.

[32] G. Deslauriers and S. Dubuc, "Symmetric iterative interpolation processes," *Constructive Approximation*, vol. 5, pp. 49–68, 1989, 10.1007/BF01889598.

[33] D. L. Gall and A. Tabiatai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pp. 761–764, 1988.

[34] L. Jacques, L. Duval, C. Chaux, and G. Peyré, "A panorama on multiscale geometric representations, intertwining spatial, directional and frequency selectivity," *CoRR*, vol. abs/1101.5320, 2011.

[35] G. Peyré, *Advanced Signal, Image and Surface Processing*, 2010. [Online]. Available: http://www.ceremade.dauphine.fr/~peyre/numerical-tour/book/

[36] F. Lestussi, L. Di Persia, and D. H. Milone, "Comparison of on-line wavelet analysis and reconstruction: with application to ECG," in *5th International Conference on Bioinformatics ans Biomedical Engineering (iCBBE 2011)*, 2011, pp. 1–4, iSSN: 2151-7614.

[37] M. Tippach. (2003–2012) Asio4all — Universal ASIO Driver For WDM Audio. [Online]. Available: http://www.asio4all.com

[38] B. Leslie and M. Sandler, "A wavelet packet algorithm for 1D data with no block end effects," in *ISCAS (3)*, 1999, pp. 423–426.

[39] 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.

[40] R. Xia, K. Meng, F. Qian, and Z.-L. Wang, "Online wavelet denoising via a moving window," *Acta Automatica Sinica*, vol. 33, no. 9, pp. 897 – 901, 2007.

[41] J. Nealand, A. Bradley, and M. Lech, "Overlap-save convolution applied to wavelet analysis," vol. 10, no. 2, pp. 47–49, February 2003.

[42] W. Jiang and A. Ortega, "Lifting factorization-based discrete wavelet transform architecture design," pp. 651–657, 2001.

[43] J. Fridman and E. Manolakos, "On the scalability of 2-D discrete wavelet transform algorithms," *Multidimensional Systems and Signal Processing*, vol. 8, pp. 185–217, 1997.

[44] P. González, J. C. Cabaleiro, and T. F. Pena, "Parallel computation of wavelet transforms using the lifting scheme," *The Journal of Supercomputing*, vol. 18, pp. 141–152, 2001.

[45] J. Franco, G. Bernabe, J. Fernandez, and M. E. Acacio, "A parallel implementation of the 2D wavelet transform using CUDA," *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, pp. 111–118, 2009.

[46] W. van der Laan, A. Jalba, and J. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 132–146, 2011.

[47] J. Jan, *Digital Signal Filtering, Analysis and Restoration (in Czech).* VUTIUM, Brno, 2002.

[48] J. Reinders, *Intel Threading Building Blocks.* New York: McGraw-Hill, 2007.

[49] P. Rajmic. SegDWT web page. [Online]. Available: http://www.utko.feec.vutbr.cz/~rajmic/segwt/index_en.html

[50] W. A. Pearlman and A. Said, "Set partition coding: Part ii of set partition coding and image wavelet coding systems," *Foundations and Trends in Signal Processing*, vol. 2, no. 3, pp. 181–246, 2008.

[51] J. Schimmel. (2009) VST plug-in module template. [Online]. Available: http://www.utko.feec.vutbr.cz/~schimmel/DAP/download/VST_template.zip

[52] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[53] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[54] I. Daubechies, B. Han, A. Ron, and Z. Shen, "Framelets: Mra-based constructions of wavelet frames," vol. 14, pp. 1–46, 2000.

[55] I. W. Selesnick, R. G. Baramiuk, and N. G. Kingsbury, "The dual-tree complex wavelet transform," *IEEE Signal Processing Magazine*, vol. 22, no. 6, pp. 123–151, 2005.

LIST OF ABBREVIATIONS

ASIO	Audio Stream Input/Output, Trademark of Steinberg Media Technologies GmbH
BUPU	Bounded Uniform Partition of Unity
CDF 9/7	Cohen-Daubechies-Feauveau 9-tap/7-tap wavelet filter bank
CUDA	Compute Unified Device Architecture
`db#`	Order `#` Daubechies wavelet filter bank
DWT	Discrete Wavelet Transform
EZW	Embedded Zero Trees
FIR	Finite Impulse Response Filter
FFT	Fast Fourier Transform
JPEG	Joint Photographic Experts Group – image compression standard
JPEG2000	Joint Photographic Experts Group 2000 – image compression standard
LWT	Lifting Wavelet Transform
MRA	MultiResolution Analysis
MRI	Magnetic Resonance Imaging
OLA	Overlap-Add method for real-time convolution
OLS	Overlap-Save method for real-time convolution
ROI	Region of Interest
SegDWT	Segmentwise Discrete Wavelet Transform
SegLWT	Segmented Lifting Wavelet Transform
SPIHT	Set Partitioning In Hierarchical Trees
URL	Uniform Resource Locator
VST	Virtual Studio Technology, Trademark of Steinberg Media Technologies GmbH
WT	Wavelet Transform

LIST OF SYMBOLS AND MATH OPERATIONS

$\mathbb{R}^N$	N-dimensional Eucleidian space.
$\mathbb{Z}, \mathbb{N}, \mathbb{N}^0$	set of integers, positive integers and non-negative integers.
$\ell^2(\mathbb{Z})$	Hilberts space of sequences with finite energy.
$\delta[n]$	Dirac impulse. $\delta[n] = 1$ for $n = 0$ and $\delta[n] = 0$ otherwise.
m	wavelet filers length
J	depth of wavelet decomposition
D	number of signal dimensions
$\boldsymbol{x}$	input signal
$\hat{\boldsymbol{x}}$	reconstructed signal
${}^n\boldsymbol{x}$	input segment
${}^n s, s$	length of the input segment n. Superscript is dropped if segments have equal lengths.
${}^n\boldsymbol{x}_{\text{ext}}$	extended input segment
${}^n l$	input segment length including left extension
$\mathcal{W}^{(j)}$	detail subspace level j
$\mathcal{V}^{(j)}$	approximation subspace level j
$\boldsymbol{\psi}_p^{(j)}$	base vector of the $\mathcal{W}^{(j)}$ subspace
$\boldsymbol{\varphi}_p^{(j)}$	base vector of the $\mathcal{V}^{(j)}$ subspace
$\widetilde{\mathcal{W}}^{(j)}$	dual detail subspace level j
$\widetilde{\mathcal{V}}^{(j)}$	dual approximation subspace level j
$\widetilde{\boldsymbol{\psi}}_p^{(j)}$	base vector of the $\mathcal{W}^{(j)}$ subspace
$\widetilde{\boldsymbol{\varphi}}_p^{(j)}$	base vector of the $\mathcal{V}^{(j)}$ subspace
$\boldsymbol{g}_{\text{mr}}, \boldsymbol{h}_{\text{mr}}$	dilatation coefficient vectors
$\widetilde{\boldsymbol{g}}_{\text{mr}}, \widetilde{\boldsymbol{h}}_{\text{mr}}$	dual dilatation coefficient vectors
$\boldsymbol{a}^{(j)}, \boldsymbol{d}^{(j)}, \boldsymbol{c}^{(j)}$	approximation, detail, general coefficient vector at level j. Concrete coefficients are denoted as $a^{(j)}[k], d^{(j)}[k], c^{(j)}[k]$
$\boldsymbol{g}, \boldsymbol{h}$	wavelet analysis filters
$\widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{h}}$	wavelet synthesis filters

