Wavelet Domain Based Techniques for Video Coding

Dissertation
zur Erlangung des Doktorgrades
an der Naturwissenschaftlichen Fakultät
der Universität Salzburg

eingereicht von

Dipl.-Ing. Rade Kutil

Salzburg, 22. Juli 2002

Contents

List of Figures

List of Pseudo Codes

Chapter 1

Introduction

There have been many attempts to establish wavelet compression methods into video coding. However, a convincing wavelet video coding scheme is still to be found.

Today's DCT based methods are highly optimised and widespread. Many efficient implementations exist as well as know-how in DCT video compression. Even, cheap hardware units are available which support or are dedicated to these compression schemes. Of course, this makes it hard for wavelet methods to compete.

Moreover, new video coding methods should not just improve the efficiency of conventional methods. They should provide features that are not or only rudimentarily present in conventional methods. One of the most desirable features is scalability.

There are several types of scalability, of which rate-scalability (also known as quality scalability of embeddedness) is the most common. While in image compression, rate-scalability can be considered a solved problem, there is still room for improvement in video coding. MPEG is, for instance, known to lose up to 1 dB of overall quality for each scalability layer inserted in the bit-stream although a single scalability layer is still an insufficient approximation to scalability. The main problem of rate-scalability in video coding is motion compensation. However, there are approaches which dispense with motion compensation. The most common is 3-D transform coding, usually employing the wavelet transform. Chapter 5 uses an efficient scalable image codec on 3-D wavelet transformed video data and shows that it is a competitive alternative.

Other types of scalability are resolution progressive and colour component progressive coding. From a complexity point of view, encoder/decoder complexity scalability is a big issue. While encoder scalability is usually possible through motion estimation techniques with variable accuracy, the decoder cannot decide to reduce the computational demands or the buffer sizes while accepting minor quality degradations. Again, 3-D wavelet transform based methods can help here by partially decoding the bit-stream (which also reduces decoding time) and/or by performing the inverse wavelet transform on lower resolutions.

Apart from these issues, there are other reasons that wavelet methods didn't have much success in video coding. The biggest problem in this respect is that wavelet compression usually does not perform well on motion predicted residuals. The reason might be that wavelets are not a proper base for transform coding of residuals. However, the reason might as well be that existing codecs are not sufficiently adaptable to the specific distribution of energy on wavelet coefficients in these residuals. Another possibility to improve the situation is to modify the motion model so that it produces residuals that are more suitable for wavelet compression. This is basically what Chapter 6 aims at.

Wavelet image compression methods actually have been considered as potential parts of video compression standards such as MPEG-4. The reason is obvious since wavelet methods outperform DCT methods in image compression and image compression, applied on frames, is a core part of video coding. However, quality improvements were too small to justify the use of two fundamentally different kinds of compression techniques for residual and non-residual frames.

Therefore, this work tries to leave the beaten tracks of conventional video coding methods. It investigates methods that are solely based on operations in the wavelet domain. This means that raw video data is transformed into the wavelet domain without any preprocessing. Spatial as well as temporal redundancy is removed by coding methods based on wavelet transformed data.

Chapter 5 treats temporal redundancy by temporal wavelet transform. This leads to a three-dimensional wavelet transform which has already proved that it is in fact competitive to conventional video coding methods. As this method does not explicitly handle motion, many kinds of wavelet transforms have to be tested for the best compression performance. For this purpose, the unusual *anisotropic wavelet packet transform* is employed because it is a generalisation of the wavelet packet transform that is able to treat the temporal dimension differently from the spatial dimensions. Additionally, context modeling for arithmetic coding shall be enhanced so that it reflects the different behaviour of wavelet transformed video data in spatial and temporal dimensions.

Chapter 6 treats temporal redundancy directly by performing motion compensation in the wavelet domain. The problem with this method is that it requires shift operations to be executed in the wavelet domain. This is difficult because the wavelet transform is known to be highly shift variant. Thus, an efficient method for this task has to be found. Of course, there are also advantages of this method. Firstly, the encoder does not have to produce the reference frames for motion compensation which would require the inverse transform of a whole frame. In this way, encoder complexity can possibly be reduced. Secondly, the predictions should be more accurate because the predicted coefficients do not contain any energy from motion compensation artifacts such as block border discontinuities.

Because all these methods still have high computational demands (especially for real-time processing) there are some parallelisation efforts to be taken. Unlike

DCT based compression, the wavelet transform does not imply the segmentation of image data into blocks. While this improves the quality of reconstructed images, it makes parallelisation more complicated (at least for MIMD architectures). A lot of work has been done on the parallelisation of the wavelet transform itself. However, the parallelisation of the subsequent compression algorithms is still an open question. Chapter 7 concentrates on this topic.

As part of this work, several articles appeared in conference proceedings [25, 24, 12, 28, 27] and journals [58, 26]; more are in preparation.

Road Map

Chapter 2 will give a brief introduction in video coding concepts from pixels to motion compensation.

Chapter 3 describes the wavelet transform in all forms that are used later in the work. It explains how the fast wavelet transform works, introduces wavelet packets and elaborates anisotropic wavelet packets and its implementation details.

Chapter 4 develops a zero-tree based wavelet codec (SMAWZ) that is similar to the well known SPIHT. The codec is extended to be compatible with wavelet packets as well as anisotropic wavelet packets. It can also compress 3-D data and optionally uses arithmetic coding.

Chapter 5 applies the codec developed in Chapter 4 to 3-D video data and compares the results for all types of wavelet transforms to MPEG-4.

Chapter 6 develops a motion compensation method that works in the wavelet transform domain. In this way, a uniform transform domain codec can be built. In order to investigate if this method can improve the problematic wavelet compression of motion compensated residuals, comparisons with conventional (overlapped) block motion techniques for wavelet compression as well as comparisons with MPEG-4 are performed.

Chapter 7 shows how SPIHT and SMAWZ can be implemented on MIMD architectures. Problems and bottlenecks of both parallelisation approaches are detected and speedup results for a Cray T3E are presented.

Chapter 2

Video Coding Concepts

2.1 Basic Concepts

2.1.1 Raw Video Data

A *video sequence* is basically a three-dimensional array of colour pixels. Two dimensions serve as spatial (horizontal and vertical) directions of the (moving) pictures and one dimension represents the time domain.

A *pixel* (*pel*, *picture element*) is the smallest element of a video sequence. It is a small coloured dot and is represented by three colour values (see later). As there is a finite number of pixels in a picture, each pixel fills a certain area of a picture and is, therefore, associated with a width and a height. These values need not be equal. Their quotient is called pixel aspect ratio. This ratio is mostly equal to one, though. Moreover, a pixel has a certain duration, meaning that pixels are replaced in fixed time intervals. The change rate is referred to as *frame-rate*.

A frame is a set of all pixels that (approximately) correspond to a single point in time. No two pixels with equal spatial positions are contained in one frame. Basically, a frame is the same as a (still) picture. However, in interlaced video sequences, frame and picture are different.

2.1.2 Interlaced Video

In interlaced video, the set of lines with even numbers and the set with odd numbers are grouped together and created one after the other. This leads to the situation that alternating lines correspond to different points in time. The advantage of interlaced video is the impression of an increased frame rate without an increased data size. This makes motion in the video smoother and reduces flickering. However, motion compensation techniques (see later) become more complicated.

The distinct sets of lines are denoted as *fields* (top-fields and bottom-field). The collection of the two fields is called *frame*. *Picture* can refer to a frame or a field.

This work will concentrate on non-interlaced videos. Therefore, the terms frame and picture are synonymous.

2.1.3 Colour

Colour is always represented by three independent values because the human vision system responds to only three stimuli. These three values are called *components*. Collections of one type of component for a block of pixels or whole pictures or frames can be called components too.

The three components form a *colour space*. There are many possibilities to define how three values correspond to a colour. A very common and hardware oriented colour space is RGB, where the components determine the red (R), green (G) and blue (B) part of the colour. In image and video processing, common colour spaces are YUV and YCbCr, where the Y-component corresponds to the *intensity* of the colour. The other components (Y, U, Cb, Cr) are called the *chrominance* components.

As the human vision system is much more sensitive to the intensity (Y) component, chrominance components are usually compressed with a higher compression ratio. Therefore, the YUV or YCbCr representations are preferable to the RGB representation. Usually, the chrominance components are down-scaled to the half resolution (half in each spatial dimension, i.e. the chrominance data size is reduced to a $\frac{1}{4}$, the overall data size to $\frac{1}{2}$). The actual distribution of the bit-rate among the colour components is subject to psycho-visual investigations.

However, this work concentrates on monochrome data sets, i.e. mostly Y-components of video sequences. Extensions to colour videos can easily be done by performing the shown algorithms on each component separately. This is also the common method in existing image and video coding standards.

2.2 Image Coding Techniques

Video coding is very similar to image coding. In the vast majority of video coding algorithms, the frames of a video sequence are encoded as images using traditional image coding algorithms. Because of strong statistical relationship between frames and thereof resulting temporal redundancy, the frames are usually modified significantly before being encoded, though. This can be done efficiently by motion compensation methods (see Section 2.3).

The most common technique for image coding is *transform coding*. Transform coding is a general method to compress audio signals or images. In transform coding, first a *frequency transform* is applied. This has the advantage that the resulting coefficients have a statistically significant distribution and can be modeled and compressed more easily. This means: After the transform, some coefficients are predictably greater, others smaller. Thus, some coefficients can be neglected (lossy compression) and/or entropy encoded (lossless compression).

Common frequency transforms in image compression are the *discrete cosine transform* (DCT, used in JPEG [62] and MPEG [5, 21]) and the *wavelet transform* (WT, used in JPEG2000 [16]). The major disadvantage of the DCT is its higher computational complexity. Although this can be overcome by dividing the images into small blocks of fixed size and performing the transform on these blocks, the result is the introduction of blocking artifacts which become significant especially for high compression ratios. This work concentrates on the use of the wavelet transform.

There are well known and efficient algorithms for image coding using the wavelet transform which include EZW [49], SPIHT [48], SFQ [68], SLCCA [4] and EBCOT [56] which JPEG2000 [16] is based upon. Many extensions for the use of wavelet packets exist [8, 3, 56, 69, 41, 42].

The transformed image can be compressed by finding small coefficients and removing them completely from the data set (lossy compression). An important class of wavelet transform based image codecs is using a concept called zero-trees [49, 48]. Coefficients with approximately equal spatial positions are grouped together. This leads to a tree-like arrangement of coefficients. In this way, groups of neglectable coefficients (*zero-trees*) can be encoded efficiently by single symbols.

The remaining coefficients are *quantised*, i.e. the whole domain of coefficient's values is reduced to a set of values with equal distances. This distance is called the *quantisation step*. The wider the quantisation step is, the smaller is the set of values and the less information has to be encoded. Quantisation is also a form of lossy compression. Usually, the set of coefficients with a quantised value of zero is equal to the set of neglected (not encoded) coefficients (see above).

The quantised values are usually compressed by an entropy coder (lossless compression) such as a Huffman coder or an arithmetic coder. Often, the set of possible quantised values simply serves as the set of symbols that have to be encoded. The entropy coder assigns shorter bit codes to symbols with higher probability, which reduces the overall bit-rate (number of bits required to encode a whole sequence of symbols).

More sophisticated algorithms split each quantised value into the information about its most significant bit and the following (refinement) bits. This information is encoded in several passes from the most significant to the less significant bits. This has two advantages: Firstly, the statistical relationship of the significance and sign of neighbouring coefficients can be used to create efficient models for the entropy coder (e.g. grouping bits of neighbouring coefficients with equal significance together into a single symbol). Secondly, the resulting bit-stream can be truncated at arbitrary positions to create a lower-quality (and lower-bit-rate) representation of the image. This property is called *embeddedness* or *rate-scalability*. Important examples of codecs with this property can be found in [49, 48, 56, 16].

2.3 Motion Compensation

Motion compensation is a technique (or a set of techniques) to reduce temporal redundancy of a video sequence. A first approach would be to simply subtract a reference frame from a given frame. The difference is then called *residual* and usually contains less energy than the original frame. The residual can be encoded at a lower bit-rate with the same quality. The decoder can reconstruct the original frame by adding the reference frame.

A more sophisticated approach is to approximate the motion of the whole scene and the objects of a video sequence. The motion is described by some parameters that have to be encoded in the bit-stream. The pixels of the predicted frame are approximated by appropriately translated pixels of the reference frame. This gives much better residuals than a simple subtraction. However, the bit-rate occupied by the parameters of the motion model must not become too big.

Usually, the frames are processed in groups. One frame (usually the first) is encoded without motion compensation just as a normal image. This frame is called *I-frame* (intra-coded frame, MPEG terminology) or I-picture. The other frames are called *P-frames* or P-pictures and are predicted from the I-frame or P-frame that comes (temporally) immediately before it. The prediction schemes are, for instance, described as IPPPP, meaning that a group consists of one I-frame followed by four P-frames.

Frames can also be predicted from future frames. The future frames have to be encoded before the predicted frames, of course. Thus, the encoding order does not necessarily match the real frame order. Such frames are usually predicted from two directions, i.e. from the I- or P-frames that immediately precede or follow the predicted frame. These bidirectionally predicted frames are called B-frames. A coding scheme could, for instance, be IBBPBBPBBPBB.

2.3.1 Global Motion Compensation

In global motion compensation (MC), the motion model basically reflects camera motions such as dolly (forward, backwards), track (left, right), boom (up, down), pan (left, right), tilt (up, down) and roll (along the view axis). It works best for still scenes without moving objects. There are several advantages of global motion compensation:

- It models precisely the major part of motion usually found in video sequences with just a few parameters. The share in bit-rate of these parameters is neglectably small.
- It does not partition the frames. This avoids artifacts at partition borders.
- A straight line (in the time direction) of pixels with equal spatial positions in the frame corresponds to a continuously moving point in the real scene. Therefore, 3-D coding of motion compensated frames (as in [63])

is a reasonable approach. Other MC schemes introduce discontinuities in the time direction.

However, moving objects are not sufficiently represented by global motion compensation. Thus, other methods are preferable.

2.3.2 Block Motion Compensation

In block motion compensation, the frames are partitioned in blocks of pixels (e.g. macroblocks of 16×16 pixels in MPEG). Each block is predicted from a block of equal size in the reference frame. The blocks are not transformed in any way apart from being shifted to the position of the predicted block. This shift is represented by a *motion vector*.

The motion vectors are the parameters of this motion model and have to be encoded into the bit-stream. As the motion vectors are not statistically independent (e.g. if two neighbouring blocks belong to the same moving object), they are usually encoded differentially to save bit-rate. This means that the difference of the motion vector and the neighbouring motion vector(s) encoded before is encoded. An entropy codec can exploit the resulting statistical distribution of the motion vectors (around the zero vector).

It is possible to shift blocks by non-integer vectors, which is called *sub-pixel precision*. This is done by interpolating the pixel's values. Usually, the precision of the motion vectors is increased by one bit: *half-pixel precision*. Of course, the computational expense for sub-pixel precision is much higher since the interpolation functions can be quite consuming.

The big disadvantage of block motion compensation are the discontinuities introduced at block borders (blocking artifacts). They have the form of sharp horizontal and vertical edges which, firstly, are highly visual for the human eye and, secondly, produce ringing effects (big coefficients in high frequency subbands) in the frequency transform used for transform coding of the residual frames.

2.3.3 Overlapped Block Motion Compensation

Overlapped block motion compensation (OBMC) [44, 54] is a good solution to these problems because it not only increases prediction accuracy but also avoids blocking artifacts. Blocks are usually twice as big in each dimension and overlap quadrant-wise with all 8 neighbouring blocks. Thus, each pixel belongs to 4 blocks. There are 4 predictions for each pixel which are summed up to a weighted mean. For this purpose, blocks are associated with a window function that owns the property that the sum of 4 overlapped windows is equal to 1 everywhere.

2.3.4 Motion Estimation

Motion estimation (BME, OBME) is the process of finding optimal or near-optimal motion vectors which minimise the overall prediction error. The prediction error of a block is defined as the mean squared error (MSE) between predicted and actual pixel values over all pixels of the block.

To find optimal MVs, one basically has to calculate the block prediction error for each MV within a certain search range and pick the one with the smallest error (full search). A faster and sub-optimal method is to use a coarse search grid for a first approximation and to refine the grid in the surrounding of this approximation in further steps. The most common representative of this method is the 3-step search which uses search grids of 3×3 MVs and 3 refinement steps to get an overall search range of 15×15.

For OBME, the pixel-wise prediction errors of a block and its overlapping neighbouring blocks have to be weighted and summed according to the window function before being squared. As in the process of successively finding/refining MVs some neighbouring MVs are not known yet, the corresponding prediction errors can be ignored (not added) as a sub-optimal solution.

The major disadvantages of OBMC are increased computational complexity of OBME, and the fact that prediction errors and, thus, also the optimal MVs depend on neighbouring blocks/MVs. Therefore, there is no algorithm with polynomial complexity that guarantees optimal MVs. However, there are near-optimal iterative [44] and non-iterative [53] methods with acceptable computational complexity.

