Computing Discrete-time Wavelet Transform Segmentwise

Pavel Rajmic

Dept. of Telecommunications, FEEC, Brno Univ. of Technology, Czech Republic

28. 10. 2011

Introduction

- Segmented discrete-time wavelet transform
  — computing transform coefficients segment-by-segment
- Why? Many applications:
  - audio signal processing
  - image processing (JPEG2000)
  - parallel computing
  - DSPs (low memory capacity). . .
- Generally: *Any* application requiring segmentwise processing of wavelet coefficients

Introduction

- Segmented discrete-time wavelet transform
  — computing transform coefficients segment-by-segment
- Why? Many applications:
  - audio signal processing
  - image processing (JPEG2000)
  - parallel computing
  - DSPs (low memory capacity)...
- Generally: *Any* application requiring segmentwise processing of wavelet coefficients *in an exact way*

Introduction

- Segmented discrete-time wavelet transform
  — computing transform coefficients segment-by-segment
- Why? Many applications:
  - audio signal processing
  - image processing (JPEG2000)
  - parallel computing
  - DSPs (low memory capacity). . .
- Generally: *Any* application requiring segmentwise processing of wavelet coefficients *in an exact way*
- Introducing abbreviations *SegDTWT* and *SegIDTWT*

Outline

Introduction
State-of-the-Art Methods

Introduction

State-of-the-Art Methods

- Continuous processing sample-by-sample

Introduction

State-of-the-Art Methods

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping
    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping

    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders

Introduction

State-of-the-Art Methods

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping

    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders

Introduction
State-of-the-Art Methods

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping

    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders

Introduction

State-of-the-Art Methods

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping
    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders
- Segments treated in context
  - Segments are first extended by the samples from beyond their borders, then processed independently
  - Segment processes "communicate" with the others after each loop of the processing

Introduction

State-of-the-Art Methods

- Continuous processing sample-by-sample
- Segments treated individually
  - Sharp borders
  - Windowing and overlapping
    — not suitable for nonlinear processing (e.g. thresholding), causes errors near the borders
- Segments treated in context
  - *Segments are first extended by the samples from beyond their borders, then processed independently*
  - Segment processes “communicate” with the others after each loop of the processing

- Published methods belonging to the second category suffer from limitations:
  - segment length is a power of two
  - just the forward DTWT is presented (parallel compression)
  - specialized to CDF 9/7 wavelet (JPEG2000)
  - (segment length is fixed)

- Published methods belonging to the second category suffer from limitations:
  - segment length is a power of two
  - just the forward DTWT is presented (parallel compression)
  - specialized to CDF 9/7 wavelet (JPEG2000)
  - (segment length is fixed)
- SegDTWT method gets over the limitations

  so that
  - s — segment length,
  - m — wavelet filter length,
  - J — decomposition depth/level,

  would be arbitrary!

- Algorithm of one stage:
  - Extend the signal's left and right borders
  - Filter (lowpass, highpass)
  - Cut specified number of coefficients at the borders
  - Downsample

Classic (forward) DTWT

Algorithm

- Algorithm of one stage:

  - Extend the signal's left and right borders
  - Filter (lowpass, highpass)
  - Cut specified number of coefficients at the borders
  - Downsample

Classic (forward) DTWT
Algorithm

- Algorithm of one stage:

  - Extend the signal's left and right borders
  - Filter (lowpass, highpass)
  - Cut specified number of coefficients at the borders
  - Downsample

Recursion on low-band produces "tree" of coefficients

Classic (forward) DTWT

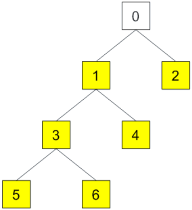
Algorithm

- Algorithm of one stage:

  - Extend the signal's left and right borders
  - Filter (lowpass, highpass)
  - Cut specified number of coefficients at the borders
  - Downsample

Recursion on low-band produces "tree" of coefficients

Depth $J = 3$:



Classic (forward) DTWT

Boundary handling

Signal is of finite length — convolution needs some samples beyond the borders!

Classic (forward) DTWT

Boundary handling

Signal is of finite length — convolution needs some samples beyond the borders!

- number of samples?
- how to extrapolate?

Classic (forward) DTWT

Boundary handling

Signal is of finite length — convolution needs some samples beyond the borders!

- number of samples?
- how to extrapolate?

We aim at the non-periodic type of border extension.

Classic (forward) DTWT — analysis

To derive the algorithm in general setup, we need a detailed analysis of DTWT.