$\boldsymbol{f}$	general filter
ω	angular frequency $[\pi \cdot \boldsymbol{rad}]$
$H(\omega), G(\omega)$	frequency responses of the wavelet analysis filters $\boldsymbol{g}, \boldsymbol{h}$
l_0, r_0	variables for biorthogonal wavelet filers zero padding
${}^n\boldsymbol{a}^{(j)}, {}^n\boldsymbol{d}^{(j)}$	approximation and detail coefficient vectors at level j belonging to segment n
${}^n\boldsymbol{c}^{(j)}$	general coefficient vectors at level j belonging to segment n
${}^n\boldsymbol{c}^{(j)}_{\text{ext}}$	general coefficient vectors at level j belonging to segment n including redundant coefficients at the beginning of the vectors
${}^nS$	Index of n segment's first sample in the global point of view. Alternatively, sum of numbers of samples of all preceding segments or number of yet "processed samples".
${}^nS^{(j)}$	Index of n segment's first wavelet coefficient at the level j in the global point of view. $S^0_n = S_n$.
${}^nN^{(j)}_{\text{ext}}$	$\text{len}({}^n\boldsymbol{c}^{(j)}_{\text{ext}})$
${}^nN^{(j)}_{\text{disc}}$	Number of discarded coefficient
${}^nN^{(j)}_{\text{coef}}$	$\text{len}({}^n\boldsymbol{c}^{(j)})$
${}^nR$	right extension of segment n
${}^{nS}R$	right extension associated with index ${}^nS$
${}^nR_{\min}$	minimal right extension of segment n
${}^{n+1}L_{\max}$	maximal left extension of segment $(n+1)$
$r(j)$	$r(j) = (2^j - 1)(m-1)$. Returns total extension necessary for calculating one wavelet coefficient at level j. The following holds $r(J) = {}^nR + {}^{n+1}L$ and $r(J) = {}^{nS}R + {}^{nS}L$.
$r_{\text{red}}(j)$	$r_{\text{red}}(j) = (2^j - 1)(m-2)$. Reduced $r(j)$ when odd downsampling is considered.
${}^{nS}R_{\text{neg}}$	"Negative" right extension for SegDWT causal workaround.
${}^{nS}L_{\text{neg}}$	Left extension associated with ${}^{nS}R_{\text{neg}}$.
${}^{nS}L_{\text{noright}}$	Left extension for modification removing necessity of the right extension.
${}^{nS}L^{(j)}_{\text{noright}}$	Left extension in the approximation coefficient vectors at level j for

	overlap-save analysis modification transferring extensions into the wavelet domain.
${}^{nS}N^{(j)}_{\mathrm{OLA}}$	Number of approximation coefficients at level j overlapping to the next segment for overlap-add synthesis modification transferring overlap to the wavelet domain.
${}^{nS}R^{(j)}_{\mathrm{OLA}}$	Number of overlapping coefficients at level j for overlap-add analysis modification.
${}^{nS}R^{(j)}_{\mathrm{OLS}}$	Length of right extension at level j for overlap-save synthesis modification.
${}^{n}N^{(j)}_{\mathrm{discright}}$	Number of discarded coefficients from the right for overlap-save synthesis modification.
${}^{n}N^{(j)}_{\mathrm{extright}}$	Number of coefficients at level j produced by overlap-save synthesis modification after cropping the coefficient vectors from the left.
$\boldsymbol{P}, \boldsymbol{U}$	predict, update lifting step coefficient vectors
${}^{n+1}L^{(j)}$	left extension of the segment $(n+1)$ at level j for SegLWT
${}^{n}R^{(j)}$	Right extension of the segment n at level j for SegLWT
$e_{\min}, e_{\max}$	Minimum, maximum exponents of the lifting scheme steps Laurent polynomials
$\boldsymbol{a}^{(j)}_{\mathrm{E}}, \boldsymbol{a}^{(j)}_{\mathrm{O}}$	even, odd indexed approximation coefficients for the lifting scheme
${}^{n+1}S^{(j)}_{L}$	Index of the leftmost coefficient of the segment $n+1$ at level j after extending.
${}^{n+1}S^{(j)}_{R}$	Index of the rightmost coefficient of the segment n at level j after extending.
$\mathrm{len}(\boldsymbol{x})$	returns number of elements of the vector $\boldsymbol{x}$
$\uparrow N$	N-fold upsampling
$\downarrow N$	N-fold downsampling
$\boldsymbol{x} * \boldsymbol{h}$	linear convolution.
$\mathrm{odd}(n)$	indicator function. Returns 1 when $n = 2p$ for $p \in \mathbb{Z}$, and 0 otherwise.
$\mathrm{even}(n)$	indicator function. Returns 1 when $n = 2p+1$ for $p \in \mathbb{Z}$, and 0 otherwise.
$\mathrm{pow2}_p(n)$	indicator function. Returns 1 when $n = 2^p$ for $p \in \mathbb{Z}$, and 0 otherwise.
$\lceil \cdot \rceil, \lfloor \cdot \rfloor$	ceil, floor operations

$(a \bmod m)$	Modulo operation. $(a \bmod m) = a - \left\lfloor \frac{a}{m} \right\rfloor m$ where $a, m \in \mathbb{Z}$
$\langle \boldsymbol{x}, \boldsymbol{y} \rangle$	Vector scalar product $\boldsymbol{x}, \boldsymbol{y}$. For real sequences in $\ell^2(\mathbb{Z})$ it is defined as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{k \in \mathbb{Z}} x[k]y[k]$

LIST OF APPENDICES

A PROOFS

Proof of the theorem 9

Proof: Let us consider that the portion of the signal preceding segment n of length ${}^{n}s$ is divided into $(n-1)$ segments of general sizes ${}^{i}s$

$$ {}^{n}S = \sum_{i=0}^{n-1} {}^{i}s. \tag{A.1} $$

Then, according to (4.7) and the fact that ${}^{n}l = {}^{n}L + {}^{n}s$, we can write

$$ {}^{n+1}L = {}^{n}L + {}^{n}s - 2^{J}\left(\left\lceil \frac{{}^{n}L + {}^{n}s - r(J)}{2^{J}} \right\rceil + k\right). \tag{A.2} $$