2.4 Rate and Distortion

2.4.1 Quality measurement

To compare the visual quality of video sequences (e.g. before and after compression), the peak signal to noise ratio (PSNR) is used. It is a numerical representation of how equal one pixel-set is to another. It is defined as:

$$\mathrm{PSNR} = 20 \log_{10} \frac{255}{\sqrt{\mathrm{MSE}}}$$

where

$$\mathrm{MSE} = \frac{1}{|A|} \sum_{\mathbf{x} \in A} (p(\mathbf{x}) - \bar{p}(\mathbf{x}))^2$$

where $p(\mathbf{x})$ is the pixel (grey value, $\in [0, 255]$) of the reference pixel set at position $\mathbf{x}$ and $\bar{p}(\mathbf{x})$ is the same pixel of the compared pixel set. A is the range in which pixels are arranged. When considering images or frames, usually $A = [1 \ldots \mathrm{width}] \times [1 \ldots \mathrm{height}]$. $|A|$ is the number of pixels, i.e. $|A| = \mathrm{width} \cdot \mathrm{height}$. Usually, p and $\bar{p}$ represent the uncompressed and the reconstructed images or frames respectively. Thus, the PSNR measures the quality performance of a video/image codec for a specific compression rate.

When considering whole video sequences, the PSNRs of the frames could be averaged to get a single number representing the quality of the whole sequence. However, we choose to set A to the three-dimensional set of pixels of the video sequence instead throughout this work. So, the PSNR is calculated by means of the mean squared error of all pixels.

2.4.2 Scalability

An important property of a video coding scheme is rate-scalability. Rate-scalability means that the bit-stream can be post-processed in order to reduce the bit-rate. This should result in lower visual quality. Alternatively, the spatial or temporal resolution could be reduced. The advantage of rate-scalability is that given for instance a broadcasted digital video stream, this stream can be forwarded onto a channel with smaller bandwidth without completely decoding and re-encoding the video stream. Rate-scalability is also known as progressive coding or embedded bit-streams.

While rate-scalability may be considered a solved problem in image compression as for JPEG-2000 and SPIHT, it is still a problem in video coding. Video standards as MPEG-2 or MPEG-4 include the possibility of dividing the bit-stream into quality layers. The first (base layer) is a low-quality approximation of the video sequence. Subsequent layers successively refine the reconstructed video. Unfortunately, it is known that each layer reduces the overall compression performance by up to 1dB. This is an important argument for video compression schemes that do not apply motion compensation because without motion compensation, rate-scalability techniques can be adopted from image compression without major modifications. This problem is addressed e.g. in [10].

2.4.3 Error Resilience

Another important property of video streams is error resilience. When a digital stream of video data is sent over an error-prone communication channel, arbitrary bits can be lost. This can corrupt the bit-stream, and the reconstructed video sequence could be destroyed. However, most video coding standards include methods to insert specific redundant data into the bit-streams which enable the decoder to reconstruct corrupted data. If this fails, the decoder must keep the error in the reconstructed frames as small as possible.

It is shown in [28] that 3-D wavelet transform based zero-tree coding (as used in this work) can be protected efficiently. This is done by exploiting the rate-scalability of the bit-stream. Low-quality parts of the bit-stream are protected strongly (high redundancy), while refinement information is protected more weakly to save bandwidth (unequal error protection).

Chapter 3

Wavelet Transform

3.1 Wavelet Analysis

Wavelet analysis has to cope with function spaces just as the Fourier analysis. Here we discuss a function space V that forms a Hilbert space with the following operations: V is a Hilbert space over $\mathbb{R}$ with

$$\langle f|g\rangle := \int_{-\infty}^{+\infty} f(t)g(t)dt$$
$$(f+g)(t) := f(t)+g(t)$$
$$(\lambda f)(t) := \lambda f(t)$$

The most commonly used function space is

$$L^2(\mathbb{R}) := \{f : \mathbb{R} \to \mathbb{R} \,|\, \int_{-\infty}^{+\infty} f^2(t)dt < \infty\}\,.$$

The problem is now to find suitable (orthonormal) bases for this space.

3.1.1 Fourier Transform (FT)

The most common base for $L^2(\mathbb{R})$ is $\{t \to e^{i\omega t} \,|\, \omega \in \mathbb{R}\}$. For this base, computing the coefficients

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dt$$

is called Fourier transform. Note that for real valued functions, $\hat{f}(\omega) = \overline{\hat{f}(-\omega)}$. Although the base functions are not elements of L^2, a reverse transform can be found and is given by

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \hat{f}(\omega)e^{i\omega t}d\omega\,.$$

For the subspace of τ-periodic functions (or functions with $\text{supp}(f) \subseteq [0, \tau]$), where typically $\tau = 2\pi$, it is sufficient to limit ω to $2\pi n$ where $n \in \mathbb{Z}$. Then the transform

$$\hat{f}(n) = \int_{0}^{2\pi} f(t) e^{-i2\pi n t} dt$$

gives the coefficients for the Fourier series

$$f(t) = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \hat{f}(n) e^{i2\pi n t} .$$

The Fourier coefficients are very important for frequency based operations. To be more precise, a linear time invariant operator is entirely specified by its eigenvalues $\hat{h}(\omega)$ and $e^{i\omega t}$ are the eigenvectors. Therefore, the domain of ω is called frequency domain and $|\hat{f}|$ is the frequency spectrum.

3.1.2 Short Time Fourier Transform (STFT)

However, there exist commonly used operations that are not time invariant. Because certain properties of signals are localised around a certain point in time, one has to find base functions that are well localised in time domain. On the other hand one does not want to lose the advantages of frequency localisation. Heisenberg shows that there cannot be a base function that has finite support in time domain as well as in the frequency domain. Therefore, one has to find a compromise.

One method is to use a so called time window g, that is concentrated around zero. By translating g along the time axis, the Fourier base functions can be *windowed* to give the new base functions (time frequency atoms)

$$g_{u,\xi}(t) = g(t-u) e^{i\xi t} .$$

The localisation of these functions in time and frequency domain can be controlled by the choice of g and is optimal if g is Gaussian. In this case the $g_{u,\xi}$ are called Gabor functions (Gabor atoms). The transform of a function f into the coefficient space (STFT – short time Fourier transform) is then given by:

$$Sf(u,\xi) = \langle f | g_{u,\xi} \rangle = \int_{-\infty}^{+\infty} f(t) \overline{g_{u,\xi}(t)} dt = \int_{-\infty}^{+\infty} f(t) g(t-u) e^{-i\xi t} .$$

3.1.3 Wavelets

The main disadvantage of this scheme is that, in many practical applications, single signals have shorter duration if they have higher frequency components. For this reason, time windows are preferred which shrink proportionally to the frequency scale of the base functions. It is then possible to retain the shape of the base functions at all translations and dilations. The base functions are then called wavelets. To be more precise, a so called mother wavelet is a function ψ with

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0$$

from which all wavelets can be derived by dilation with a scale parameter s and translation by u:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right) .$$

The wavelet transform is then given by

$$Wf(u,s) = \langle f|\psi_{u,s}\rangle = \int_{-\infty}^{+\infty} f(t)\overline{\frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right)}dt.$$

In most cases, $u \in \mathbb{Z}$ and $s = 2^j$, where $j \in \mathbb{Z}$. Thus, we can redefine

$$\psi_{i,j}(t) = \frac{1}{\sqrt{2^j}}\psi\left(\frac{t-2^j i}{2^j}\right) = \frac{1}{\sqrt{2^j}}\psi\left(\frac{t}{2^j} - i\right) .$$

$$Wf(i,j) = \langle f|\psi_{i,j}\rangle$$

3.2 Pyramidal Wavelet Transform

3.2.1 Multiresolution Analysis

The idea of approximating a given function successively by increasing the frequency scale of the time frequency atoms leads to the development of a sequence of subspaces $\{V_j\}_{j\in\mathbb{Z}}$ that give successive approximations of $L^2(\mathbb{R})$. Such a sequence of subspaces (multiresolution analysis) has to satisfy the following conditions:

$$\forall j \in \mathbb{Z}: \quad V_{j+1} \subset V_j$$

$$\forall i \in \mathbb{Z}: \quad f \in V_0 \Leftrightarrow f(\cdot - i) \in \mathbb{Z}$$

$$\forall j \in \mathbb{Z}: \quad f \in V_j \Leftrightarrow f\left(\frac{\cdot}{2}\right) \in V_{j+1}$$

$$\lim_{j\to+\infty} V_j = \{0\}$$

$$\lim_{j\to-\infty} V_j = L^2(\mathbb{R})$$

$$\exists \phi \in V_0 : \quad \{\phi(\cdot - i)\,|\,i \in \mathbb{Z}\} \quad \text{is an orthonormal basis of} \quad V_0$$

Now one can show that, for each multiresolution analysis, there exists an orthonormal basis of wavelets $\{\psi_{i,j}\}$, such that

$$V_{j-1} = V_j \oplus W_j,$$

where $W_j = [\{\psi_{i,j} \mid i \in \mathbb{Z}\}]$. Moreover, as ϕ can be constructed out of the $\phi_{-1,j}$ by

$$\phi = \sum_j h_j \phi_{-1,j} \, ,$$

where $h_n = \langle \phi | \phi_{-1,j} \rangle$ (because $\{\phi_{-1,j}\}$ is an orthonormal basis of $V_{-1} \supset V_0 \ni \phi$), ψ can be constructed in the same way:

$$\psi = \sum_j g_j \phi_{-1,j} \, .$$

These g_j turn out to be given by

$$g_j = (-1)^j h_{1-j} \, .$$

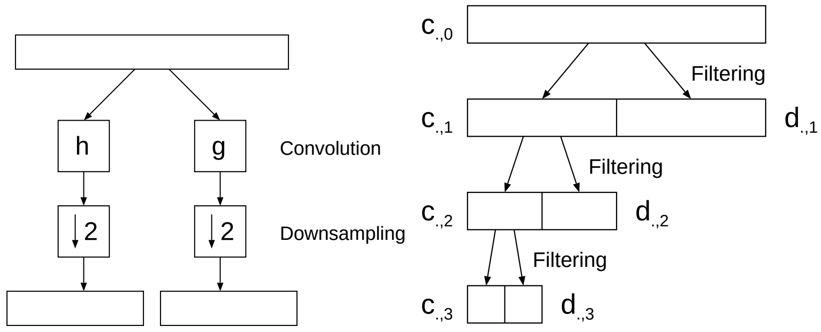
Note that $\{\phi(\cdot - i) \mid i \in \mathbb{Z}\}$ does not have to be an orthonormal basis. If it is a *Riesz basis*, the wavelet transform produces a stable but not perfect reconstruction (inverse transformation).

3.2.2 Fast Wavelet Transform (FWT)

The multiresolution analysis offers big advantages for practical implementations of the wavelet decomposition in the case that the wavelet has finite support. Let us first define $c_{i,j} = \langle f | \phi_{i,j} \rangle$ and $d_{i,j} = \langle f | \psi_{i,j} \rangle$, so that the wavelet transform is the mapping $f \mapsto d$. If ϕ and ψ are part of a multi resolution analysis as defined above, the projection of f onto the subspace V_j is given by $d_{\cdot,j}$. Because of the properties of multiresolution analysis, the coefficients of the next coarser level $(j+1)$ are given by

$$c_{i,j+1} = \sum_{k \in \mathbb{Z}} c_{2i+k,j} h_k$$
$$d_{i,j+1} = \sum_{k \in \mathbb{Z}} c_{2i+k,j} g_k.$$

The $d_{\cdot,j}$ represent a data sequence, that approximates the function f. The approximation is coarser if j is higher. The given decomposition step is essentially a convolution of the data sequence $d_{\cdot,j}$ with the sequences h and g followed by a down-sampling by a factor of 2. The whole decomposition process is then completed by continuing recursively to decompose the sequence $c_{\cdot,j+1}$ in the same way. This is illustrated in Figure 3.1. The sequences h and g are called low-pass and high pass filters respectively because they *divide* the original *signal* $c_{\cdot,j}$ into a low frequency part $(c_{\cdot,j+1})$ and a high frequency part $(d_{\cdot,j+1})$. Note that $d_{i,j}$ is



(a) Filtering (b) Decomposition

Figure 3.1: The fast wavelet transform

the coefficient for the wavelet $\psi_{i,j}$, which contains lower frequency components as j increases. Because of their special properties, h and g are called quadrature mirror filters (QMF).

If the wavelet ψ has finite support, then also only a finite number of filter coefficients are non-zero. This is what makes the FWT fast (see Section 3.2.4).

3.2.3 2-D and 3-D Wavelet Transform

Let us first look at the 2-D case. A 2-D wavelet base for $L^2(\mathbb{R})$ is constructed with the 1-D wavelet ψ and the scaling function ϕ by

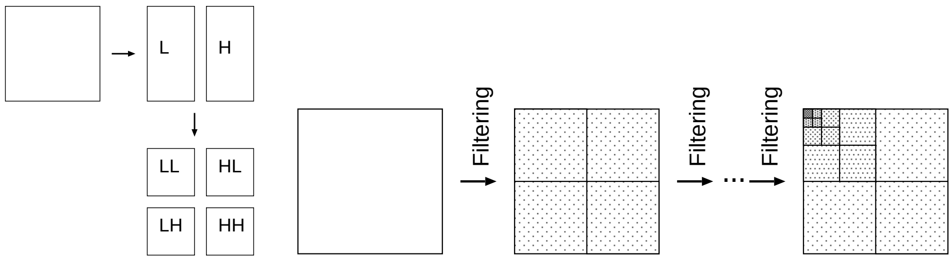
$$\begin{aligned} \psi^1(\boldsymbol{x}) &= \psi(x_1)\phi(x_2), \\ \psi^2(\boldsymbol{x}) &= \phi(x_1)\psi(x_2) \quad \text{and} \\ \psi^3(\boldsymbol{x}) &= \psi(x_1)\psi(x_2), \end{aligned}$$

where $\boldsymbol{x} = (x_1, x_2)$ and the derived wavelets are

$$\psi^k_{\boldsymbol{i},j}(\boldsymbol{x}) = 2^j \psi^k(2^j \boldsymbol{x} - \boldsymbol{i}) \quad \boldsymbol{i} \in \mathbb{Z}^2 \ .$$

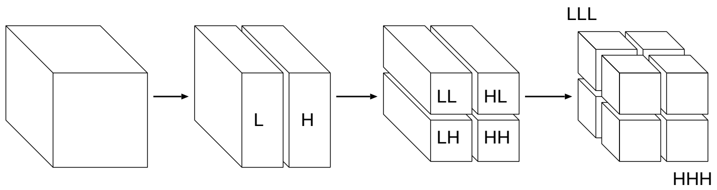
The fast wavelet transform (in terms of QMF) can then be implemented by a column-wise 1-D filtering followed by a row-wise 1-D filtering resulting in four new sub-bands as shown in Figure 3.2(a). The sub-band that results from low-pass filtering in both dimensions is called approximation sub-band and is decomposed in the same way, leading to the pyramidal decomposition structure as shown Figure 3.2(b). Each of the other sub-bands correspond to the wavelets ψ^1, ψ^2, ψ^3.

The 3-D case is analogous. There are seven wavelets ψ^k varying in three variables t, x, y. The original data (a cuboid) is filtered in three steps, one for each dimension, as shown in Figure 3.3(a). The approximation sub-band is decomposed further just as in the 2-D case (see Figure 3.3(b)).

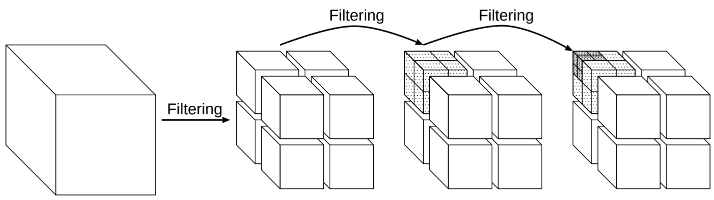


(a) Filtering (b) Decomposition

Figure 3.2: Fast wavelet transform of 2-D data



(a) Filtering



(b) Decomposition

Figure 3.3: Fast wavelet transform of 3-D data

3.2.4 Complexity

The number of sub-bands for a maximum decomposition level l and dimension d (including all intermediate sub-bands) is

$$S_P = 1 + l2^d .$$

The number of wavelet coefficients (not counting the initial sub-band, i.e. the source data) is

$$C_P = n \sum_{k=0}^{l-1} 2^{-dk} = n \frac{1 - 2^{-dl}}{1 - 2^{-d}} .$$

If we set the decomposition depth l to the logarithm of the size of the data in one direction ($= \sqrt[d]{n}$) then $l = \log_2 \sqrt[d]{n} - c = \frac{1}{d} \log_2 n - c$ where c makes the decomposition stop before the sub-bands get too small (sub-bands will contain at least 2^{cd} coefficients). This makes

$$C_P = n \frac{1 - 2^{-c}}{1 - 2^{-d}} = O(n)$$

To investigate the computational complexity, let us look at the number of multiplies necessary to compute the WT. Because each coefficient can be calculated by an inner product of a filter with another sub-band, the work (number of multiplies) is $W_P = dfC_P$, where f is the filter length. If the low- and high-pass filter do not have equal lengths, it is sufficient to set f to the average: $f = \frac{f_L + f_H}{2}$ because the WT always requires equal amounts of low- and high-pass filterings.

This shows that the FWT is able to reduce the computational complexity from $O(n \log n)$ for direct convolution with the wavelet functions to linear complexity $O(n)$. Note that the fast Fourier transform (FFT) manages to reduce the complexity from $O(n^2)$ to $O(n \log n)$.