Classic (forward) DTWT — analysis

To derive the algorithm in general setup, we need a detailed analysis of DTWT.

Given depth J and filter length m:

- How many coefficients do we compute from a given signal?
- What portion of signal do we need to compute specified coefficient(s)?
- How many such samples do the neighbouring coefficients have in common?
- How many wavelet coefficients are affected by the concrete boundary-extrapolating samples?
- etc.

Classic (forward) DTWT — analysis

Answers include powers of two due to downsampling.

Theorem

From signal of length s, DTWT algorithm returns

$$n_{\mathrm{coef}}(s,m,j) = \lfloor 2^{-j}s + (1-2^{-j})(m-1) \rfloor$$

wavelet coefficients at level $j \geq 1$.

Theorem

One needs

$$n_{\mathrm{samp}}(1,j,m) = (2^j - 1)(m-1) + 1.$$

input signal samples to compute one wavelet coefficient at j-th level.

(Forward) SegDTWT

The Goal

The goal is to compute the coefficients of the signal from its segments

(Forward) SegDTWT
The Goal

The goal is to compute the coefficients of the signal from its segments, as if we computed it from the whole signal.

(Forward) SegDTWT

The Principle

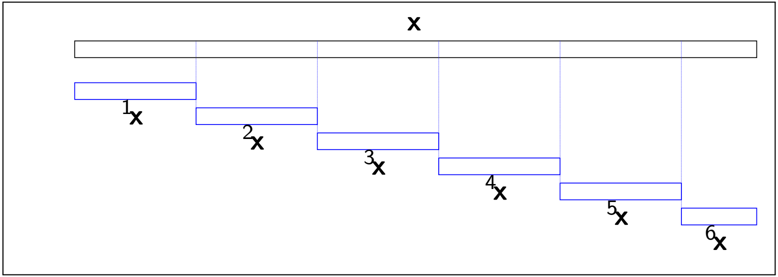


Figure: Scheme of primal segmentation. The last segment can be shorter than others.

(Forward) SegDTWT
The Principle

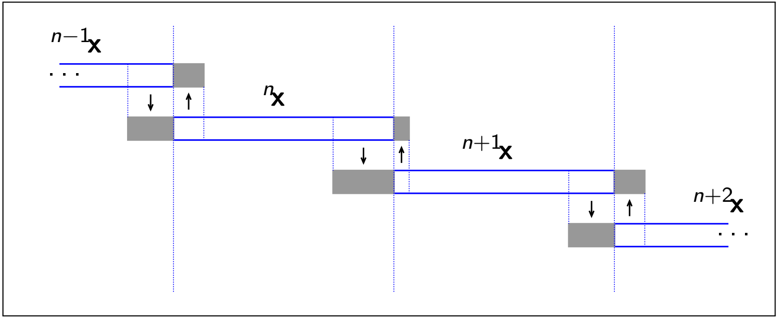
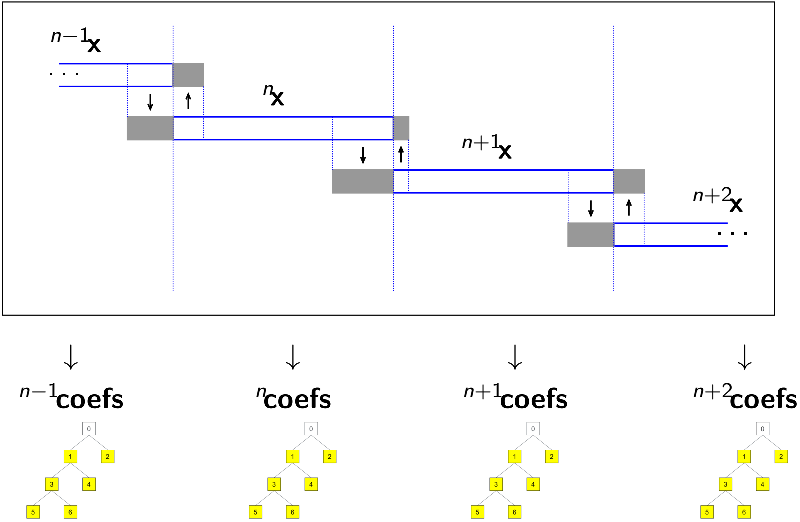