For $n = 0 \Rightarrow {}^{0}L = r(J)$ the following holds, since

$$ {}^{1}L = r(J) + {}^{0}s - 2^{J}\left(\left\lceil \frac{{}^{0}s}{2^{J}} \right\rceil + k\right), \tag{A.3} $$

so for $n = 0$ the theorem holds

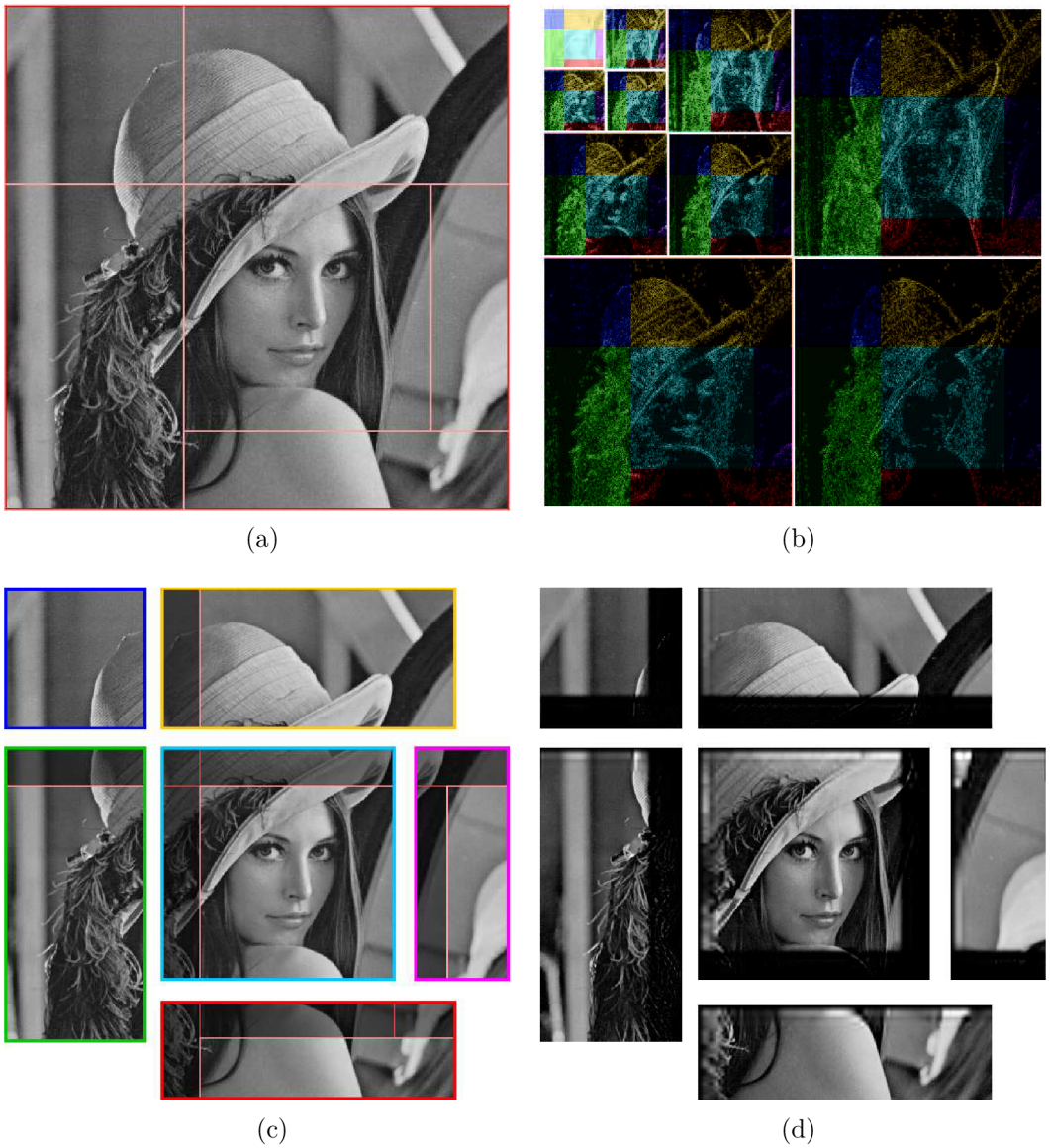
$$ {}^{0}R = r(J) - {}^{1}L = 2^{J}\left(\left\lceil \frac{{}^{0}s}{2^{J}} \right\rceil + k\right) - {}^{0}s. \tag{A.4} $$

Then we can continue using induction

$$
\begin{aligned}
{}^{n+2}L = & \\
&= {}^{n+1}L + {}^{n+1}S - 2^{J}\left(\left\lceil \frac{{}^{n+1}L + {}^{n+1}s - r(J)}{2^{J}} \right\rceil + l\right) \\
&= r(J) - {}^{n}R + {}^{n+1}s - 2^{J}\left(\left\lceil \frac{{}^{n}R + {}^{n+1}s}{2^{J}} \right\rceil + l\right) \\
&= r(J) - 2^{J}\left(\left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil + k\right) + {}^{n}S + {}^{n+1}s - 2^{J}\left(\left\lceil \frac{{}^{n+1}s - 2^{J}\left(\left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil + k\right) + {}^{n}S}{2^{J}} \right\rceil + l\right) \\
&= r(J) - 2^{J}\left(\left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil + k\right) + \sum_{i=0}^{n} {}^{i}s - 2^{J}\left(\left\lceil \frac{\sum_{i=0}^{n} {}^{i}s - 2^{J}\left(\left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil + k\right)}{2^{J}} \right\rceil + l\right) \\
&= r(J) - 2^{J}\left(\left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil + k\right) + {}^{n+1}S - 2^{J}\left(\left\lceil \frac{{}^{n+1}S}{2^{J}} - \left\lceil \frac{{}^{n}S}{2^{J}} \right\rceil - k \right\rceil + l\right) \\
&= r(J) + {}^{n+1}S - 2^{J}\left(\left\lceil \frac{{}^{n+1}S}{2^{J}} \right\rceil + l\right) \\
{}^{n+1}R = r(J) - {}^{n+2}L &= 2^{J}\left(\left\lceil \frac{{}^{n+1}S}{2^{J}} \right\rceil + l\right) - {}^{n+1}S
\end{aligned}
$$

□

B EXAMPLES



(a) (b) (c) (d)

Figure B.1: SegDWT applied to an image. Wavelet filter bank setting: $J = 3$ and $m = 8$. (a) Initial division into segments (blocks). (b) Wavelet coefficients colored according to the segments they belong to. (c) Properly extended segments. (d) Reconstructed segments.