3.3 Wavelet Packets

Wavelet packets are a generalisation of wavelets. The idea is based on the following fact: If $\{\theta(\cdot - i) \,|\, i \in \mathbb{Z}\}$ is an orthonormal basis for a space U and g, h a pair of conjugate mirror filters, then

$$\left\{ \sum_{k \in \mathbb{Z}} h_k \theta(\cdot - k - 2i) \,|\, i \in \mathbb{Z} \right\} \cup \left\{ \sum_{k \in \mathbb{Z}} g_k \theta(\cdot - k - 2i) \,|\, i \in \mathbb{Z} \right\}$$

is also an orthonormal basis of U.

This means that we can also divide the high frequency sub-bands into two sub-bands just as with the approximation sub-band. Because these new sub-bands can also be divided further, we get an entire decomposition tree as shown in Figure 3.4

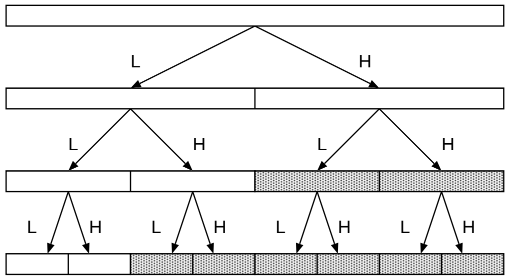


Figure 3.4: Wavelet packet decomposition

3.3.1 Best Basis

As we can now choose which basis we want to use for the transform, i.e. which sub-band should be decomposed and which not, we have to find some criterion to decide this. There are different aspects of the data that are important for different applications of wavelet transform. So it is best to define an abstract cost function cost : $L^2(\mathbb{Z}) \to \mathbb{R}$ that determines a measure of (negative) usability for a given data sequence with respect to a given purpose.

The basis can be represented by a decomposition tree T. The vertices of T represent the sub-bands where the root vertex is the original data sequence s and the leaves are the sub-bands that form the transformed data c. Now we can define the best basis as the tree T that minimises $\operatorname{cost}(c)$.

As an example let us look at data compression applications. A cost function in this case should measure the concentration of information, i.e. it should give large values when all but a few coefficients are negligible. A well known example for this is the *entropy*:

$$H(c) = -\sum_i p_i \log p_i, \quad \text{where} \quad p_i = \frac{|c_i|^2}{\|c\|^2}\,.$$

Because the number of possible trees is of an order of about $O(2^{n/2})$ in the 1-D case if the maximum decomposition depth is of order $O(\log_2 n)$, it would mean a lot of work to test all of them to find the best one. Fortunately, there is another possibility [9]: If the cost function is additive, i.e. $\operatorname{cost}(c^1 \cup c^2) = \operatorname{cost}(c^1) + \operatorname{cost}(c^2)$, then the branching decision at a certain vertex of the tree depends only on the vertices within the corresponding subtree. To be more precise: The tree has to branch for sub-band a if $\operatorname{cost}(a) > \operatorname{bestcost}(a)$, where

$$\operatorname{bestcost}(a) := \begin{cases} \operatorname{cost}(a) & \text{max. decomposition depth reached} \\ \operatorname{detcost}(a) & \operatorname{detcost}(a) < \operatorname{cost}(a) \\ \operatorname{cost}(a) & \operatorname{detcost}(a) \geq \operatorname{cost}(a) \end{cases}$$

and

$$\operatorname{detcost}(a) := \sum_{b \in \operatorname{child}(a)} \operatorname{bestcost}(b)\,.$$

Unfortunately, the entropy H from above is not additive. However, there exists an additive analogue $\lambda(c) = -\sum_i c_i^2 \log c_i^2$ which satisfies the same conditions. Note that the same algorithm applied to non additive cost functions is referred to as *near best basis* algorithm. In this case detcost has to be defined as

$$\operatorname{detcost}(a) := \operatorname{cost}\left(\bigcup_{b \in \operatorname{child}(a)} \operatorname{bestdecomp}(b)\right)$$

where bestdecomp(b) is the "near best basis" decomposition of b using the same procedure recursively.

3.3.2 Cost Functions

Throughout this work, the following (additive) cost functions will be used:

Entropy information cost

$$\operatorname{cost}(c) = -\sum_i c_i^2 \log c_i^2$$

Logarithmic energy

$$\operatorname{cost}(c) = \sum_i \log c_i^2$$

Norm

$$\operatorname{cost}(c) = \sum_i |c_i|$$

3.3.3 Complexity

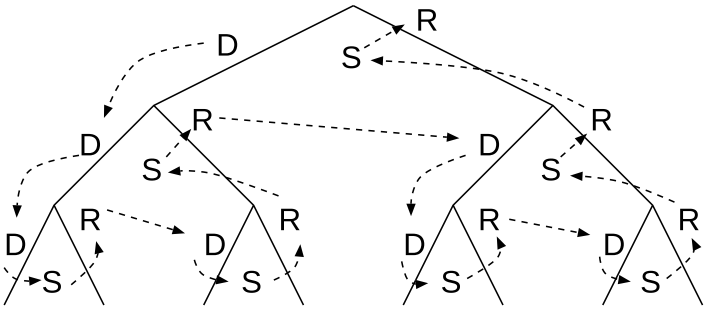
The number of sub-bands (including all intermediate sub-bands) for a full wavelet packet decomposition is

$$S_{WP} = \sum_{k=0}^{l} 2^{dk} = \frac{2^{d(l+1)} - 1}{2^d - 1} = n \frac{2^{-(c-1)}}{1 - 2^{-d}} .$$

As in Section 3.2.4, n is the data size, d is the dimension, l is the decomposition depth and c limits the decomposition depth l when l depends logarithmically on the data size. The number of all possible coefficients of a WP decomposition is

$$C_{WP} = nl = n\left(\frac{\log_2 n}{d} - c\right) = O(n \log n) .$$

The complexity of a full wavelet packet transform is, similar to the pyramidal case, $W_{WP} = dfC_{WP}$. The complexity of the best-basis algorithm is determined by the full wavelet transform, which is necessary to compute the cost for all



D = Backup sub-band and Decompose
S = Sum the cost of sub-bands
R = Restore sub-band if decomposition does not reduce the cost

Figure 3.5: Best basis decomposition without a second run and minimised memory demand

sub-bands. After the best basis is found, the proper sub-bands can be generated by either re-performing the corresponding transform (second-run approach) or by selecting them from the buffer used during the full decomposition. The first approach leads to additional computation time and the second approach has a very high memory demand. Figure 3.5 shows an alternative approach [31] which avoids a second run and has limited extra memory demand of

$$n(1 + 2^{-d} + 2^{-2d} + \ldots) \approx n\frac{2^d}{2^d - 1}\,.$$

3.4 Anisotropic Wavelet Packets

We have so far considered only those multidimensional wavelets which have the same scale in each direction. These conventional wavelet packets can therefore be called isotropic wavelet packets (or wavelet packets with isotropic dilations [66]). If we consider bases of wavelet packets whose scales are different in different directions, then the number of sub-bands and the number of bases increases substantially.

Additionally, the method of representing specific bases by a decomposition tree (including all intermediate sub-bands) is not applicable any more. As filtering steps can be interchanged almost arbitrarily, more than one *decomposition path* can be found for most sub-bands. For instance: $L_h H_v$ and $H_v L_h$, where L_h denotes horizontal low-pass filtering and H_v vertical high-pass filtering, are two paths that produce the same sub-band.

To avoid ambiguous representations of equal bases and redundant calculations, we introduce a new graph structure that is a generalisation of trees. This structure substitutes the wavelet packet decomposition trees for anisotropic wavelet packets. Because of its interwoven nature, we will call it *bush*.

3.4.1 Bushes

In general, we introduce a bush as a special coloured graph

$$B = (V, E, C) \quad E \subseteq V \times V \times C$$

where V is a set of vertices (or nodes) and C is a set of colours. E is the set of edges which connects two vertices and is associated with a certain colour. We will call the number of colours *dimension.* We write

$$a \overset{c}{\sim} b :\Leftrightarrow (a, b, c) \in E\,.$$

The edges should be non-directional:

$$a \overset{c}{\sim} b \Rightarrow b \overset{c}{\sim} a\,.$$

To be a bush, the coloured graph B has to satisfy some conditions. Before we state these conditions, we have to introduce a mapping (projection) of a coloured graph to a colourless graph that contracts certain vertices to a single vertex. The aim is to eliminate all but one colour. Therefore, every two vertices that are connected by a path that does not include edges with the specific colour should be contracted.

Definition 3.4.1. *Let $B = (V, E, C)$ be a coloured graph. Let $G = (V', E')$ be a complete uncoloured graph with at least as many vertices as B. For each colour $c \in C$, a* bush projection *is a mapping*

$$f_c : V \longrightarrow V'$$

which satisfies

$$a \overset{d}{\sim} b \wedge d \neq c \Rightarrow f_c(a) = f_c(b)\,.$$

Now, we can define the projection of B:

$$f_c(E) := \{(f_c(a), f_c(b)) \mid a \overset{c}{\sim} b\}$$

$$f_c(B) := (f_c(V), f_c(E))$$

This means that $f_c(B)$ is an "orthogonal" projection of the coloured graph in the sense that each projection represents a distinct aspect of the structure of the coloured graph. Moreover, the set of projections can serve as a base to identify elements of bushes by building tuples of elements of the projections, as we will see (Theorem 3.4.1). This is also an inherent property of orthogonality.

Now, we will define the conditions for a bush: Firstly, the projections must be trees. Secondly, if all projections of two vertices a, b are equal then a and b are equal (while the reverse is trivial).

Definition 3.4.2. *A* bush *$B = (V, E, C)$ is a finite coloured graph such that for each colour c there exists a bush projection f_c with:*

- $\forall c \in C : f_c(B)$ *is a tree and*
- *if* $\forall c \in C : f_c(a) = f_c(b)$ *then* $a = b$

The second condition states that the projections of a vertex are sufficient to localise the vertex. The question is now if and how whole bushes can be characterised by their projected trees. The following theorem gives the answer. Before that, we need two definitions, though.

Definition 3.4.3. *Let $T_1, \dots, T_m$ be trees, where $T_i = (V_i, E_i)$. The* product *of trees is*

$$T_1 \times T_2 \times \ldots \times T_m := (V, E, C)$$

where

$$V := V_1 \times \ldots \times V_m$$

$$C := \{1, 2, \dots, m\}$$

$$E := \{(a, b, c) \,|\, a, b \in V\ , \forall c' \neq c : a_{c'} = b_{c'}\ , (a_c, b_c) \in E_c\}$$

Definition 3.4.4. *Let $B_1 = (V_1, E_1, C), B_2 = (V_2, E_2, C)$ be coloured graphs. We write*

$$B_1 \subseteq B_2 :\Leftrightarrow V_1 \subseteq V_2 \wedge E_1 \subseteq E_2\,.$$

Theorem 3.4.1. *B be a connected coloured graph with dimension m. Then B is a bush if and only if there exist m finite trees $T_1, \dots, T_m$ such that $B \subseteq T_1 \times T_2 \times \ldots \times T_m$ (or isomorphic to such a subset).*

Proof. We have to prove two directions:

$\boxed{\Rightarrow}$ We set $T_c := f_c(B)$ and show that f_c is an injective mapping from vertices and edges to $T_1 \times \ldots \times T_m$. (Injectivity is sufficient because only a subset is required). From the definition of a bush, we see that no two vertices can have equal projections for all colours. Thus, $f_c(V)$ is injective. The projection of an edge $a \overset{c}{\sim} b$ gives us: $f_c(a) \sim f_c(b)$ and $f_{c'}(a) = f_{c'}(b)$ for all $c' \neq c$. Thus, the projected edges are elements of $T_1 \times \ldots \times T_m$. Furthermore, no two edges have the same projections for all colours because their two vertices would have to be the same (see above) and two vertices cannot be connected by two colours at the same time.

$\boxed{\Leftarrow}$ We set $f_c((a_1, a_2, \dots, a_m)) := a_c$. Clearly, $f_c(B)$ is a subgraph of T_c. $f_c(B)$ is connected because B is connected and, therefore, a tree. The second condition for bushes is also true because:

$$\begin{aligned} \forall c \in C : f_c(a) = f_c(b) \quad &\Rightarrow \quad \forall c \in C : a_c = b_c \\ &\Rightarrow \quad a = b\,. \end{aligned}$$

$\square$

Figure 3.6: Examples to visualise the second condition for rooted bushes. The left example is not a correct rooted bush because vertex a does not have a parent connected to it. The right example is correct.

An important class of trees are rooted trees and n-ary trees. We will now introduce similar classes for bushes.

Definition 3.4.5. *A* rooted bush B *is a bush with a single special vertex r that is called root and:*

- $\forall c \in C : f_c(r)$ *is the root of* $f_c(B)$,
- $\forall b \neq r : \exists a, c : a \overset{c}{\sim} b \wedge f_c(a) = \mathrm{parent}(f_c(b))$.

The second condition assures the existence of a parent for each vertex. This means that there is a path from the root to each vertex which maps to a path in all $f_c(B)$. (See Figure 3.6.)

Definition 3.4.6. *An* n-ary bush *is a rooted bush where*

- $\forall c \in C : f_c(B)$ *is an n-ary tree,*
- *if* $\exists b_1, b_2, k_1, k_2, c_1 \neq c_2 : \mathrm{child}(a, c_1, k_1) = b_1 \wedge \mathrm{child}(a, c_2, k_2) = b_2$
  then $\exists d : \mathrm{child}(b_1, c_2, k_2) = \mathrm{child}(b_2, c_1, k_1) = d$,
- *for all colours* $c_1, c_2,\ c_1 \neq c_2,$ *and* $m = 1 \dots n$*:*
  if $\forall k = 1 \dots n : \exists d = \mathrm{child}(\mathrm{child}(a, k, c_1), m, c_2)$
  then $\exists e = \mathrm{child}(a, m, c_2)$,

where, similar to n-ary trees, we define

$$
\begin{aligned}
\mathrm{child}(a, c, k) = b \quad &:\Leftrightarrow \quad a \overset{c}{\sim} b \wedge \mathrm{child}(f_c(a), k) = f_c(b) \\
\mathrm{parent}(b, c) = a \quad &:\Leftrightarrow \quad a \overset{c}{\sim} b \wedge \mathrm{parent}(f_c(b)) = f_c(a)
\end{aligned}
$$

where $\mathrm{child}(a, k)$ *and* $\mathrm{parent}(a)$ *give the k-th child and the parent of a vertex respectively in an n-ary tree.*

There are incorrect n-ary bushes which only satisfy the first condition (the projections are n-ary trees) but violates at least one of the two other conditions. See Figure 3.7 for a visualisation of correct and incorrect bushes. The first example is incorrect because it violates the second condition. The third example violates the third condition.

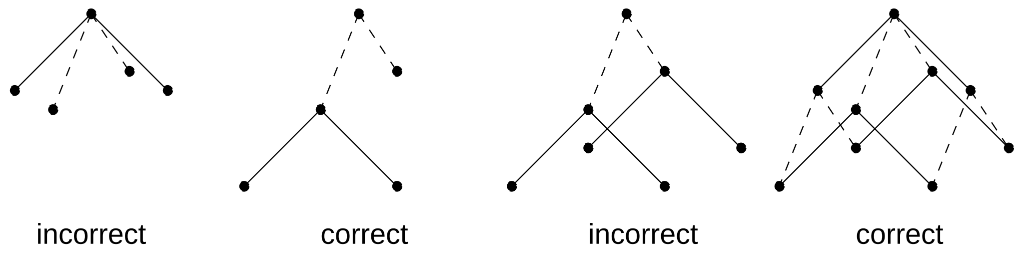


Figure 3.7: Correct and incorrect binary bushes with dimension 2.

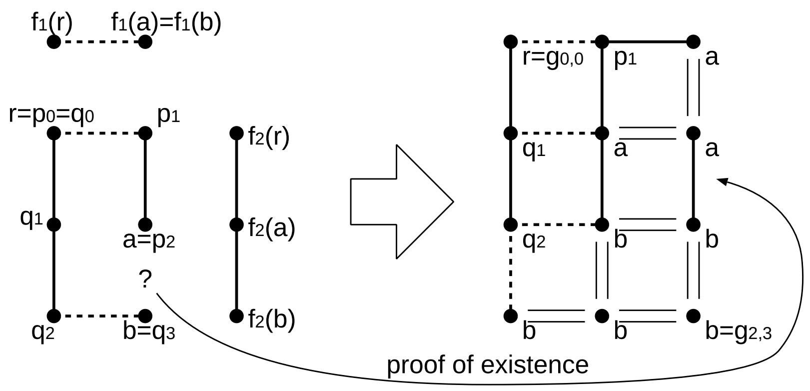


Figure 3.8: Example visualisation of the proof method for the existence of a path of descendants between two vertices whose projections stand in descendant relationship

Apart from child and parent, we also need the following functions for the direct offspring, the set of all descendants of a vertex and the set of leaves or terminal nodes of a bush.

Definition 3.4.7. *B be a bush, $a \in B$ and r be the root of B.*

$$\begin{aligned}
\mathrm{off}(a) &:= \{b \mid b = \mathrm{child}(a, c, k), c \in C, k = 1 \dots n\} \\
\mathrm{desc}(a) &:= \mathrm{off}(a) \cup \bigcup_{b \in \mathrm{off}(a)} \mathrm{desc}(b) \\
\mathrm{leaves}(a) &:= \{b \,|(b \in \mathrm{desc}(a) \vee b = a) \wedge \mathrm{off}(b) = \emptyset\} \\
\mathrm{leaves}(B) &:= \mathrm{leaves}(r)
\end{aligned}$$

The following theorem shows which vertices are connected and which are not.