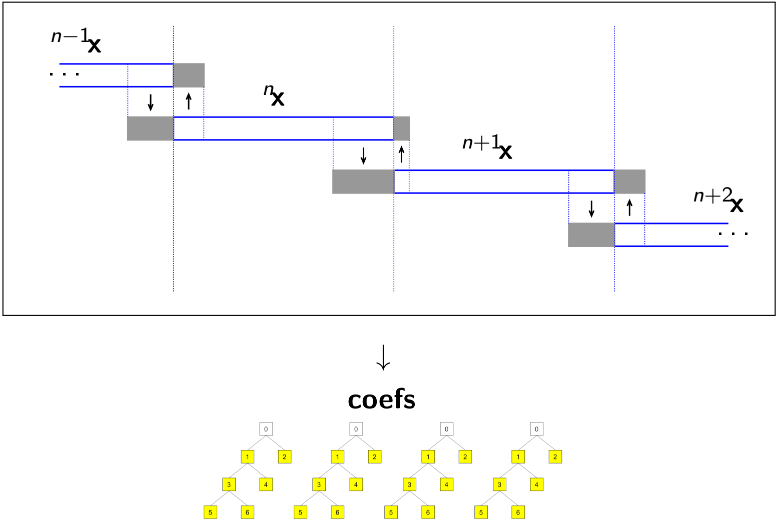
Figure: Extending of the segments. The lengths of the individual extensions are generally not the same from one segment to another!

(Forward) SegDTWT
The Principle

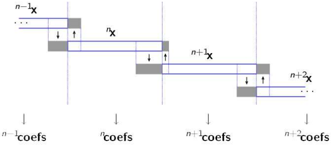


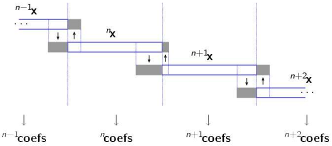
(Forward) SegDTWT

The Principle



- Problems arising from the fact that the (dyadic) **DTWT is not shift-invariant**.

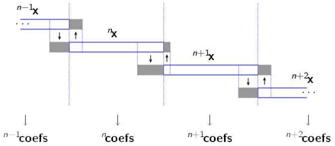




- Problems arising from the fact that the (dyadic) **DTWT is not shift-invariant**. However, DTWT is 2^J-shift-invariant.

(Forward) SegDTWT

Rules for the Extensions



- Problems arising from the fact that the (dyadic) **DTWT is not shift-invariant**. However, DTWT is 2^J-shift-invariant.
- The right extension as small as possible — introducing $R_{\min}(n)$, $L_{\max}(n)$.
- It can be shown that
  - $R_{\min}(n) = 2^J \left\lceil \frac{ns}{2^J} \right\rceil - ns$
  - $L_{\max}(n+1) = (2^J - 1)(m-1) - R_{\min}(n)$
- $R_{\min}$ is periodic, period equal to 1 achievable.

Segmented DTWT
Algorithm

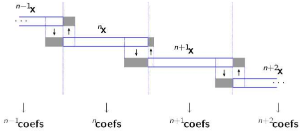
- For segment n do:
  - Extend the segment with $R_{\min}(n)$, $L_{\max}(n)$
  - Filter (lowpass, highpass)
  - Downsample
  - Trim off redundant coefficients

Segmented DTWT
Algorithm

- For segment n do:

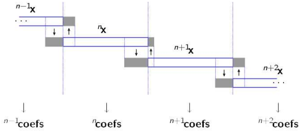
  - Extend the segment with $R_{\min}(n)$, $L_{\max}(n)$
  - Filter (lowpass, highpass)
  - Downsample
  - Trim off redundant coefficients

- For $J = 5$, $m = 8$, $s = 300$: alternating extensions $0 \leq R_{\text{min}}(n) \leq 31$, $186 \leq L_{\text{max}}(n) \leq 216$, with period 8