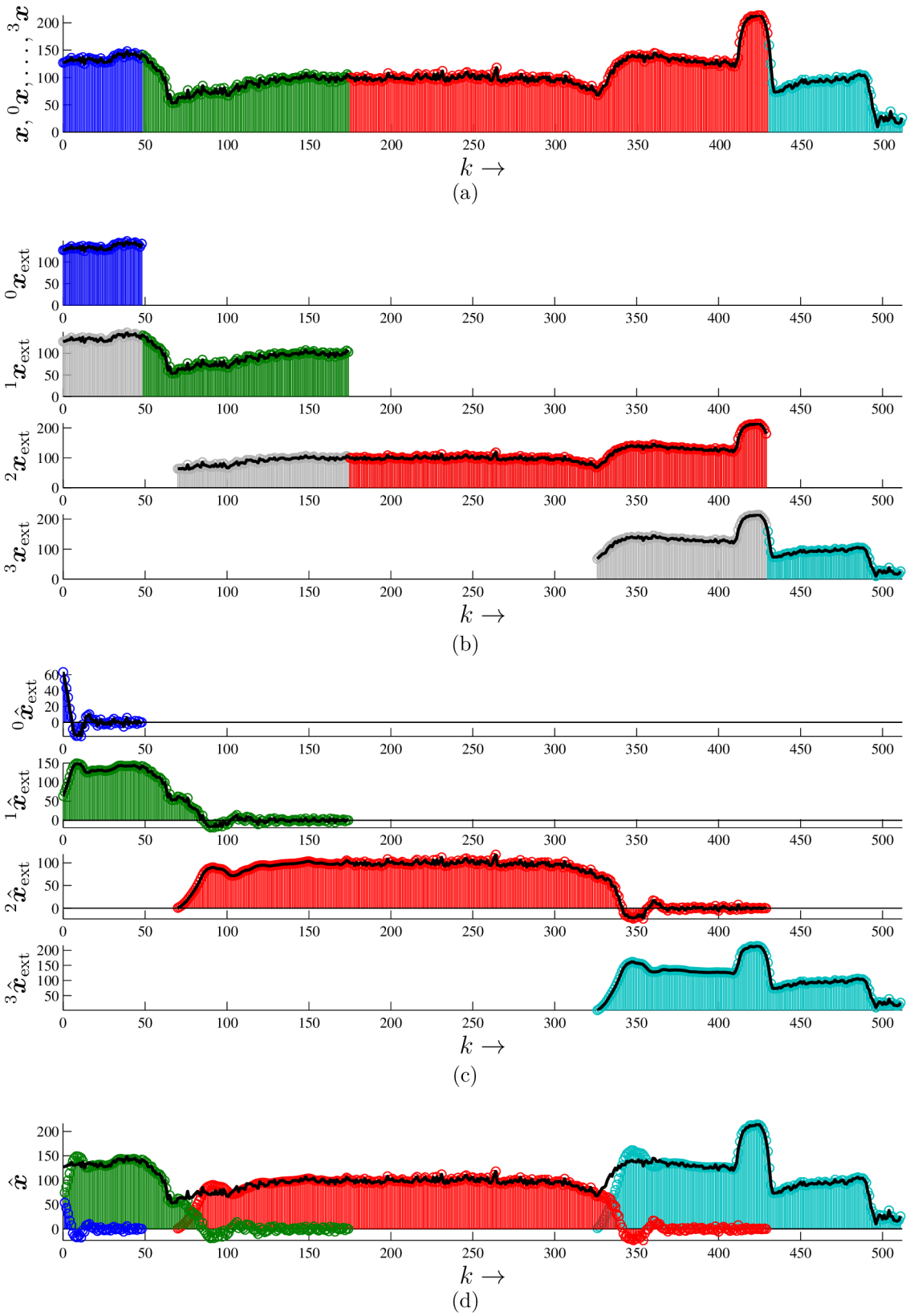


Figure B.2: No border artifact SegDWT example. The setup from fig. 2.2 ($J = 4$, $m = 8$) which results in ${}^nS{L}_{\text{noright}} = 90$ for each segment. (a) Initial segments. (b) Extended segments. (c) Segments after reconstruction. (d) Segments with added overlaps (black line).