Theorem 3.4.2. *Let B be a bush, $a, b \in B$. If $\forall c : f_c(b) \in \mathrm{desc}(f_c(a)) \vee f_c(b) = f_c(a)$ then $b \in \mathrm{desc}(a) \vee b = a$.*

Proof. There is a path $P := (p_0, p_1, \dots, p_m)$ from the root $r = p_0$ to $a = p_m$ and a path $Q := (q_0, q_1, \dots, q_m)$ from $r = q_0$ to $b = q_n$. The premise is that

$\forall c : f_c(P)$ is a sub-path of $f_c(Q)$. Now we define recursively the 2-D sequence $g_{k,l}$ of vertices (see Figure 3.8):

- $g_{k,0} := p_k$, $g_{0,l} := q_l$
- $k > 0 \wedge l > 0$:
  - $g_{k-1,l} \neq g_{k-1,l-1} \wedge g_{k,l-1} \neq g_{k-1,l-1}$
    $\Rightarrow g_{k-1,l} = \text{child}(g_{k-1,l-1}, c, i) \wedge g_{k,l-1} = \text{child}(g_{k-1,l-1}, d, j)$
    - ▷ $c = d$:
      - ◇ $i = j$: $g_{k,l} := g_{k-1,l} = g_{k,l-1}$
      - ◇ $i \neq j$: not possible (see later)
    - ▷ $c \neq d$: $\exists h = \text{child}(g_{k-1,l}, d, j) = \text{child}(g_{k,l-1}, c, i)$
      $g_{k,l} := h$.
  - $g_{k-1,l} = g_{k-1,l-1}$: $g_{k,l} := g_{k,l-1}$
  - $g_{k,l-1} = g_{k-1,l-1}$: $g_{k,l} := g_{k-1,l}$

Now we show that for all paths $\{1, \ldots, s\} \longrightarrow \mathbb{N}^2, t \mapsto (k_t, l_t)$ where $k_1 = l_1 = 0$ and either $k_t = k_{t-1} + 1 \wedge l_t = l_{t-1}$ or $k_t = k_{t-1} \wedge l_t = l_{t-1} + 1$, the path $t \mapsto f_c(g_{k_t,l_t})$ is a sub-path of $f_c(Q)$. We show that by induction over the length s of the sequence:

- The two cases $k \equiv 0$ and $l \equiv 0$ lead to sub-paths of P and Q respectively.
- Consider $g_{k,l}$ where $k > 0, l > 0$. From the induction hypothesis we know that for all paths ending in $(k, l-1)$ or $(k-1, l)$ the projections of g_{k_t,l_t} are sub-paths of $f_c(Q)$ (which includes paths with $(k-1, l-1)$ as last but one element). One can easily see that for a sequence ending in (k, l) the same is true by checking all cases above. Note that the case $c = d, i \neq j$ is not possible because it would be a contradiction to the induction hypothesis.

So we see that $g_{m,n} = b$ because the path from $f_c(g_{0,n}) = f_c(b)$ to $f_c(g_{m,n})$ must be a sub-path of $f_c(Q)$ and, thus, be constant. In this way, we have also constructed a path from $a = g_{m,0}$ to $b = g_{m,n}$, which shows that $b \in \text{desc}(a)$. □

A corollary of the theorem is that if $f_c(b) = \text{child}(f_c(a), k)$ and $\forall c' \neq c : f_{c'}(b) = f_{c'}(a)$ then $b = \text{child}(a, c, k)$ (and vice versa, of course). Another corollary is the following theorem.

Theorem 3.4.3. *Let $B_1 = (V, E_1, C)$ and $B_2 = (V, E_2, C)$ be two n-ary bushes which share one set of projection functions f_c. Then $E_1 = E_2$ and, therefore, $B_1 = B_2$.*

Proof.

$$\begin{aligned} & (a, b, c) \in E_1 \\ \Leftrightarrow\ & f_c(b) = \text{child}(f_c(a), k) \wedge \forall c' \neq c : f_{c'}(b) = f_{c'}(a) \\ \Leftrightarrow\ & (a, b, c) \in E_2 \end{aligned}$$

$\square$

Definition 3.4.8. *A* full n-ary bush *is an n-ary bush where each vertex has either 0 or exactly n children referring to a single colour. We write*

$$\mathrm{haschildren}(a,c) :\Leftrightarrow \exists\, \mathrm{child}(a,c,1) \Leftrightarrow \cdots \Leftrightarrow \exists\, \mathrm{child}(a,c,n)\,.$$

Full n-ary bushes are an important class of n-ary bushes not only because anisotropic wavelet packet sub-bands can be arranged in full binary bushes (as we will see), but also because full bushes are uniquely determined by the set of leaf vertices. The following two theorems will show this.

Theorem 3.4.4. *Let B be a full n-ary bush. Then*

$$f_c(a) \notin f_c(\mathrm{leaves}(a)) \Leftrightarrow \mathrm{haschildren}(a,c)$$

Proof. We have to prove two directions:

$\boxed{\Rightarrow}$ We use induction over the maximum distance of a to its leaves. a is not a leaf vertex because $f_c(a) \notin f_c(\mathrm{leaves}(a))$. Thus, $\exists d : \mathrm{haschildren}(a,d)$.

- If the distance is 1, then $d = c$ and we are done.
- If the distance is greater than 1: Again, If $d = c$ we are done. Otherwise, we know by induction that $\forall k : \mathrm{haschildren}(\mathrm{child}(a,d,k),c)$. Because of the third condition for n-ary bushes, we get $\mathrm{haschildren}(a,c)$.

$\boxed{\Leftarrow}$ Consider a vertex $b \in \mathrm{desc}(a)$ such that $f_c(b) = f_c(a)$. If there is no such vertex at all, there cannot be a leave vertex with that property (a is not a leave vertex) and we are done. Otherwise, all we have to show is that b cannot be a leaf vertex. There must be a path from a to b: $b_0 = a, b_1, \ldots, b_s = b$. Since $f_c(b) = f_c(a)$, all d with $b_i = \mathrm{child}(b_{i-1},d,k)$ must not be equal to c. The second condition for bushes guarantees that

$$\mathrm{haschildren}(b_0,c) \Rightarrow \mathrm{haschildren}(b_1,c) \Rightarrow \ldots \Rightarrow \mathrm{haschildren}(b_s,c)\,.$$

Thus, $b \notin \mathrm{leaves}(a)$. Therefore, $f_c(a) \notin f_c(\mathrm{leaves}(a))$. $\square$

Theorem 3.4.5. *Let $T_1, \ldots, T_m$ be n-ary trees and $B_1, B_2 \subseteq T_1 \times \ldots \times T_m$ two full n-ary bushes. Then*

$$\mathrm{leaves}(B_1) = \mathrm{leaves}(B_2) \Rightarrow B_1 = B_2\,.$$

Proof. We will show that if a vertex is element of one bush, it is also element of the other. We apply induction over the distance of a vertex a from the root. The basis case for distance 0 is the root vertex which is, of course, unique and present in B_1 and B_2. For the induction step, consider a vertex $b \in T_1 \times \ldots \times T_m$. If no parent of b is in B_1 then also no parent is in B_2 (induction hypothesis), so

b can neither be in B_1 nor in B_2. Otherwise, $\exists a = \text{parent}(b, c), a \in B_1 \wedge a \in B_2$. We get:

$$\begin{aligned}
& b \in B_1 \\
\Leftrightarrow\ & \text{haschildren}(a, c) \quad \text{in}\, B_1 \\
\Leftrightarrow\ & f_c(a) \notin f_c(\text{leaves}(a)) \quad \text{in}\, B_1 \\
\Leftrightarrow\ & f_c(a) \notin f_c(\text{leaves}(a)) \quad \text{in}\, B_2 \\
\Leftrightarrow\ & \text{haschildren}(a, c) \quad \text{in}\, B_2 \\
\Leftrightarrow\ & b \in B_2
\end{aligned}$$

□

Unfortunately and *n*ot according to trees, a similar theorem cannot be stated for non-full n-ary bushes. This can easily be seen by looking at a simple example of two bushes with three vertices r, a, b and two colours c, d: $r \overset{c}{\sim} a \overset{d}{\sim} b$ and $r \overset{d}{\sim} a \overset{c}{\sim} b$ are two correct n-ary bushes with equal sets of leaves $\{b\}$ and equal $f_c(b)$ and $f_d(b)$.

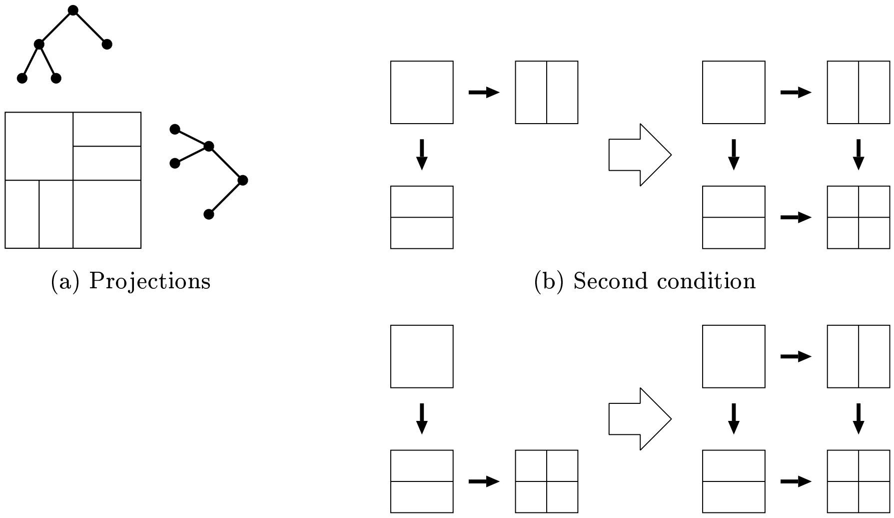
As opposed to trees, there is not a unique way to walk through a bush and to visit each leaf vertex exactly once. It is therefore important to find a proper way to do this. One possibility is to construct a spanning tree with a set of leaf vertices equal to that of the bush and to perform the usual tree-walk on it. The following theorem shows how to construct such a tree in a full n-ary bush.

Theorem 3.4.6. *Let B be a bush. For each non-leaf vertex a we choose an arbitrary colour c_a such that* $\text{haschildren}(a, c_a)$. *$T$ be the n-ary tree with the same root as B and* $\text{child}(a, k) := \text{child}(a, c_a, k)$. *Then* $\text{leaves}(T) = \text{leaves}(B)$.

Proof. Clearly, $\text{leaves}(T) \subseteq \text{leaves}(B)$. If r is the root of B, then $r \in T$. Consider $b \in leaves(B)$. If $b = r$ then $b \in leaves(T)$ and we are done. Otherwise, we know from Theorem 3.4.4 that $f_{c_r}(r) \notin f_{c_r}(\text{leaves}(r))$ and therefore $f_{c_r}(b) \neq f_{c_r}(r)$. This means that $\exists k : b \in \text{leaves}(\text{child}(r, c_r, k))$ (see Theorem 3.4.2). Continuing with the same argumentation on $\text{child}(r, c_r, k)$ instead of r ($\text{child}(r, c_r, k)$ is element of T and B) leads to $b \in leaves(T)$. Thus, $\text{leaves}(B) \subseteq \text{leaves}(T)$. □

An important question is how easy it is to implement a bush structure in a computer program. Let us look at the definition of the n-ary bush and Definition 3.4.3. The first condition for n-ary bushes together with Definition 3.4.3 shows that the vertices can be identified by a vector of tree vertices. Vertices of n-ary trees can be identified by so-called heap numbers, i.e. the root gets 0 and the children of a node get $n \cdot h + 1$ to $n \cdot h + n$ where h is the heap number of the parent. Thus, for an n-ary bush with dimension m we need vectors of m heap numbers to identify the bush's vertices.

Fortunately, the second and third condition for n-ary bushes are *l*ocal conditions, i.e. local modifications of a bush (inserts and removals of vertices) can



(a) Projections

(b) Second condition

(c) Third condition

Figure 3.9: Anisotropic wavelet packets form bushes of sub-bands

be handled locally. Secondary modifications necessary to retain the correctness of the bush can make further modifications necessary, though. This leads to a chain of modifications which can be implemented by recursive correctness repair procedures.

Finally, the condition for rooted bushes, stating that each vertex (except for the root) must have a parent, can be handled by always inserting vertices only as child of an existing vertex. After vertex removal, all vertices which lose all their parents must be removed.

3.4.2 Bushes of Anisotropic Sub-Bands

Sub-bands of wavelet packets with anisotropic dilations are generated by a sequence of horizontal or vertical (or higher dimensional) low-pass or high-pass filterings each followed by a 1-D down-scaling in the respective direction. Unfortunately, these sequences (or paths) are not unique, i.e. more than one sequence can generate the same sub-band. Therefore, the graph that connects all possible sub-bands, where two sub-bands are connected if one can be generated by 1-D filtering of the other, cannot be a tree because it contains cycles. We will now show that this graph is a correct binary bush.

We choose the dimensions as the set of colours. Now, we have to show that the conditions of Definition 3.4.2 are satisfied. One can see that the projection functions produce a 1-D decomposition tree which is a binary tree (see Figure 3.9(a)). Thus, the first condition is satisfied. If a sub-band is filtered in two distinct directions (high- and low-pass), then four sub-bands are created, all filtered in both directions. This is the behaviour formalised in the second condition for

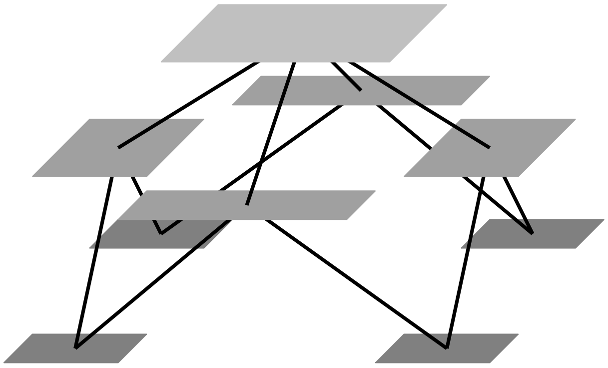


Figure 3.10: A full bush of anisotropic wavelet packets with decomposition depth 1 in each direction

n-ary bushes (see Figure 3.9(b)). The third condition is not strictly necessary but plausible. It means that if both sub-bands produced by filtering in one direction are filtered in another direction, the result could as well be achieved if the order of filtering would be exchanged. So, we also include the two sub-bands into the decomposition structure that are produced by filtering the original sub-band in the second direction first (see Figure 3.9(c)). Moreover, the third condition guarantees a unique representation of the decomposition structure. Therefore, the third condition is considered satisfied.

Because the projections are binary trees and each filtering produces exactly two sub-bands, we are dealing here with full binary bushes. Figure Figure 3.10 shows an example of a bush of anisotropic wavelet packets.

3.4.3 Best Basis

For isotropic wavelet packets, we took care to integrate the best basis decomposition (BBDec) into the best basis determination (BBDet). However, as BBDet involves calculating all possible sub-bands for anisotropic wavelet packets and BBDec does not, BBDet has higher complexity than BBDec. Therefore, it is not of high importance to integrate BBDec into BBDet. Moreover, it is not even possible without enormous memory demand. The reason is that, on the one hand, all descendant sub-bands have to be known to find the best basis of a sub-band and, on the other hand, each sub-band has to be kept in memory until all ancestors (and there is more than one parent) have decided not to include the sub-band in their best basis. Therefore, many sub-bands have to be kept in memory before all decisions depending on them are made and they can be dropped. A detailed discussion would be too extensive for the purpose of this work and is, therefore, not included here. Anyway, we choose the conventional second-run approach for best basis decompositions.

- First, we have to calculate the cost for each sub-band. This is done by generating all sub-bands as in Pseudo code 2 and calculating the cost for each sub-band immediately after it is created.

- Then, we determine for each sub-band the best cost and the best decomposition dimension by:

$$\begin{aligned}
\mathrm{bestcost}(S,k) &:= \begin{cases} \mathrm{cost}(S) & k=0 \\ \mathrm{bestcost}(\mathrm{child}(S,k,1)) + & \\ \quad \mathrm{bestcost}(\mathrm{child}(S,k,2)) & k=1\dots n \end{cases} \\
\mathrm{bestdim}(S) &:= \operatorname*{argmin}_{k=0\dots n} \mathrm{bestcost}(S,k) \\
\mathrm{bestcost}(S) &:= \mathrm{bestcost}(S, \mathrm{bestdim}(S))
\end{aligned}$$

  This is a recursive procedure ending at leaf sub-bands (sub-bands at maximum decomposition level).

- The best basis decomposition is then performed based on original data by applying a tree-walk on a spanning tree as in Theorem 3.4.6. This leads to the algorithm in Pseudo code 1.

```
BestDecomp (S) :=
  if bestdim(S) > 0
    Filter (S, bestdim(S))
    BestDecomp (child(S, bestdim(S), 1))
    BestDecomp (child(S, bestdim(S), 2))
```

Pseudo code 1: Best basis decomposition based on a spanning tree walk. Leaf-vertices are indicated by bestdim(S)=0

3.4.4 Complexity

The number of possible sub-bands is:

$$S_{AP} = \left(\sum_{k=0}^{l} 2^k \right)^d = (2^{l+1} - 1)^d \approx 2^{-d(c-1)} n \,.$$

As in Section 3.2.4, n is the data size, d is the dimension, l is the decomposition depth and c limits the decomposition depth l when l depends logarithmically on the data size.