(Forward) SegDTWT
Examples

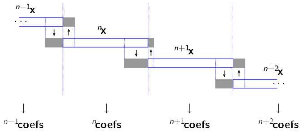
- For $J = 5, m = 8, s = 300$:
  alternating extensions
  $0 \leq R_{\min}(n) \leq 31$,
  $186 \leq L_{\max}(n) \leq 216$, with period 8




```
Command Window

>> db4_D_Lo = wfilters('db4','ld');  db4_D_Hi = wfilters('db4','hd');
>> [coefs, L, extensions, period] = segdtwt(1:5023, 300, 5, db4_D_Lo, db4_D_Hi)

coefs =

    [1x2515 double]    [1x1261 double]    [1x634 double]    [1x320 double]    [1x163 double]    [1x163 double]


L =

   300   300   300   300   300   300   300   300   300   300   300   300   300   300   300   300   223
   160   144   160   144   144   160   144   144   160   144   160   144   144   160   144   144   115
    80    72    80    72    72    80    72    72    80    72    80    72    72    80    72    72    61
    40    36    40    36    36    40    36    36    40    36    40    36    36    40    36    36    34
    20    18    20    18    18    20    18    18    20    18    20    18    18    20    18    18    20
    10     9    10     9     9    10     9     9    10     9    10     9     9    10     9     9    13
    10     9    10     9     9    10     9     9    10     9    10     9     9    10     9     9    13


extensions =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17
   217   197   209   189   201   213   193   205   217   197   209   189   201   213   193   205   217
    20     8    28    16     4    24    12     0    20     8    28    16     4    24    12     0   294


period =

     8
```

(Forward) SegDTWT
Examples

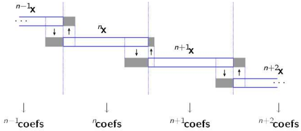
- For $J = 4, m = 12, s = 256$:
  $R_{\min}(n) = 0,\ L_{\max}(n) = 165$
  period is 1
  segment length after the extension: $256 + 165 = 421$.



(Forward) SegDTWT

Examples

- For $J = 4, m = 12, s = 256$:
  $R_{\min}(n) = 0,\ L_{\max}(n) = 165$
  period is 1



segment length after the extension: $256 + 165 = 421$.

```
Command Window

     1

>> db6_D_Lo = wfilters('db6','ld');  db6_D_Hi = wfilters('db6','hd');
>> [coefs, L, extensions, period] = segdtwt(1:4503, 256, 4, db6_D_Lo, db6_D_Hi)

coefs =

    [1x2257 double]    [1x1134 double]    [1x572 double]    [1x291 double]    [1x291 double]


L =

   256   256   256   256   256   256   256   256   256   256   256   256   256   256   256   256   256   151
   128   128   128   128   128   128   128   128   128   128   128   128   128   128   128   128   128    81
    64    64    64    64    64    64    64    64    64    64    64    64    64    64    64    64    64    46
    32    32    32    32    32    32    32    32    32    32    32    32    32    32    32    32    32    28
    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    19
    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    16    19


extensions =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18
   165   165   165   165   165   165   165   165   165   165   165   165   165   165   165   165   165   165
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0   270


period =

     1
```

(Forward) SegDTWT

Delay of the Algorithm

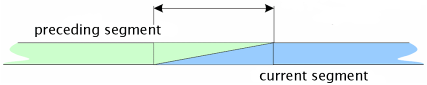
The input-output delay of the forward SegDTWT is maximally one segment, in special cases even without latency!

(Inverse) SegIDTWT
The Goal

- The goal is to compute the samples of the original segments from the respective sets of wavelet coefficients.
- The length of set of coefficients can differ from segment to segment!

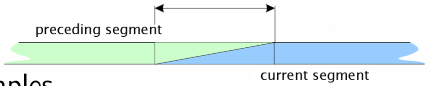
```
>> db4_D_Lo = wfilters('db4','ld');  db4_D_Hi = wfilters('db4','hd');
>> [coefs, L, extensions, period] = segdtwt(1:5023, 300, 5, db4_D_Lo, db4_D_Hi)

coefs =

    [1x2515 double]    [1x1261 double]    [1x634 double]    [1x320 double]    [1x163 double]    [1x163 double]


L =

   300   300   300   300   300   300   300   300   300   300   300   300   300   300   300   300   223
   160   144   160   144   144   160   144   144   160   144   160   144   144   160   144   144   115
    80    72    80    72    72    80    72    72    80    72    80    72    72    80    72    72    61
    40    36    40    36    36    40    36    36    40    36    40    36    36    40    36    36    34
    20    18    20    18    18    20    18    18    20    18    20    18    18    20    18    18    20
    10     9    10     9     9    10     9     9    10     9    10     9     9    10     9     9    13
    10     9    10     9     9    10     9     9    10     9    10     9     9    10     9     9    13


extensions =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17
   217   197   209   189   201   213   193   205   217   197   209   189   201   213   193   205   217
    20     8    28    16     4    24    12     0    20     8    28    16     4    24    12     0   294


period =
```

(Inverse) SegIDTWT
The Principle



(Inverse) SegIDTWT

The Principle

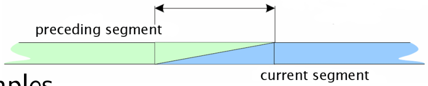


- Length of the overlap is $(2^J - 1)(m - 1)$ samples.