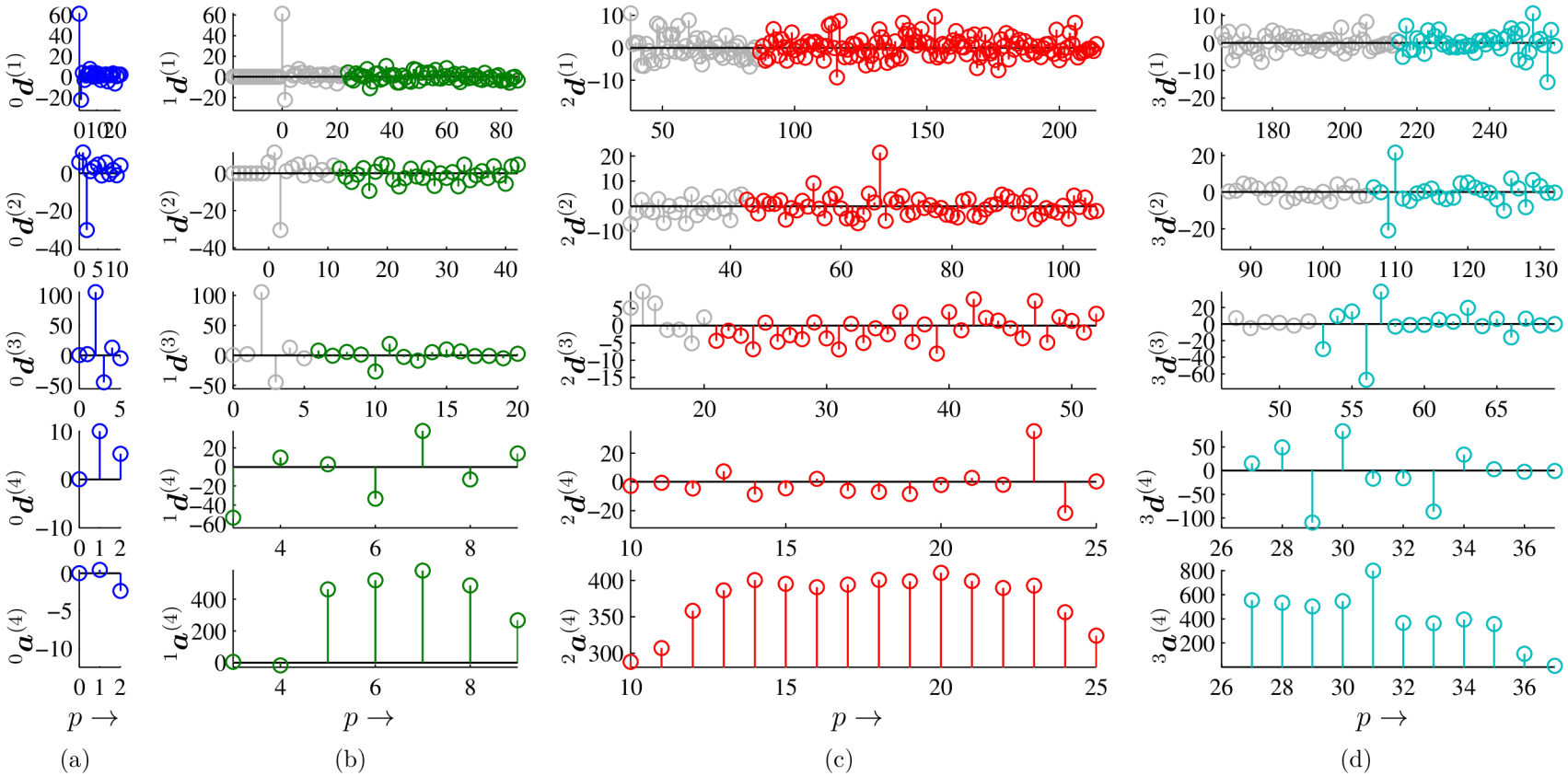


Figure B.3: Wavelet coefficients belonging to the extended segments. From the top row to the bottom detail coefficients at levels $j = 1, 2, 3$ and appox. coefficients. Gray coefficients are discarded.

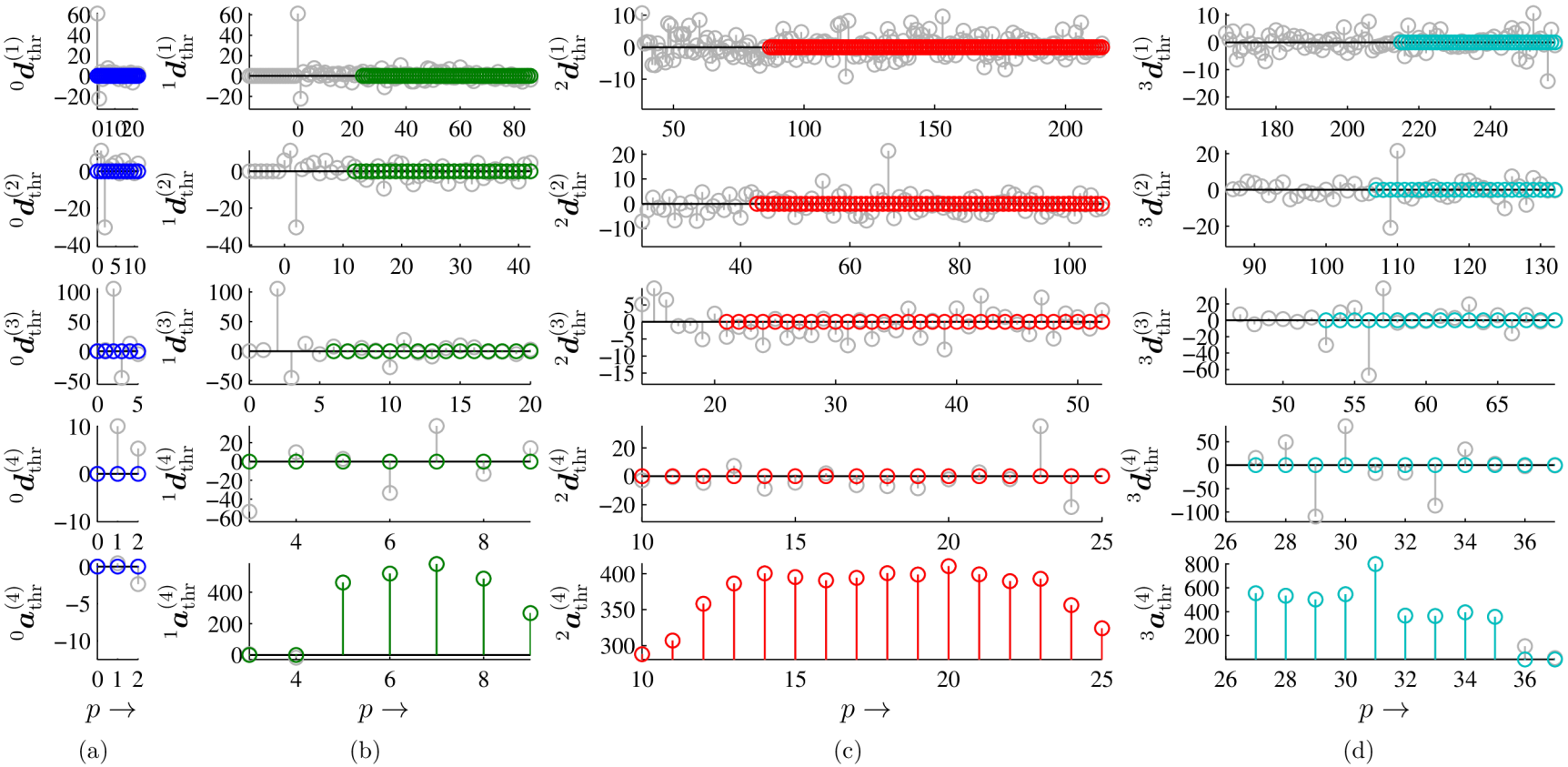


Figure B.4: Wavelet coefficients after hard thresholding. Coefficients are organized as in fig. B.2. Original coefficients are gray, colored after hard thresholding.