As opposed to isotropic wavelet packets, performing a full decomposition and calculating all possible sub-bands is not the same task for anisotropic wavelet packets. This is, by the way, a consequence of the fact that bushes are not cycle free. To perform a full decomposition, i.e. calculating all sub-bands with maximum decomposition level in each direction, is just as much work as for

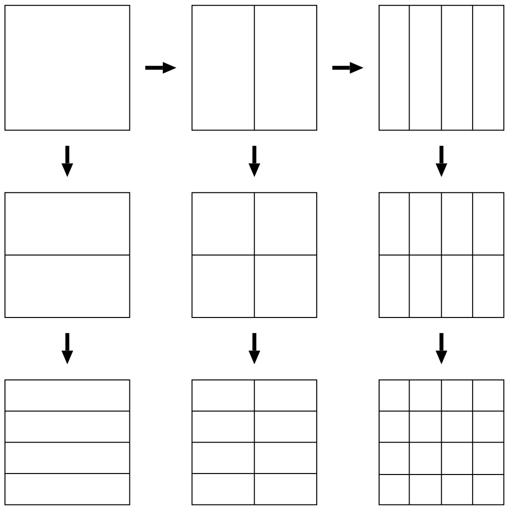


Figure 3.11: All sub-bands of a full anisotropic wavelet packet decomposition

isotropic wavelet packets. This is because the resulting sub-bands actually contain isotropic wavelet packets.

To see how much work it is to calculate all possible sub-bands, look at Figure 3.11. Again, each coefficient is calculated by the inner product of a filter and another sub-band. So we have to count the number of coefficients. Following the construction of Figure 3.11 we get:

$$C_{AP} = n((l+1)^d - 1) = O(n(\log n)^d)$$

To calculate all possible sub-bands with minimal memory demand and without redundant calculations, we apply the algorithm in Pseudo code 2. This

```
Decomp (S) :=                 Decomp (S, m) :=
    Decomp (S, n)                 if m > 1 then Backup (S)
                                  for k = 1 ... m
                                      if k > 1 then Restore (S)
                                      Filter (S, k)
                                      Decomp (child(S, 1), k)
                                      Decomp (child(S, 2), k)
```

Pseudo code 2: How to generate all possible sub-bands of anisotropic wavelet packets

algorithm follows the decomposition order shown in Figure 3.11 while concentrating on a depth first order (as far as possible for anisotropic wavelet packets).

Comparing the work for anisotropic and isotropic decompositions, one can easily see that performing a decomposition after a known decomposition structure is of equal complexity. Only, the best basis decomposition is more complex

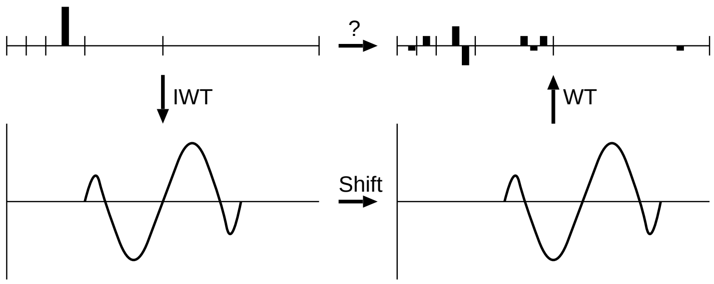


Figure 3.12: Shifting 1-D wavelets

for anisotropic wavelet packets. In order to investigate how much the complexity differs, we build the quotient of W_{AP} and W_{WP}.

$$\frac{W_{AP}}{W_{WP}} = \frac{fn((l+1)^d - 1)}{2fnl} = \frac{(l+1)^d - 1}{2l}$$

For $d = 2$ we get $\frac{l}{2} + 1$ and for $d = 3$ we get $\frac{l^2}{2} + \frac{3}{2}(l+1)$. This shows that the work for anisotropic wavelet packet best basis decompositions compared to the isotropic case increases with $O(\log^{d-1} n)$.

3.5 Shift Invariance

Shift operations are very common in video coding applications as to represent moving objects. To integrate these methods into wavelet compression, it is often necessary to perform shift operations in the wavelet domain.

As stated in [51], a transform is shiftable if there are interpolation functions that calculate coefficients representing a shifted version of an original signal from neighbouring coefficients representing the non-shifted signal. Given a wavelet transform, neighbourhood is considered intra-sub-band only. This means that the coefficients of a sub-band of a shifted signal can be calculated from the same sub-band of the non-shifted signal.

In [51] it is shown that this condition is equivalent to the condition that the energy (sum of squared coefficients) of a sub-band is invariant to shifts of the signal. Hence the frequently used term “shift invariance”.

Unfortunately, the wavelet transform (including wavelet packet transforms) is not shift invariant (see Figure 3.12). However, there are (approximately) shift invariant versions of the wavelet transform, such as the undecimated wavelet transform [50] and complex wavelet transforms [2, 19]. All these transforms either imply significant redundancy (over-completeness) or lack of perfect reconstruction (especially for higher dimensions), which both runs contrary to compression purposes.

Chapter 4

Wavelet Transform Based Coding

Wavelet transform based methods get more and more widely used in image and video compression. They are used in numerous scientific approaches (as e.g. EZW [49], SPIHT [48, 18], SFQ [68, 69] and others [4, 59]) and compression standards (JPEG2000 [16, 7] and MPEG-4 [21]).

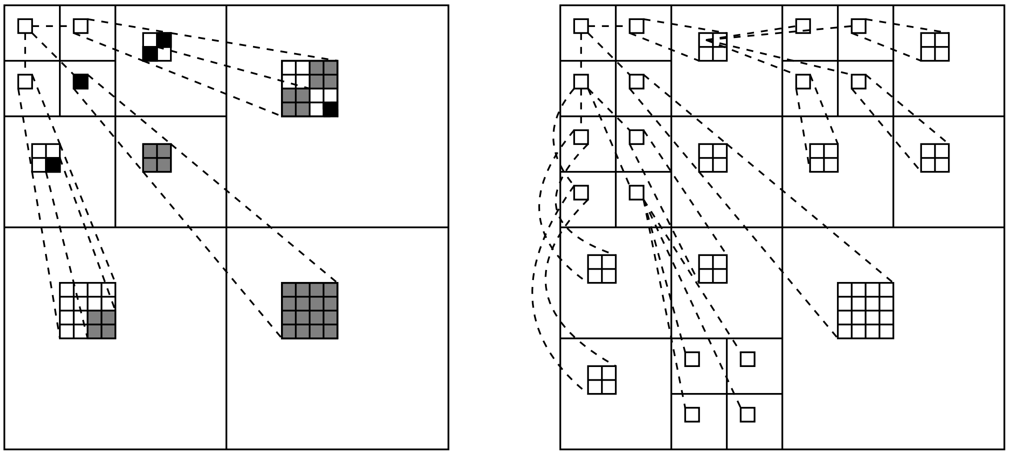
The superiority of wavelet methods over discrete cosine transform (DCT) based methods like JPEG [62] lies in the fact that DCT methods segment images into small blocks due to complexity problems (see Section 3.2.4). This produces significant blocking artifacts especially for high compression ratios. However, this is not a big problem in common video compression schemes like MPEG-2 [5] or MPEG-4 [21] because block matching methods are used for motion compensation. Since the borders of motion blocks and DCT blocks match, block motion errors do not imply serious ringing in the DCT coefficients. This is one reason that wavelet methods have not been as successful in video coding as in image coding.

In this chapter, we will concentrate on zero-tree coding algorithms. These algorithms do not only exploit the statistical distribution of energy among sub-bands and within sub-bands. They also consider dependencies between spatially neighbouring coefficients of distinct sub-bands. In this way, groups of neglectable coefficients can be encoded efficiently as so-called zero-trees.

4.1 Zero-Trees

Zero-tree based algorithms arrange the coefficients of a wavelet transform in a tree-like manner (see Figure 4.1 (a)). All coefficients are connected by trees that are rooted in the approximation sub-band's coefficients (the sub-band that is low-pass filtered only). The following notations are used:

off(p) The direct offspring of a coefficient p, i.e. all coefficients whose parent coefficient is p.



(a) Pyramidal decomposition (b) Wavelet packet decomposition

Figure 4.1: Coefficient trees in zero-tree coding. The dark areas are insignificant coefficients. Black coefficients are zero-tree roots

desc(p) All descendants of a coefficient p. This includes $\text{off}(p)$, $\text{off}(\text{off}(p))$ and so on.

parent(p) The parent coefficient of p. $p \in \text{off}(\text{parent}(p))$.

Furthermore, a zero-tree is a sub-tree which entirely consists of insignificant coefficients. The significance of a coefficient is relative to a threshold which plays an imported role in zero-tree coding algorithms:

$$\text{sig}(p) \Leftrightarrow |p| \geq \text{threshold}$$

We will see that sometimes the root coefficient of the subtree (or even its direct offspring) does not have to be insignificant. Zero-trees can be viewed as a collection of coefficients with approximately equal spatial position.

4.2 Similarity Trees

To build trees of coefficients is more complicated when we face arbitrary wavelet packet decompositions (see Figure 4.1 (b)). So let us start at the beginning: Coefficient trees should connect coefficients of approximately equal spatial positions in a way so that bigger coefficients (usually the low-pass filtered ones) are next to the root of the trees and the significance (absolute value) of parent and child coefficients is statistically related (similarity condition).

To achieve this, we have to find similar sub-bands to define parent-child relationships between their coefficients. We connect similar sub-bands by trees which we will call similarity trees.

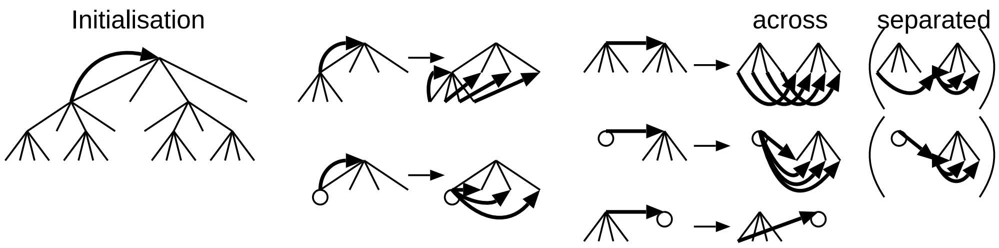


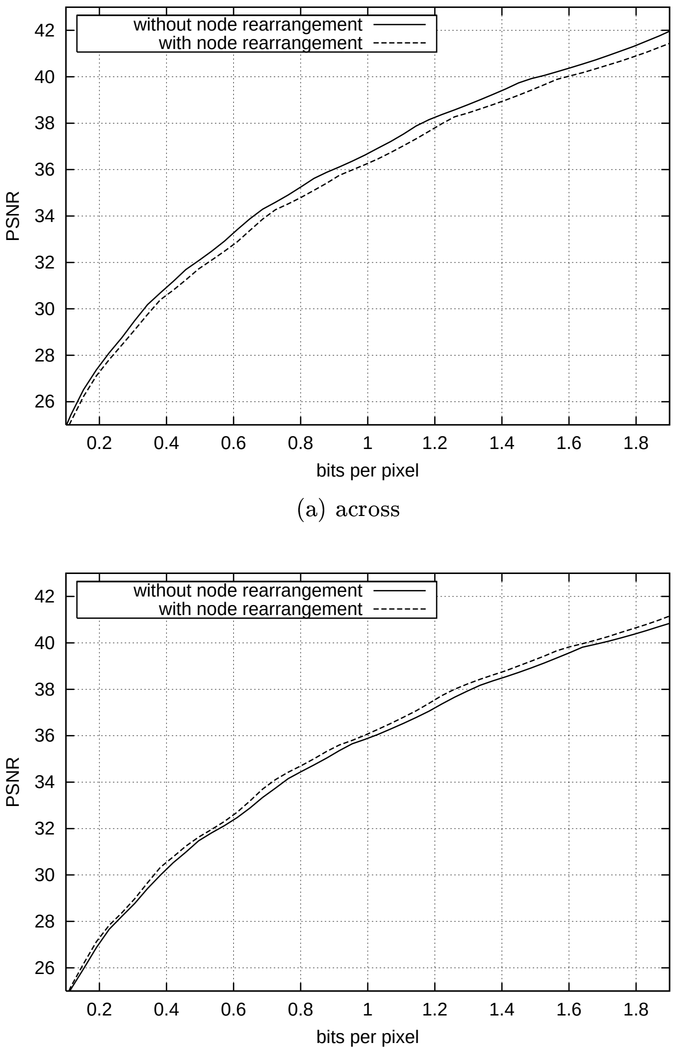
Figure 4.2: Construction of similarity trees. Thin lines indicate the wavelet packet decomposition tree; thick lines indicate the similarity tree. Starting with an arbitrary packet decomposition tree (left), the shown tree transform operations are applied until only leaf sub-bands are connected. Small circles indicate leaf sub-bands

Note that there are three kinds of trees now: (a) the wavelet packet decomposition tree, (b) the similarity tree that connects the leaves of the packet tree and (c) the coefficient tree ("carrying" the zero-trees) that connects coefficients as forced by the similarity tree. The last one is actually a *wood* because it has roots in every coefficient of the approximation sub-band.

See Figure 4.2 for a set of construction operations for similarity trees. The leftmost child-node always represents the double-low-pass-filtered sub-band (LL). The construction starts with the basic similarity between the first approximation sub-band and the original image. The operations transform coarse similarity relations into detail relations until only leaf-sub-bands (non-decomposed) are part of the similarity tree.

There are two options: *across* and *separated.* While similarity trees of type separated try to keep similarity sub-trees entirely within a wavelet packet subtree, similarity trees of type across span across a number of sub-bands with significantly differing frequency content. Type separated favours deep similarity trees. Type across produces sub-trees of equal depth as far as possible. Figure 4.1 (b) is, for instance, of type across. Figure 4.3 shows a comparison of SMAWZ (see Section 4.5) coding performance in relation to the two types of similarity trees. It shows that across is superior. The results are similar for all images tested by the author.

A method was proposed in [46] to rearrange the nodes (sub-bands) within the similarity tree to avoid the case of a parent sub-band being larger than its child sub-band. The problem with this is that in the corresponding coefficient trees a coefficient could have more than one parent coefficient. We choose to resolve this by terminating the coefficient trees at all but one of the multiple parent coefficients. I.e. only one parent coefficient is connected to its child coefficients. Nevertheless, the rearrangement method was implemented additionally for comparison purposes. See Figure 4.3 for PSNR results. Unfortunately, the rearrangement does not increase the rate-distortion performance significantly.



(a) across

(b) separated

Figure 4.3: Comparison of codec performance for distinct types of similarity trees. The results relate to the "Barbara" image at a maximum decomposition depth of 6 using the biorthogonal 7/9-tap filter, best-basis algorithm and the L1 norm (sum of absolute values) as cost function

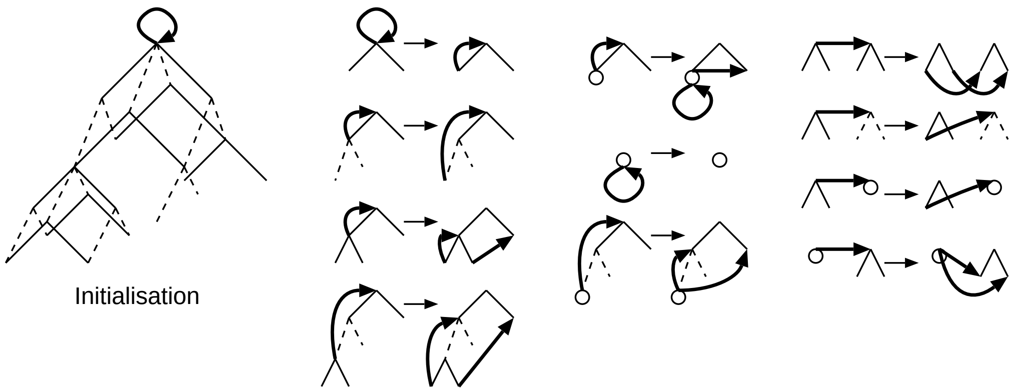


Figure 4.4: Construction of similarity trees for bushes of anisotropic wavelet packets

The difference is neglectable for most images. If not, this method usually increases performance slightly for similarity trees of separated type and decreases performance for the across type.

4.3 Similarity Trees for Anisotropic Wavelet Packets

To build similarity trees for anisotropic wavelet packets is even more complicated. First of all, we have to make clear which result we expect. Since isotropic decompositions are a subset of anisotropic decompositions, the rules for anisotropic decompositions should produce the same similarity trees as the isotropic for isotropic decompositions. For real anisotropic decompositions, the result should closely resemble what one would expect for a similar isotropic decomposition.

This is done by dividing the rules for isotropic decompositions into several sub-steps which follow branches in the decomposition bush only in a single dimension. Figure 4.4 shows these rules. First, the initialisation is simply a connection of the root node to itself. The rules are supposed to substitute the rules in Figure 4.2 when applied in groups of appropriate order. For instance, the first rule followed by the second (in the leftmost column) applied to the initial state produces the initial state of the isotropic case. Continuing with the fourth rule twice and the first rule of the rightmost column also twice accomplishes what the first rule of Figure 4.2 does. However, if the application of one of these rules is not possible because the anisotropic decomposition bush is not complete (not isotropic), only an approximation of the isotropic case can be achieved. For instance, the second rule of the rightmost column might be chosen instead of the first. Note that often more than one rule could be applied. In these cases, the first available rule should be chosen.