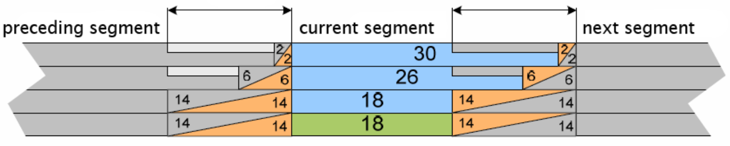
(Inverse) SegIDTWT

The Principle

- Length of the overlap is $(2^J-1)(m-1)$ samples.



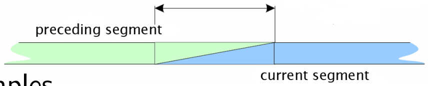
- Detail for $J=3$, $m=3$:



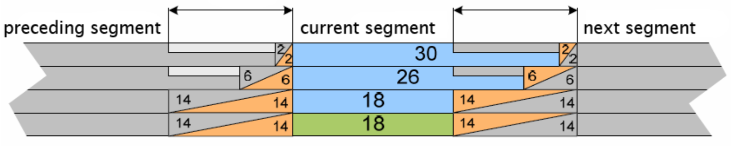
(Inverse) SegIDTWT

The Principle

- Length of the overlap is $(2^J - 1)(m - 1)$ samples.



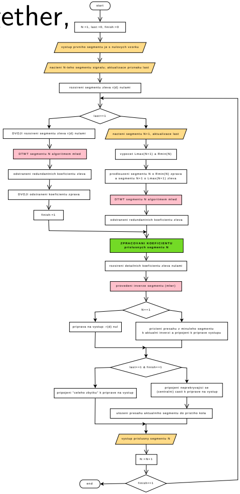
- Detail for $J = 3$, $m = 3$:



- Length of the overlap is $(2^{J-j} - 1)(m - 1)$ samples in the j-th level of transform.

The Real-Time Segmented DTWT

The **real-time algorithm** is created by an appropriate joining blocks from SegDTWT and SegIDTWT together, plus mechanism of interchanging the coefficients.



Implementations of 1D SegDTWT

- Matlab
- C++ offline version (command line)
- C++ VST plug-in module (real-time audio processing)

Version presented at DAFx 2006 utilized in “Vamp Plugins” by Centre for Digital Music at Queen Mary University, London

Generalization to 2D

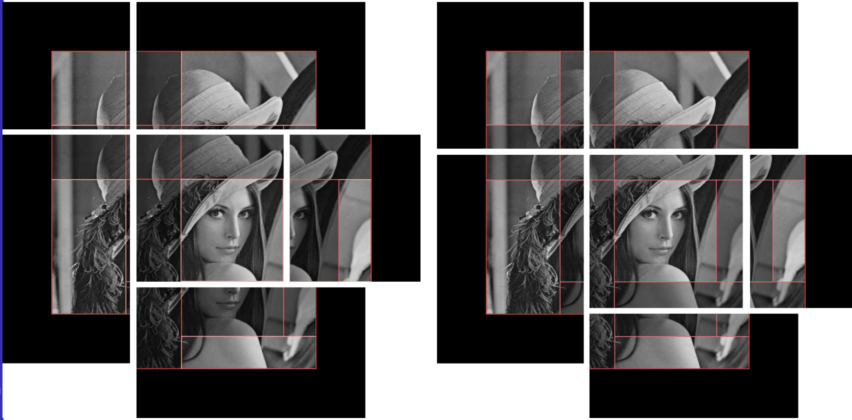
- work of Zdenek Prusa
- images
- main idea is straightforward (separability)
- no "time" in an image — not pushing the right extension to zero

Generalization to 2D

Example

Generalization to 2D

Example



Generalization to 2D

Main contributions

- any rectangular blocks
- lifting scheme!
- optimization for parallel processing
- parallel implementation in C++ using Intel libraries

Conclusion

- Forward and Inverse SegDTWT principles were presented

Conclusion

- Forward and Inverse SegDTWT principles were presented
- 1D and 2D

Conclusion

- Forward and Inverse SegDTWT principles were presented
- 1D and 2D
- Fully universal (m, J, s arbitrary)

Conclusion

- Forward and Inverse SegDTWT principles were presented
- 1D and 2D
- Fully universal (m, J, s arbitrary)
- Biorthogonal wavelets

Conclusion

Thanks for your attention!