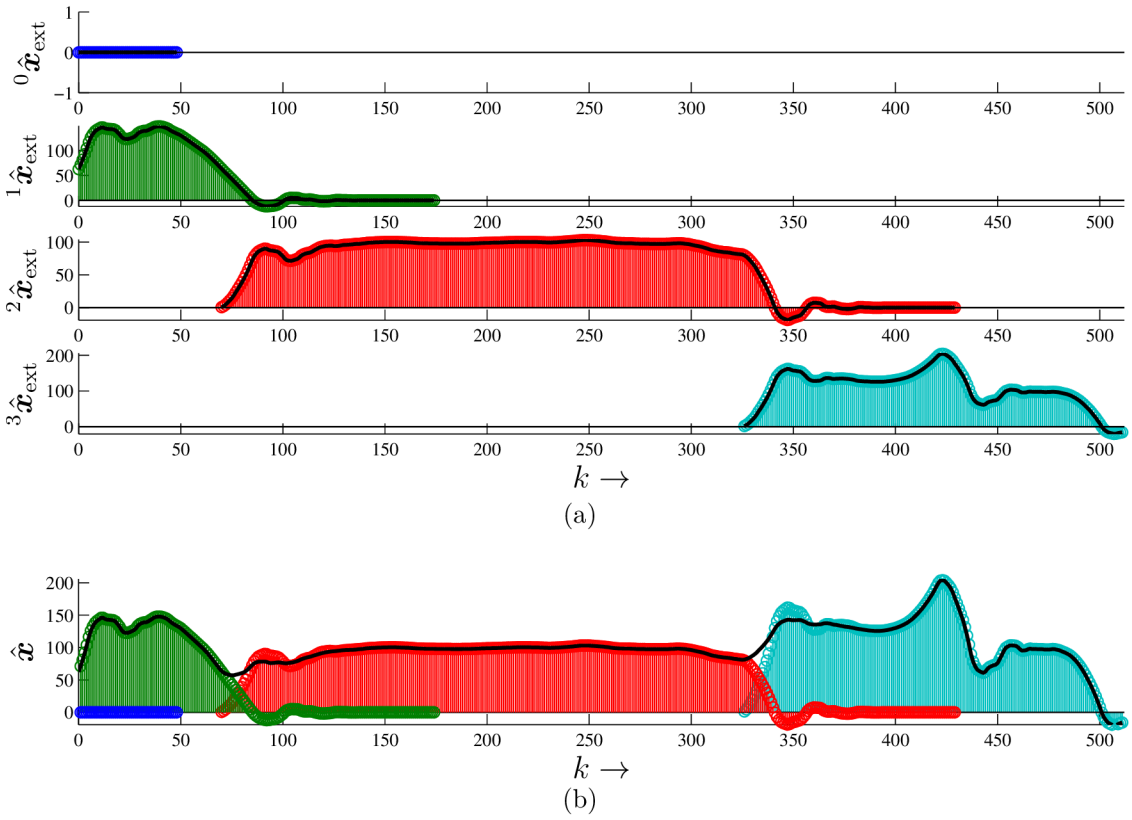


Figure B.5: Segment reconstruction after coefficient thresholding.

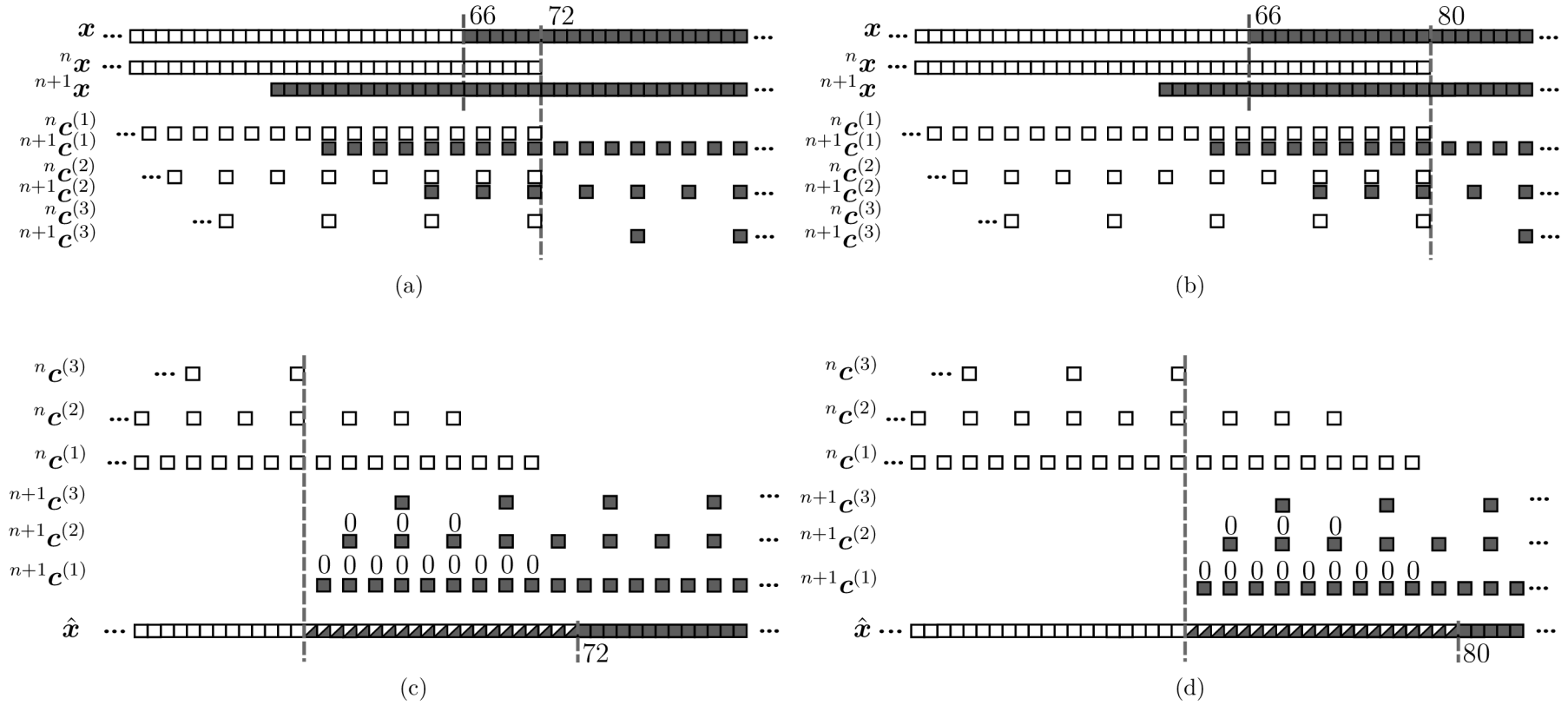


Figure B.6: SegDWT algorithm and its modifications examples. Depth of decomposition $J = 3$ and filter length $m = 4$, which leads to $r(J) = 21$. The dividing index between segments is ${}^{n+1}S = 66$. (a) Original SegDWT analysis algorithm with ${}^{n+1}SR_{\min} = 6$, ${}^{n+1}SL_{\max} = 15$. Dark coefficients on the left from the index 72 are discarded in the next step of the algorithm. (b) Original SegDWT analysis algorithm with the extension tradeof: ${}^{n+1}R_{\min} = 14$, ${}^{n+1}L_{\max} = 6$ ($2^J = 2^3 = 8$ samples are traded). (c) Original SegDWT algorithm synthesis. Zero coefficients are appended to the beginning of coefficients vectors: $r(3-1) = 9$ in level $j = 1$ and $r(3-2) = 3$ in level $j = 2$. The non-zero coefficients are shifted accordingly. (d) Original SegDWT algorithm synthesis with the extension tradeof.