The author cannot show any 2-D coding results here because anisotropic wavelet packets are only used for 3-D video coding in this work (see Chapter 5).

4.4 SPIHT

As the actual coding algorithm we will use an algorithm that is derived from SPIHT [48]. This algorithm is both simple and efficient in terms of execution and rate-distortion performance. However, to understand the derived algorithm, it is advantageous to know how SPIHT works.

Although the SPIHT algorithm is sufficiently explained in the original paper [48], it is helpful in this context to reformulate the algorithm.

Significance information is represented by three lists:

LIS List of insignificant set of pixels
An entry in this list can be of two types:
Type A: $p \in \mathrm{LIS}_A \Leftrightarrow \neg\,\mathrm{sig}(\mathrm{desc}(p)) \wedge p \notin \mathrm{desc}(\mathrm{LIS})$
Type B: $p \in \mathrm{LIS}_B \Leftrightarrow \neg\,\mathrm{sig}(\mathrm{desc}(\mathrm{off}(p))) \wedge p \notin \mathrm{desc}(\mathrm{LIS})$

LIP List of insignificant pixels
$p \in \mathrm{LIP} \Leftrightarrow \neg\,\mathrm{sig}(p) \wedge p \notin \mathrm{desc}(\mathrm{LIS}_A) \wedge p \notin \mathrm{desc}(\mathrm{off}(\mathrm{LIS}_B))$

LSP List of significant pixels
$p \in \mathrm{LSP} \Leftrightarrow \mathrm{sig}(p)$

The LIS basically contains all zero-tree roots. An entry of type A corresponds to an insignificant sub-tree without its root. An entry of type B corresponds to an insignificant sub-tree without its root and the root's direct offspring. The LIP contains all insignificant coefficients that are not part of any zero-tree in the LIS. The LSP contains all significant coefficients.

```
threshold ← max_{p∈allcoefficients} |p|
fill LIS, LIP with approximation sub-band
set LSP empty
for each refinement step
   threshold ← threshold /2
   process LIS
   process LIP
   process LSP
```

Pseudo code 3: The SPIHT algorithm

The algorithm is shown at a coarse level in Pseudo code 3. Initially, the threshold is greater than all coefficients. Thus, the LIS and the LIP are filled with the approximation sub-band's coefficients, and the LSP is empty. After that, each entry of each list has to be tested for a change of significance, and the result of the test has to be encoded as a bit in the bit-stream (BS). If, for instance, a type A entry of the LIS turns out not to to be insignificant any more (to be precise: its descendants), a zero bit has to be written into the bit-stream, and the entry has to be deleted from the LIS, inserted as a type B entry at the

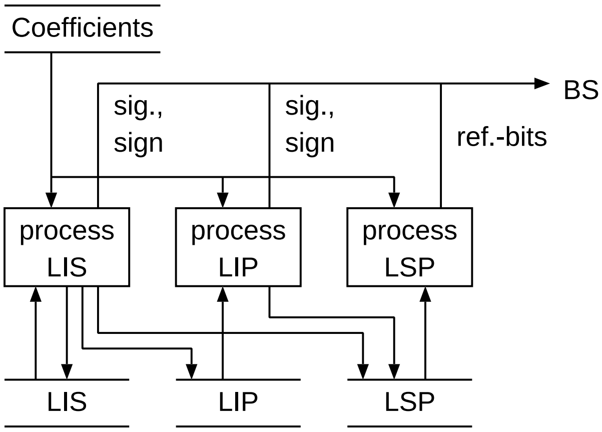


Figure 4.5: Data flow of the SPIHT algorithm

end of the LIS and its direct offspring has to be inserted at the end of the LIP. All entries inserted at the end of a list are also processed in the same refinement step until no more entries are left. Figure 4.5 shows a data flow graph for a refinement step.

The decoding process performs the same algorithm. It does not, however, evaluate the significance of the list entries, but simply reads this information from the bit-stream and approximates the value of the corresponding coefficient as good as it can.

4.5 Significance Map Based Adaptive Wavelet Zerotree Coding (SMAWZ)

The use of lists of coefficients in SPIHT is problematic for several reasons. The maximum size of SPIHT's lists is hard to predict. They can grow very large. This is a problem especially for hardware implementations where the memory size has to be chosen according to maximum size of the algorithm's data structures. Another problem is that of easy parallelisation (see Section 7.2). Data driven parallelity favours spatially oriented coefficient scan orders. Frequent inserts into and deletes from coefficient lists cause highly jumbled coefficient scan orders. Therefore, we will develop an algorithm in this section that substitutes significance maps for the lists.

Algorithms similar to SPIHT that do not use lists are proposed in [64, 36, 37]. Although a breadth-first scan order through the trees of coefficients [64] is PSNR-optimal, a depth-first scan order [36, 37] is preferred here because of better spatial separability in parallelisation.

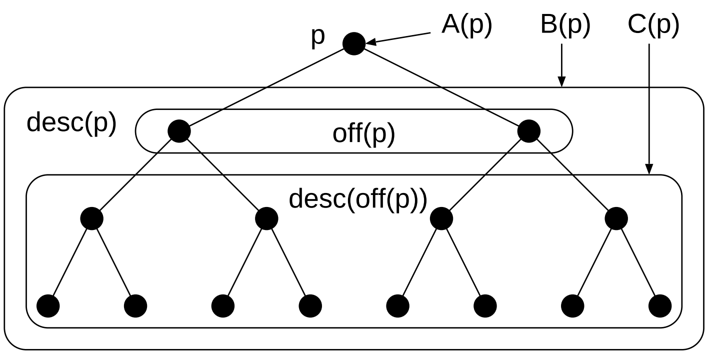


Figure 4.6: Predicates used in the SMAWZ algorithm

4.5.1 Simple Approach

As mentioned above, we want to take the SPIHT algorithm and substitute the lists of coefficients by bitmaps. Nevertheless, this is not as easy as it sounds. Thus, we will rewrite the whole algorithm.

In the following we will use three logical predicates $A(p)$, $B(p)$ and $C(p)$ which are defined as follows:

$$\begin{aligned} A(p) &\Leftrightarrow |p| \geq \text{threshold} \\ B(p) &\Leftrightarrow \bigvee_{q \in \text{desc}(p)} A(q) \\ C(p) &\Leftrightarrow \bigvee_{q \in \text{off}(p)} B(q) \end{aligned}$$

$A(p)$ simply denotes the significance of the coefficient p. $B(p)$ is true if and only if at least one of p's descendants is significant, while $C(p)$ denotes the same but does not include the direct offspring of p. This is visualised in Figure 4.6. By these predicates the state of significance of a given set of coefficients can be described in terms of zero-trees. Corresponding to these predicates we will use the mappings a, b and c which essentially represent the same as A, B and C. The difference is that A, B and C immediately change their values if the threshold is changed and are therefore implemented as a function in the used programming language. a, b and c have to be updated explicitly and are therefore implemented as array of Boolean values. a, b and c are called "significance maps" that appear in the title and substitute the lists of coefficients used in SPIHT.

The algorithm is responsible for the equality of a, b, c and A, B, C respectively while the threshold is successively decreased by a factor of $\frac{1}{2}$. This should be done by avoiding the evaluation of A, B and C as far as possible because – following the idea of the SPIHT algorithm – the result of each evaluation will be encoded into the bit-stream as one bit to allow the decoder to reproduce the decisions the encoder has made.

```
ProcessAll :=
    threshold ← max_{p∈allcoefficients} |p|
    set a, b and c to all false
    for each refinement step
        threshold ← threshold /2
        for p in approximation-subband
            ProcessCoeff(p, true)

ProcessCoeff(p, ĉ) :=
    if a_p then Refine(p) else a_p ← A(p)
    if ¬b_p ∧ ĉ then b_p ← B(p)
    if b_p ∧ ¬c_p then c_p ← C(p)
    if b_p then
        for q in off(p)
            ProcessCoeff(q, c_p)
```

Pseudo code 4: Zero-tree coding algorithm

The algorithm that obeys these rules is shown in Pseudo code 4. The outer loop is the refinement loop which divides the threshold by 2 in each iteration. This is exactly the same as in the original SPIHT algorithm. Within this loop the algorithm navigates through the set of coefficients along trees of coefficients in a depth-first manner starting at the set of coefficients contained in the approximation sub-band. This is accomplished by the recursive procedure "ProcessCoeff".

For each coefficient, the state of a, b and c is checked one after another. $a_p \leftarrow A(p)$ has to be evaluated only if a_p is false because the transition true $\mapsto$ false is not possible for a_p. If $A(p)$ is true then the sign of p has to be encoded as well (not shown). If a_p is already true the procedure "Refine" is called which encodes another bit of the coefficients value to refine the decoded approximation. $b_p \leftarrow B(p)$ has to be evaluated only if (again) b_p is false *and* $\hat{c} = C(\text{parent}(p))$ is true because $B(p) \Rightarrow C(\text{parent}(p))$. $c_p \leftarrow C(p)$ has to be evaluated only if c_p is false *and* b_p is true because $C(p) \Rightarrow B(p)$. At last the recursion to the child coefficients only has to be performed if b_p is true (for an obvious reason).

The decoding algorithm looks exactly the same. The only difference is that instead of encoding the results of the evaluation of A, B and C, this information is read from the bit-stream. Together with the sign- and refinement-bits this is enough information to enable the decoder to perform the same steps as the encoder and to approximate the coefficients with an error below the threshold.

Note that this algorithm encodes the same information (in fact the same bits) as the SPIHT algorithm. Only the order of the bits is different. This means that at the end of each refinement step the compression performance is equal to that of SPIHT. In between, the order of the bits that are written to the bit-stream is crucial because the bits can have different effect on the decoded image. Thus, it is important to encode bits with bigger effect first. Figure 4.7 shows the comparison of the PSNR-performance for the "houses"-image and pyramidal wavelet decomposition. The simple algorithm shows major drawbacks with respect to SPIHT.

4.5.2 Improved Approach

The fact that the simple approach is significantly inferior to SPIHT forces us to look for a possibility to improve this approach. The reason for the loss of

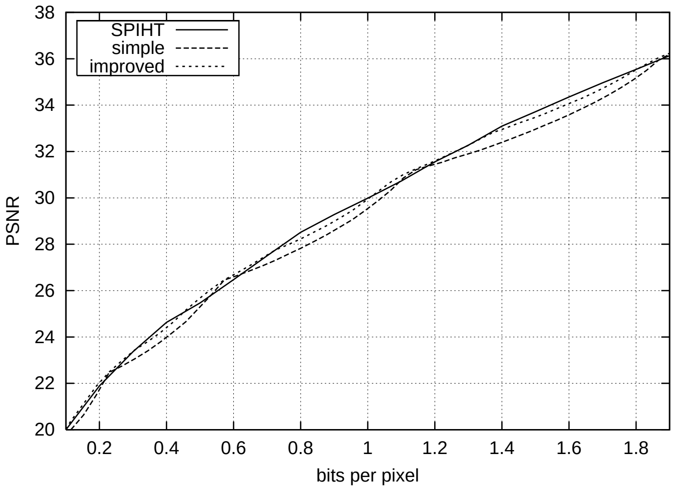


Figure 4.7: PSNR of SMAWZ compared to SPIHT. The results relate to the "houses" image at a decomposition depth of 6 using the biorthogonal 7/9-tap filter and pyramidal wavelet decomposition

rate-distortion performance must have something to do with the order in which bits are written to the bit-stream during a refinement step – since the bits are essentially the same as in SPIHT. The order of bits is determined by the order in which the coefficients are processed. So this is what has to be changed – without losing the advantage of significance maps and the depth-first scan order.

First of all, we have to distinguish between important and unimportant bits: Important bits are those that have big impact on the reconstructed image, i.e. A-bits plus sign-bits and to some extent refinement bits. Most important are coefficients that are very likely to turn significant ($A(p) = \text{true} \wedge a_p = \text{false}$). Unimportant bits are those that have no direct influence on image content, i.e. B- and C-bits. Thus, important bits have to be encoded as early as possible – without loss of correctness.

The algorithm in Pseudo code 5 takes these requirements into account. Each refinement step is split into at least three passes. The first pass only processes A-bits of insignificant coefficients that are not covered by zero-trees. No evaluations of B and C are performed. Pass 2 and 3 evaluate B- and C-bits as well as A-bits of newly discovered potentially significant coefficients (immediately after they are discovered). B- and C-bits of these newly discovered coefficients are evaluated in subsequent passes. This is to evaluate unimportant bits as late as possible. Pass 3 has to be repeated until no new coefficients are found.

In this way, the rate-distortion performance of our improved algorithm closely matches the performance of SPIHT, as is shown in Figure 4.7.

```
ProcessAll :=
    threshold ← max_{p∈allcoefficients} |p|
    for each refinement step
        threshold ← threshold /2
        for p in approximation-subband
            ProcessCoeffPass1(p)
        for p in approximation-subband
            ProcessCoeffPass2(p, true)
        do
            foundnewcoeffs ← false
            for p in approximation-subband
                ProcessCoeffPass3(p, true)
        while foundnewcoeffs

CheckDescSig(p, ĉ) :=
    if ĉ then b̄_p ← B(p)
    if b̄_p
        for q in off(p) do a_q ← A(q)
        foundnewcoeffs ← true

ProcessCoeffPass1(p) :=
    if ¬a_p then ā_p ← A(p)
    if b_p then
        for q in off(p) do ProcessCoeffPass1(q)

ProcessCoeffPass2(p, ĉ) :=
    if ā_p then ā_p ← false, a_p ← true else Refine(p)
    if b_p then
        if ¬c_p then c_p ← C(p)
        for q in off(p) do ProcessCoeffPass2(q, c_p)
    else CheckDescSig(p, ĉ)

ProcessCoeffPass3(p, ĉ) :=
    if b_p then
        for q in off(p) do ProcessCoeffPass3(q, c_p)
    elsif b̄_p then
        b̄_p ← false, b_p ← true
        if ¬c_p then c_p ← C(p)
        for q in off(p) do CheckDescSig(q, c_p)
```

Pseudo code 5: Improved zero-tree coding algorithm

4.5.3 3-D Coding

To encode three-dimensional data sets is basically a straight-forward extension of the two-dimensional case. As a first step, some kind of 3-D wavelet transform has to take place. Similarity trees have to be constructed as shown above. The 3-D extension of the rules for the construction of similarity trees is obvious.

Note that the average number of child nodes of non-terminal nodes increases in the similarity trees (and in the coefficient trees as well). Therefore, the zerotree structure is more flat in three dimensions. This effect is further intensified by the fact that the decomposition depth is usually smaller in the 3-D case. The reason for this is that the decomposition depth is assumed to be proportional to the data size in one dimension ($\approx \log_2 \sqrt[d]{n}$).

Once the expressions *off*, *desc* and *parent* are defined, the algorithms formulated above can be used without modification.

4.6 Arithmetic Coding

SPIHT uses arithmetic coding to improve rate-distortion performance by about 0.5 dB. Four bits of four neighbouring coefficients (the direct offspring of a single parent coefficient) are grouped together and fed into the arithmetic coder as a single symbol. This section deals with the integration of arithmetic coding into the SMAWZ codec.

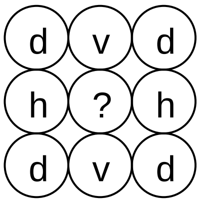
4.6.1 2-D Contexts

In the case of wavelet packet decompositions (and similarity trees), the direct offspring of a coefficient does not necessarily consist of exactly four coefficients. This makes the above approach not applicable. Looking for another context-based arithmetic coding approach for coefficient significance, we find EBCOT [56], which JPEG2000 is based upon.

It basically sums the significance bits of neighbouring coefficients (horizontally, vertically and diagonally) and builds several contexts out of this information (see Figure 4.8). Each context yields a different probability for 1- and 0-bits. The contexts also depend on the sub-band type (LL, HL, LH and HH).

A similar approach is used for the sign bits. 1 and -1 denote positive and negative signs respectively and 0 denotes a so far insignificant coefficient. If both neighbouring coefficients are significant and their signs are not equal, 0 is used as well.

Using this approach, the rate-distortion performance is improved by about 0.5 dB (as in SPIHT). See Figure 4.9 for a comparison of SMAWZ including arithmetic coding and SPIHT.



Significance coding:

LL,LH,HL	h	0	0	0	0	0	1	1	1	2
	v	0	0	0	1	2	0	0	1+	x
	d	0	1	2+	x	x	0	1+	x	x
	context	0	1	2	3	4	5	6	7	8
HH	d	0	0	0	1	1	1	2	2	3+
	h + v	0	1	2+	0	1	2+	0	1+	x
	context	0	1	2	3	4	5	6	7	8

Sign coding:

h	1	1	1	0	0	0	-1	-1	-1
v	1	0	-1	1	0	-1	1	0	-1
predicted	1	1	1	-1	1	1	-1	-1	-1
context	4	3	2	1	0	1	2	3	4

Figure 4.8: Contexts for arithmetic coding taken from EBCOT. Neighbouring coefficients are used to predict the significance of a coefficient

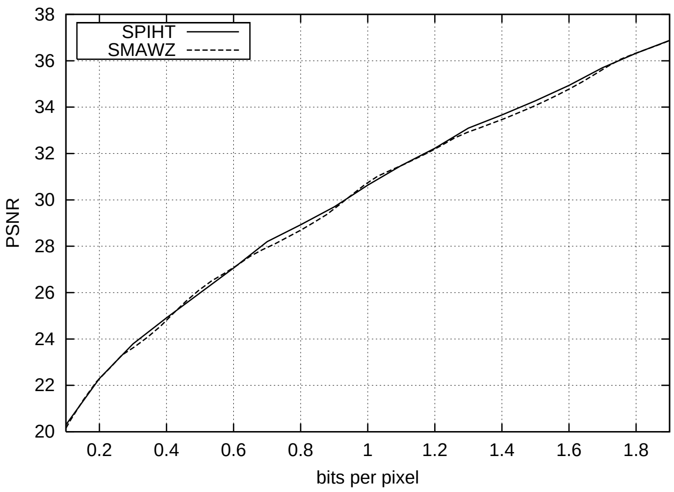
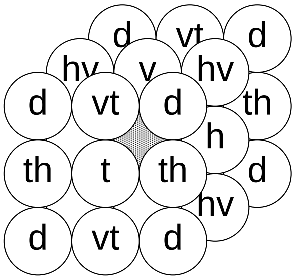


Figure 4.9: PSNR of SMAWZ compared to SPIHT, both using arithmetic coding



h	0	>0	0	0	0	>0	>0	>0
v	0	0	>0	0	>0	0	>0	>0
t	0	0	0	>0	>0	>0	0	>0
a	0	1	2	3	4	5	6	7

vt	0	>0	0	0	0	>0	>0	>0
th	0	0	>0	0	>0	0	>0	>0
hv	0	0	0	>0	>0	>0	0	>0
b	0	4	5	6	1	2	3	7

a	0	0	0	0	0	>0	>0
b	0	0	0	0	>0	$=$ a	$\neq$ a
d	0	0	>0	>0			
sub-band	xxL	xxH	xxL	xxH			
context	0	1	2	3	b+3	a+10	a+18

Figure 4.10: Contexts for arithmetic coding of 3-D significance information

4.6.2 3-D Contexts

In the 3-D case, arithmetic coding is slightly more complicated since a straightforward extension of the contexts in Figure 4.8 would produce too many contexts. This would undermine the adaptiveness of the arithmetic coding algorithm. Note that 26 neighbours together with 8 sub-band types would produce about $5 \cdot 10^8$ contexts.

Again, we will sum the significance bits of neighbouring coefficients. There are two first order neighbours in each of the three dimensions. The corresponding significance sums h, v and t can take values from 0 to 2. The second order neighbours are summed in hv, vt and th which take values from 0 to 4. The third order neighbours are summed in d which takes values from 0 to 8. See Figure 4.10.

To reduce the number of contexts, only a few cases for first and second order neighbours are considered. There are eight cases for each type which are represented by a and b (see Figure 4.10). a and b, together with d and the sub-band type are reduced further to 26 different contexts. This is the result of lengthy experiments with significance statistics.

Adopting the context model for signs to the 3-D case leads to the same problem of too many contexts. Again, we set h, v and t to the sign of neighbouring coefficients in horizontal, vertical and temporal directions. h, v and t are set to $+1$ (-1) if at least one neighbour has positive (negative) sign. They are set to zero if either both neighbours are insignificant (not encoded yet) or the neighbours have differing signs.

So the number of contexts is $3^3 \cdot 8 = 216$. As in the 2-D case, the contexts (h, v, t) and $(-h, -v, -t)$ have the same probabilities for $+1$ and -1 respectively, so the number of contexts can be halved. Nevertheless, the result is still too

hvt	000	00+ 00-	0+- 0-+	0+0 0-0	0++ 0--	+-- -++	+-0 -+0	+-+ -+-	+0- -0+	+00 -00	+0+ -0-	++- --+	++0 --0	+++ ---
LLL	48	48	48	48	48	48	48	48	48	48	48	48	48	48
HHH	0	1	2	1	3	4	2	-4	2	1	3	4	3	5
HLL	0	6	7	9	11	13	15	-14	8	10	12	16	17	18
LHL	0	6	8	10	12	14	-15	13	7	9	11	16	17	18
HLH	0	19	20	22	24	26	28	-27	21	23	25	29	30	31
LHH	0	19	21	23	25	27	-28	-26	20	22	24	29	30	31
LLH	0	32	33	34	35	36	0	-36	33	34	35	37	38	39
HHL	0	40	41	42	43	44	0	-44	41	42	43	45	46	47

Figure 4.11: Contexts for arithmetic coding of 3-D sign information

much.

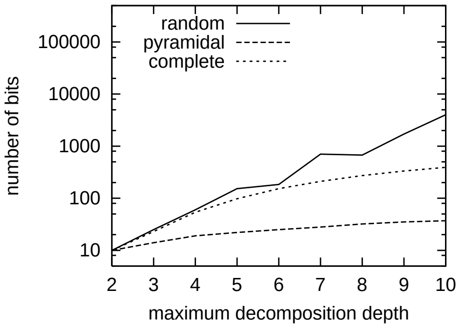
Experiments have shown that the context (h, v, t) for a HLx sub-band can be used as (v, h, t) context for a LHx sub-band as well because of approximately equal probabilities. Additionally, (h, v, t) and (v, h, t) can be identified for a LLx or a HHx sub-band. Unfortunately, the t-dimension behaves differently in this respect, so that e.g. (h, v, t) and (h, t, v) cannot be identified. Figure 4.11 shows the resulting context assignment. Context types are listed in the header line. Two context types in the same column (e.g. 0+- and 0-+) indicate that the same context number is used but 0 is predicted instead of 1 for the lower context type. Likewise, a negative context number in the table indicates the same. The author admits that 49 contexts is still a high number but is convinced that this is more or less unavoidable.

Regarding anisotropic wavelet packet decompositions, another problem arises. These sub-bands are not inherently of a LLL, ..., HHH type. Since the type is used to determine the contexts for arithmetic coding, an equivalent has to be found. This can be done by finding the type of the last filter (high- or low-pass) for each dimension that was used to produce the sub-band. Together, these types form the desired sub-band type.

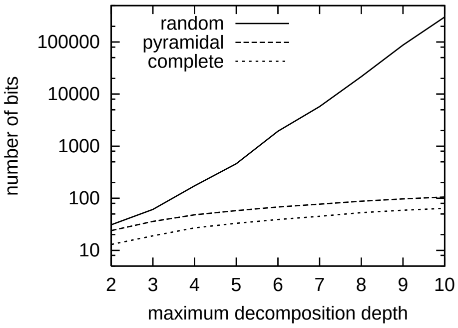
4.7 Encoding the Decomposition Structure

The decoder can decode bit-streams created by encoders described above only if it is capable of constructing decomposition trees and similarity trees exactly as used by the encoder. The similarity trees are constructed from the decomposition structure which should be known by the decoder. However, the decomposition structure has to be encoded to be available to the decoder.

For pyramidal wavelet decompositions, it is sufficient for the decoder to know the decomposition depth. This value can either be set to a fixed number or encoded into the bit-stream. However, the decomposition structure of wavelet packets is more complicated and can consume a serious amount of bit-rate when



(a) isotropic WP



(b) anisotropic WP

Figure 4.12: Number of bits necessary to encode a 2-D isotropic wavelet packet tree structure (a) and a 2-D anisotropic wavelet packet bush structure (b). The numbers for the random decomposition represent the maximum of ten randomly generated decompositions.

it is encoded. Therefore, the encoding has to be efficient. Moreover, we choose to search for an encoding scheme that is especially efficient on more regular decomposition structures. This section presents methods to use arithmetic coding in order to encode repeating decomposition patterns with a minimum amount of bits.

4.7.1 Encoding an Isotropic Decomposition Structure

The decomposition structure of conventional isotropic wavelet packets is a full n-ary tree. To encode such a tree, basically one bit has to be encoded for each node (vertex), indicating if the tree branches at this node. In most cases, it is better to encode the maximum tree depth before these bits. In this way, the decoder knows that nodes at the maximum level do not branch. Therefore, these bits may be omitted from the bit-stream. As many (or even most) nodes usually are at maximum level, this reduces the bit-rate significantly.

To use arithmetic coding, four neighbouring nodes are grouped together. The combination of the corresponding four bits gives us 16 possible symbols to be encoded by the arithmetic coder. Experience shows that certain symbols have higher probability than others, which can be exploited by the arithmetic coder. See Figure 4.12 (a) for an evaluation of coding performance of this scheme.

4.7.2 Encoding an Anisotropic Decomposition Structure

To encode a decomposition structure optimally (with a minimum number of bits) it is essential to start with a unique representation. Otherwise, the encoded bits necessarily contain redundancy. Fortunately, a bush *is* a unique representation of the decomposition structure (see Section 3.4.1). However, this does not mean that it is trivial to encode this structure without redundancy. The algorithm in Pseudo code 6 does this. It takes into account that branches of a node imply

```
Encode (B) :=
   Encode (root (B), ∅, ∅)

Encode (S, dobranch, nobranch) :=
   for all dimensions d
      if not nobranch[d] and not dobranch[d]
         Encode (haschildren(d))
   if a dimension d exists with haschildren(S, d)
      for all dimensions e
         D_{1,2}[e] ← haschildren(S, e) ∧ d ≠ e
      for all dimensions e with ¬ haschildren(S, e)
         D_1[e] ← haschildren(child(S, d, 1), e)
         D_2[e] ← haschildren(child(S, d, 2), e)
         Encode D_1[e] and D_2[e] together (note: at least one is false)
         N_1[e] ← ¬D_1[e]; N_2[e] ← ¬D_2[e]
      Encode (child(S, d, 1), D_1, N_1)
      Encode (child(S, d, 2), D_2, N_2)
```

Pseudo code 6: Algorithm to encode an anisotropic wavelet packet decomposition structure without redundancy

certain branches of descendant nodes: Firstly, if the bush branches at a node in dimension d, then all child nodes in any dimension different from d have to branch in dimension d. Secondly, if the bush does not branch at a node in dimension d, then at least one child node in a dimension different from d does not branch in dimension d. The algorithm, therefore, keeps track of dimensions in which nodes must or must not branch ("dobranch" and "nobranch") and does not encode this information. This removes all redundancy from the bush's structure.

Additionally, nothing has to be encoded if the maximum decomposition depth is reached (if it is known). At one point, two Boolean values are encoded together where only three states are possible. To encode this without redundancy, an arithmetic coder is necessary. This coder also exploits probabilistic redundancies in the structure.

Figure 4.12 shows the encoding results compared to the isotropic case. Note that in both cases regular structures such as the pyramidal and the complete decomposition can be encoded very efficiently. Higher bit-rates for the anisotropic case are a consequence of the fact that the number of possible bushes is much higher than the number of trees with the same maximum depth.

Chapter 5

Video Coding by 3-D Wavelet Transform

Most video compression algorithms rely on 2-D based schemes employing motion compensation techniques. On the other hand, rate-distortion efficient 3-D algorithms exist which are able to capture temporal redundancies in a more natural way (see e.g. [6, 35, 17, 14, 57] for 3-D wavelet/sub-band coding and [1] for hybrid 3-D fractal/sub-band coding).

Wavelet packet based compression methods [69, 41] have been developed which outperform the most advanced wavelet coders (e.g. SPIHT[48]) significantly for textured images in terms of rate-distortion performance. A similar development in the area of video compression may be observed for 3-D sub-band [34, 60] or 3-D wavelet packet [15] video coding.

The first approach for 3-D video coding is to apply the traditional pyramidal wavelet transform on the 3-D video data. However, there is evidence that other decompositions can do a better job. Firstly, wavelet packet decompositions have proved their efficiency for certain types of images (especially textures, see [21]). So, this *can* also be the case for 3-D video data. Secondly, cyclic events such as rotating wheels produce regular patterns in the time domain which can efficiently be represented by wavelet packets.

Finally, the fact that the behaviour of video data in the spatial domain differs from that in the time domain. This brings up anisotropic wavelet packets, which are able to represent shapes and features with different properties in different directions (in terms of frequency components).

This chapter deals with the application of the SMAWZ codec developed in Chapter 4 on the 3-D wavelet transforms of a video sequence. We will use the pyramidal wavelet transform as well as wavelet packets and anisotropic wavelet packets. The results are compared against MPEG-4.

Experiments are conducted on the following well-known video sequences: *carphone*, *coastguard*, *claire*, *container* and *foreman*. All sequences are in the QCIF format (176×144 pixel). We select the first 64 frames (Y-component only) of each sequence and perform the wavelet transform on the resulting $176 \times 144 \times 64$ pixel array.

As MPEG-4 codec, the implementation of the verification model from December 1999 is used. We choose a frame group with three P-frames and 2 B-frames per P-frame, which results in IBBPBBPBBP as encoding scheme. Because we only consider monochrome (grey) videos, the MPEG-4 codec is modified not to encode any colour data, i.e. only Y- but no U- and V-components are encoded.

Equal quantisation parameters are used for all frame types in order to have all frames compressed at approximately equal quality. Of course, quantisation matrices are different for I-, P- and B-frames – the defaults are used here. For purposes of easier comparison, we will have to measure the PSNR (peak signal to noise ratio) of a whole video sequence. See Section 2.4.1 how this is done.

It might be important that some frames be encoded with higher quality than others because of non-constant error visibility. Nevertheless, we accept this kind of uniform quality measure for the following discussion.

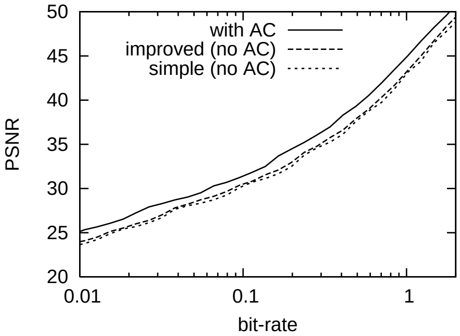
5.1 Plain 3-D Coding

The most common type of wavelet transform in image and video compression is the pyramidal wavelet transform because of its optimal computational complexity ($O(n)$). Although not optimal in terms of rate-distortion performance, there are only minor differences to the optimal wavelet packet decomposition for most images.

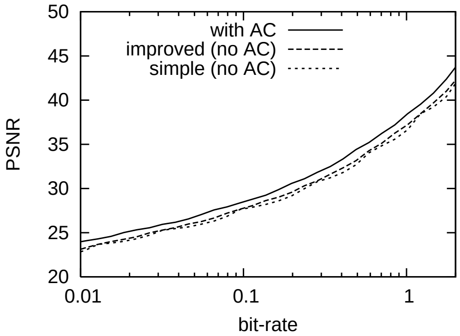
As a first evaluation of SMAWZ with a 3-D pyramidal wavelet transform, let us look at Figure 5.1. It compares the coding results for all three possible algorithms implemented in SMAWZ, i.e. *simple*, *improved* and with *arithmetic coding*. We see that there is the usual difference between the simple and the improved algorithm. What is more surprising about this figure is the fact that arithmetic coding gains an impressive quality improvement. While in image coding [48, 28] the difference is usually about 0.5 dB, we get 1 to 4 dB difference here. The reason might be that 3-D contexts (see Section 4.6.2) deliver more information for the prediction of significance and sign bits than 2-D contexts. Additionally, we see that this effect is greater for sequences with low motion content (*claire*, *container*).

This creates hope for a competitive wavelet codec. In Figure 5.2, the result is compared to MPEG-4. We see that especially for *carphone*, *coastguard* and *claire* the rate-distortion performance is equal to MPEG-4. For *container* and *foreman* we get a loss of up to 2 dB for medium compression rates.

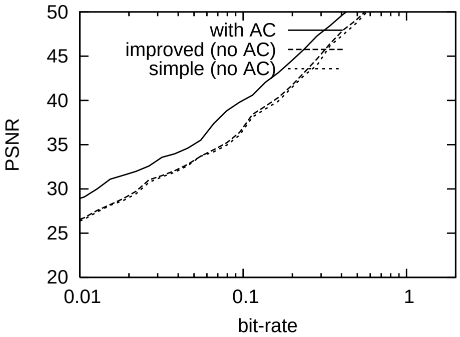
This behaviour obviously does not correlate with the compressibility of the sequence since neither sequences with higher average compression performance nor ones with a lower one necessarily perform better with 3-D SMAWZ. There is also no correlation with the amount of motion in the sequence since e.g. *container* and *claire* contain low motion while *foreman* contains high motion.