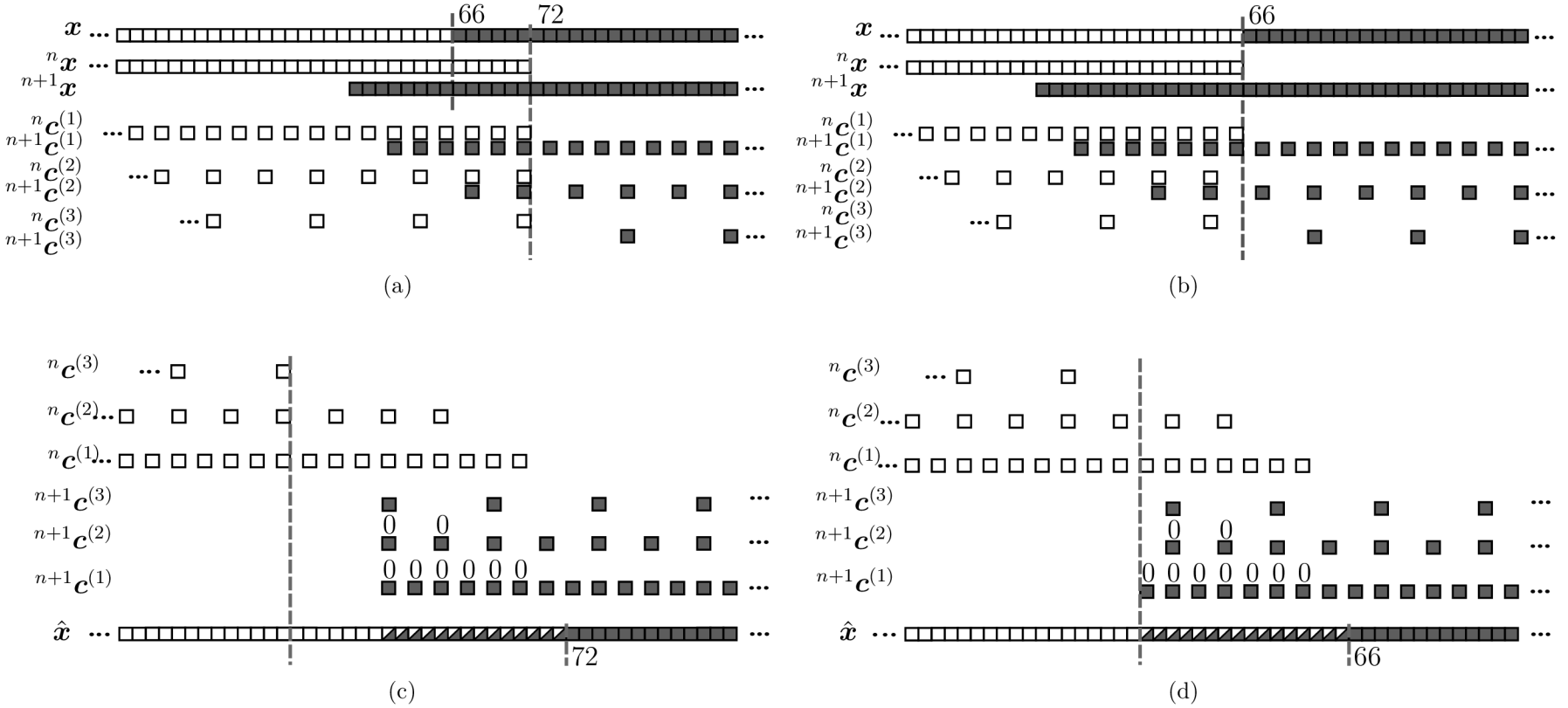


Figure B.7: SegDWT algorithm modifications examples. Depth of decomposition $J = 3$ and filter length $m = 4$, which leads to $r_{\text{red}}(J) = 14$. The dividing index between segments is ${}^{n+1}S = 66$. (a) SegDWT analysis modification employing odd subsampling with ${}^{n+1S}R_{\min} = 6$, ${}^{n+1S}L_{\max} = 8$. (b) SegDWT analysis modification employing odd subsampling and no right extension: ${}^{n+1S}L_{\text{noright}} = 14 + (66 \bmod 2^3) = 16$. (c) SegDWT synthesis modification for odd subsampling. Zero coefficients are appended to the beginning of coefficients vectors: $r_{\text{red}}(3-1) = 6$ at level $j = 1$ and $r_{\text{red}}(3-2) = 2$ in level $j = 2$. The non-zero reconstructed segments overlap is equal to $r_{\text{red}}(J) = 14$. (d) SegDWT synthesis modification for odd subsampling and no right extension. Zero coefficients are appended to the beginning of coefficients vectors: $r_{\text{red}}(3-1) + \left\lfloor \frac{(66 \bmod 2^3)}{2^1} \right\rfloor = 7$ at level $j = 1$ and $r_{\text{red}}(3-2) + \left\lfloor \frac{(66 \bmod 2^3)}{2^2} \right\rfloor = 2$ in level $j = 2$. The reconstructed segment overlap is equal to ${}^{n+1S}L_{\text{noright}} = 16$.

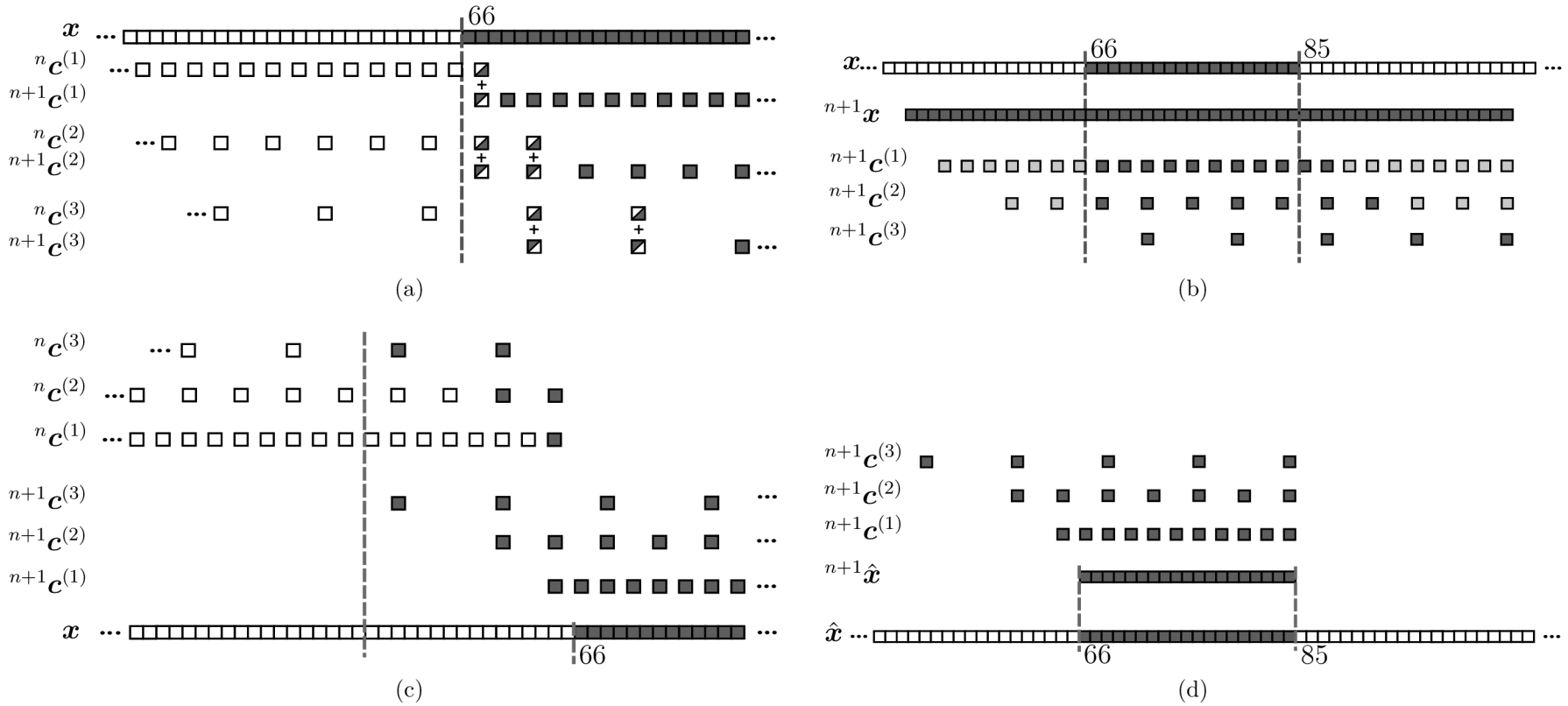


Figure B.8: OLA and ROI SegDWT algorithm modifications. Depth of decomposition is $J = 3$ and filter length is $m = 4$. (a) SegDWT analysis algorithm OLA modification. Numbers of coefficients forming overlaps in the wavelet coefficients vectors are: for $j = 1$, ${}^{66}R^{(1)}_{\mathrm{OLA}} = \left\lfloor \frac{r(1)+(66 \bmod 2^3)}{2^1} \right\rfloor - \left\lfloor \frac{(66 \bmod 2^3)}{2^1} \right\rfloor = 1$, for $j = 2$, ${}^{66}R^{(2)}_{\mathrm{OLA}} = 2$ and for $j = 3$, ${}^{66}R^{(3)}_{\mathrm{OLA}} = 2$. (b) ROI SegDWT analysis algorithm modification. The segment is initially extended by ${}^{n+1}S_{\mathrm{noright}} = 14 + (66 \bmod 2^3) = 16$ samples from the left and by ${}^{85}R_{\mathrm{OLS}} = \left\lfloor \frac{21+(85 \bmod 2^3)}{2^3} \right\rfloor 2^3 - (85 \bmod 2^3) = 19$ samples from the right. Gray coefficients are discarded prior to further processing. (c) SegDWT synthesis algorithm OLS modification. Initially, there are ${}^{66}R^{(j)}_{\mathrm{OLA}}$ coefficients borrowed from the following segment. The number of zero samples, appended to the beginning of wavelet coefficients at level j is $r_{\mathrm{red}}(J - j)$. (d) ROI inverse SegDWT.

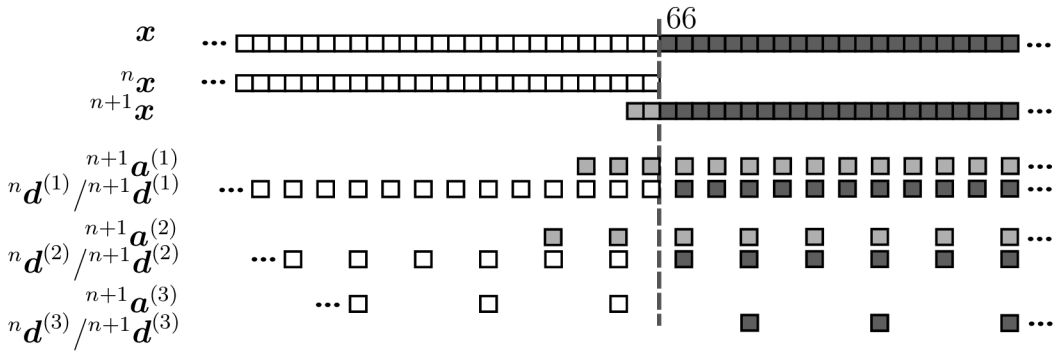


Figure B.9: SegDWT analysis with overlaps in the wavelet domain. $J = 3$, $m = 4$. Overlap-save SegDWT analysis: vectors ${}^{n+1}\boldsymbol{a}^{(0)}$ (which are identical to ${}^{n+1}\boldsymbol{x}$), ${}^{n+1}\boldsymbol{a}^{(1)}$, ${}^{n+1}\boldsymbol{a}^{(2)}$ are extended from the left by $m-1$ or $m-2$ coefficients taken from the previous approximation coefficient vectors.

C DVD CONTENT

The folders on the accompanied DVD are organized as follows:

- `algorithms` – folder contains Matlab code implementing proposed algorithms. See `readme` files.
  - `ch4_SegDWT` – Matlab implementation of the original SegDWT and the proposed SegDWT algorithm modifications
  - `ch5_SegLWT` – Matlab implementation of the novel SegDWT algorithm
  - `ch6_multidimensional_SegDWT` – Matlab demonstration of the 1D-3D SegDWT algorithm
- `software` – VST plugin and SegDWT library.
  - `ch7.1_VST_plugin` – contain source codes and binaries of the VST plugin
  - `ch7.2_Parallel_2D` – contain source codes and binaries of the parallel implementation of 2D SegDWT.
- `text` – contains the electronic version of this thesis in pdf and ps formats as well as all LaTeX source codes.

The codes can be also found on the SegDWT algorithm webpage [49].

Curriculum Vitae

Zdeněk Průša

Affiliation: Brno University of Technology, Brno, Czech Republic
Contact: Dynín 11, 37364 Dynín
zdenek.prusa@phd.feec.vutbr.cz
+420728741499

EDUCATION

2008–2012	Department of Telecommunications Brno University of Technology, Czech Republic Doctoral program
2006–2008	Brno University of Technology, Czech Republic Faculty of Electrical Engineering and Communications Master's program, Graduated with honors
2003–2006	Brno University of Technology, Czech Republic Faculty of Electrical Engineering and Communications Bachelor's program, Graduated with honors

PARTICIPATION IN PROJECTS

Holder:

- **2011** FRVŠ G1 – 1424 Including state-of-the-art signal transforms in education
- **2010** FRVŠ G1 – 3004 Including parallel programming for digital signal and image processing in education
- **2009** FRVŠ G1 – 2215 Intel Threading Building Blocks library for application development and education

Participated:

- **2012** FRVŠ F1 a – 2471 Innovation of courses on computer bitmap and vector graphics
- **2011–2012** FEKT-S-11-17 Research of complex methods for digital audio and image processing
- **2010** FEKT-J-10-8 Optimal algorithms for parallel wavelet transform of large scale images
- **2009** FRVŠ F1 a – 2677 New lectures for course Multimedia and graphics processors.

INVITED REVIEWS

- Elektrorevue – ISSN 1213-1539 `http://www.elektrorevue.cz`
- 33rd International Conference on Telecommunications and Signal Processing (TSP 2010) `http://tsp.vutbr.cz`
- Student Electrical Engineering, Information and Communication Technologies (EEICT 2010) `http://www.feec.vutbr.cz/EEICT/`