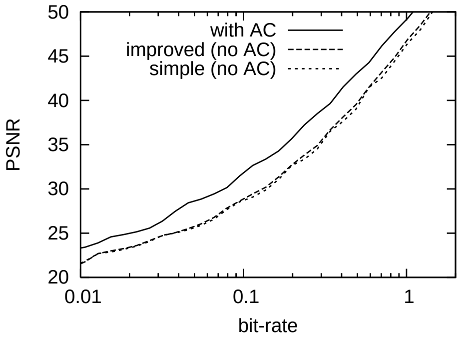
(a) carphone



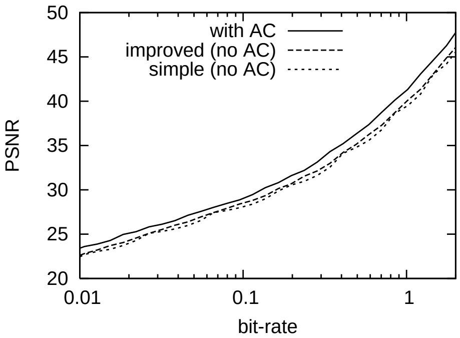
(b) coastguard



(c) claire

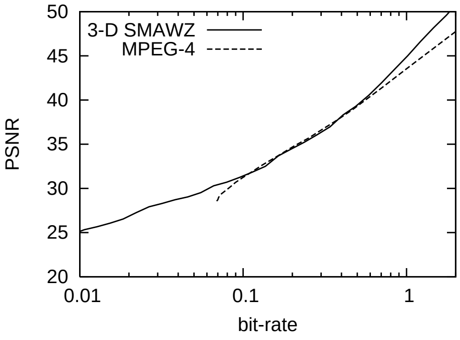


(d) container

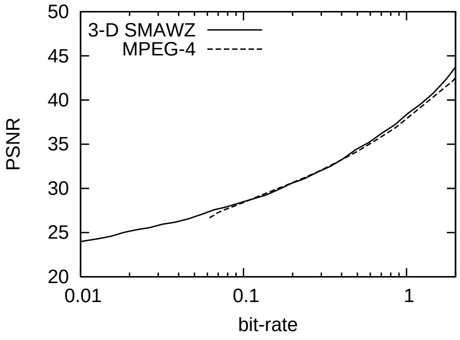


(e) foreman

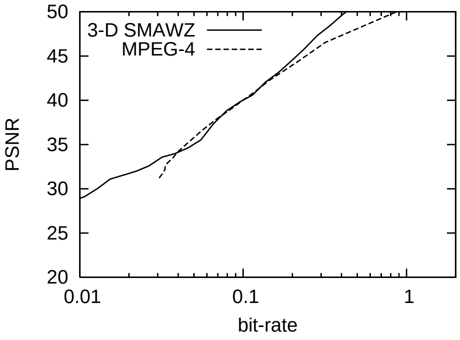
Figure 5.1: Quality comparison of different SMAWZ coding algorithms for 3-D pyramidal wavelet transform



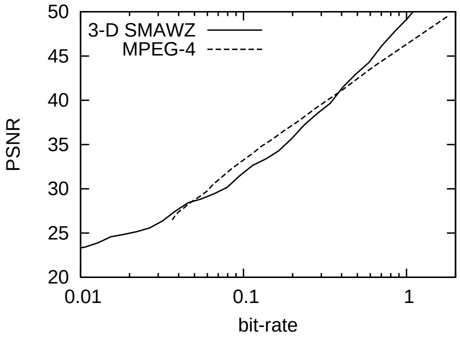
(a) carphone



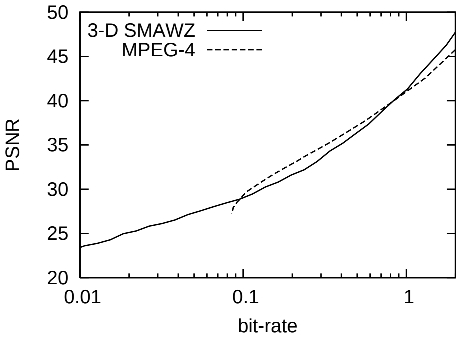
(b) coastguard



(c) claire



(d) container



(e) foreman

Figure 5.2: Comparison of MPEG-4 and SMAWZ using 3-D pyramidal wavelet transform

5.2 Wavelet Packets

When using wavelet packets, the choice of how to find optimal decomposition structures is an important decision. The most elaborate way is probably to use rate-distortion optimisation [47]. However, we choose the less complicated method of cost functions together with the best-basis algorithm (see Section 3.3.1). We will use three different additive cost functions: *entropy*, *logarithmic energy* and *norm* (see Section 3.3.2) and compare the results.

Figures 5.3, 5.4, 5.5, 5.6 and 5.7 show the coding results for all sequences and cost functions. Additionally, the influence of arithmetic coding on the rate-distortion performance is investigated. Each of these figures concentrates on a single video sequence. Within the figures, each row of plots is dedicated to a single cost function. There are two plots in each row depicting the results with (right) and without (left) arithmetic coding. Each plot shows three curves for the pyramidal wavelet transform, wavelet packets and anisotropic wavelet packets. The latter are included for reasons of space saving and are discussed in the next section. For now, we will ignore these (solid) lines.

We see that the wavelet packet decomposition usually can improve the rate-distortion performance by about 0.5 to 1 dB with the exception of the low-motion sequence *container* where the difference is up to 5 dB and the high-motion sequence *foreman* where no improvement can be achieved at all.

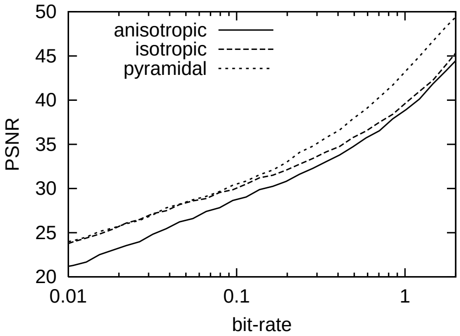
The *norm* cost function turns out to produce the best bases while the *entropy* does not seem to work properly. It even degrades the performance in some cases.

Another astonishing result is that arithmetic coding favours the wavelet packet decompositions. The difference between the pyramidal and the wavelet packet curves is always greater when arithmetic coding is applied. Often, there is a difference introduced between the two curves where they approximately match in the case without arithmetic coding. Sometimes, even a negative difference is turned into a positive (compare Figure 5.6 (a) and (b)).

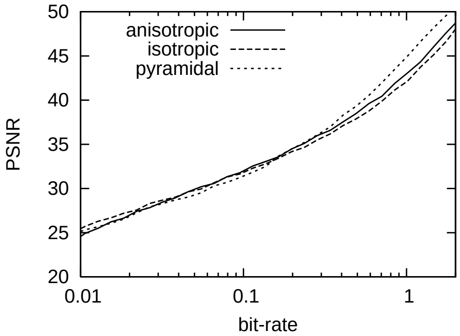
The reason is probably that the problem of finding appropriate similarity trees degrades the positive impact of wavelet packet decompositions on the compressibility – an effect that can be compensated to some extent by arithmetic coding. In complicated wavelet packet decompositions, coefficient trees often lose their localising character and, thus, include many coefficients with small probability of significance. Obviously, these can successfully be detected through their contexts.

In Figure 5.8 the quality improvement of arithmetic coding is depicted directly (for the *norm* as cost function). It is bigger than in the pyramidal case (compare to Figure 5.1) but the behaviour is similar: We usually gain 1 to 2 dB and, in low-motion sequences as *claire* and *container* up to 4 dB.

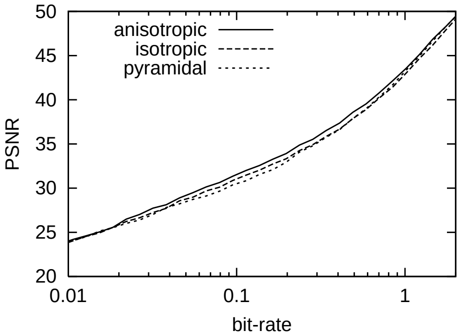
In Figure 5.9, the resulting coding performance is compared to MPEG-4. For the first time, 3-D SMAWZ is able to outperform MPEG-4. Especially for the low-motion sequences, we get an advance of at least 2 dB. However, it is still inferior to MPEG-4 on the *foreman* sequence. This means that motion



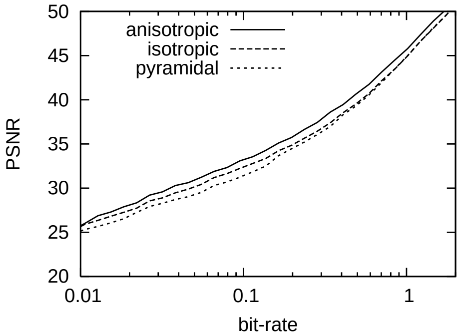
(a) Entropy (no AC)



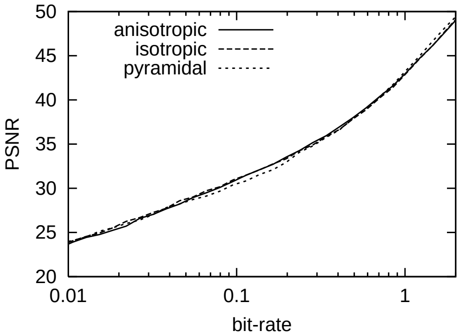
(b) Entropy (with AC)



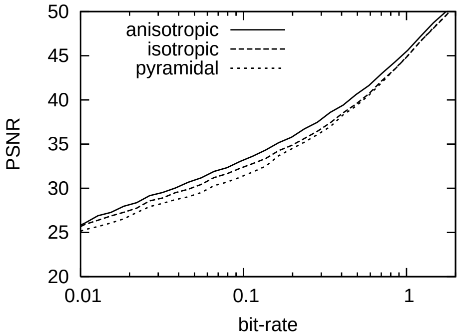
(c) Log. energy (no AC)



(d) Log. energy (with AC)

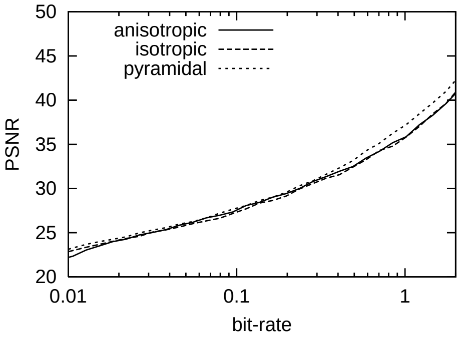


(e) Norm (no AC)

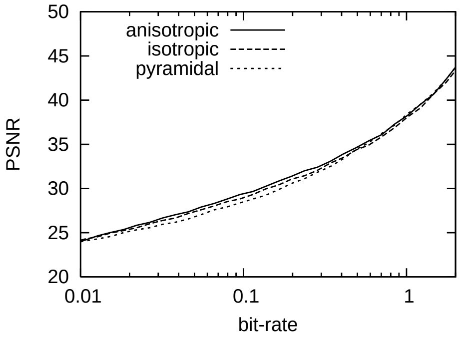


(f) Norm (with AC)

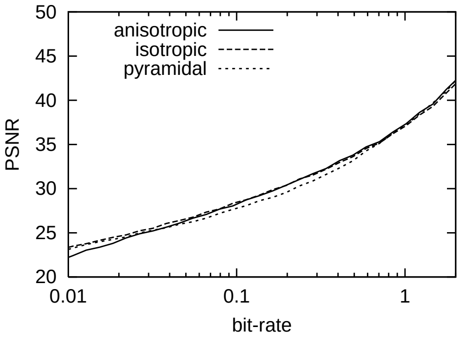
Figure 5.3: Comparison of cost functions for 3-D wavelet packet based SMAWZ for the *carphone* sequence



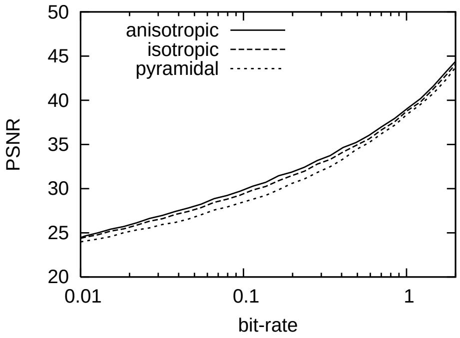
(a) Entropy (no AC)



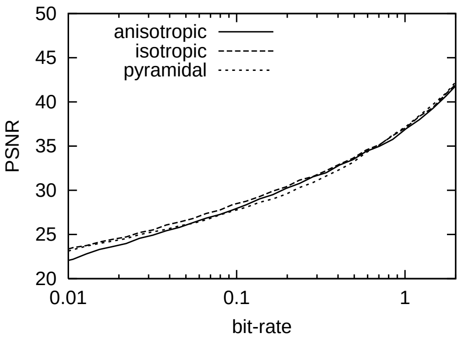
(b) Entropy (with AC)



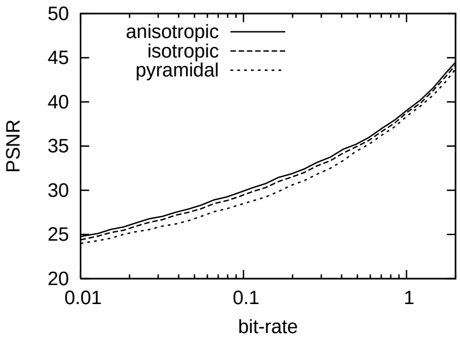
(c) Log. energy (no AC)



(d) Log. energy (with AC)

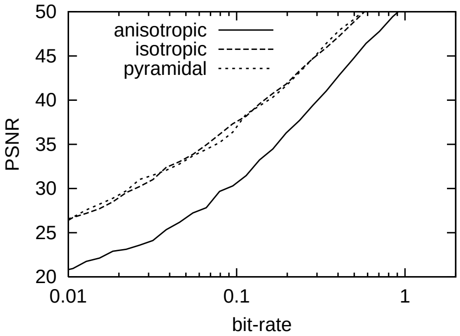


(e) Norm (no AC)

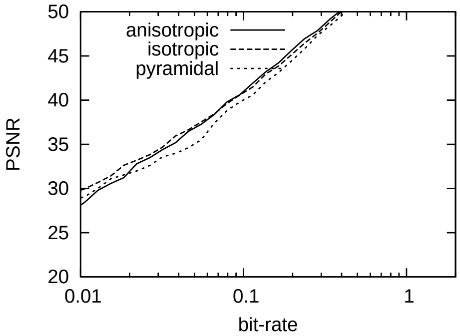


(f) Norm (with AC)

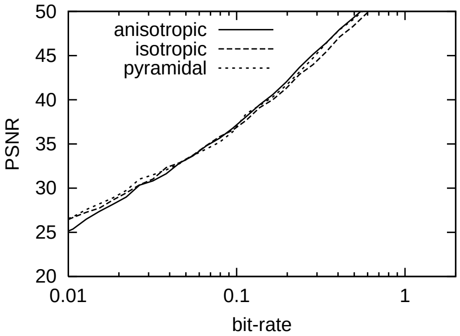
Figure 5.4: Comparison of cost functions for 3-D wavelet packet based SMAWZ for the *coastguard* sequence



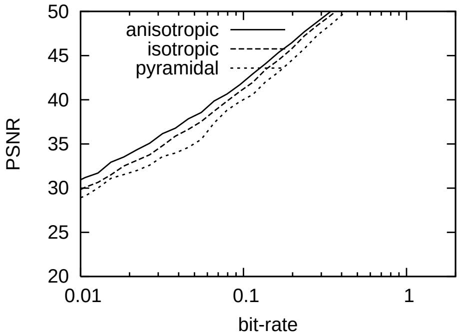
(a) Entropy (no AC)



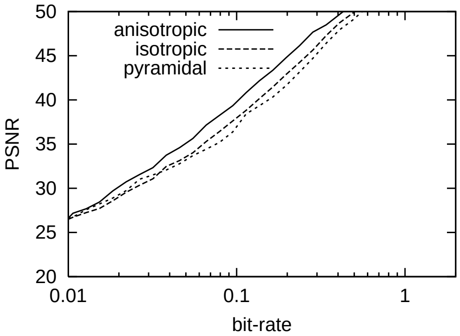
(b) Entropy (with AC)



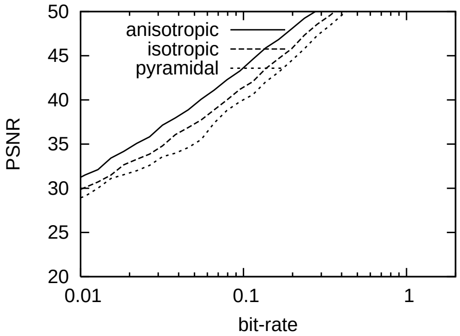
(c) Log. energy (no AC)



(d) Log. energy (with AC)

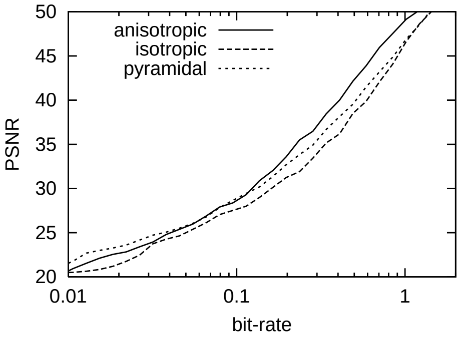


(e) Norm (no AC)

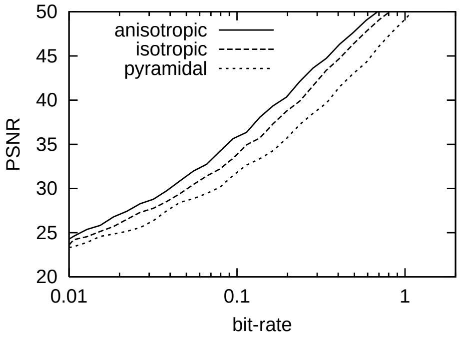


(f) Norm (with AC)

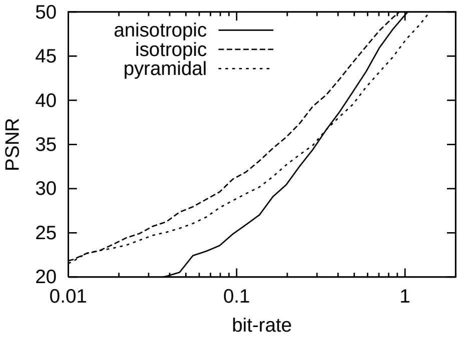
Figure 5.5: Comparison of cost functions for 3-D wavelet packet based SMAWZ for the *claire* sequence



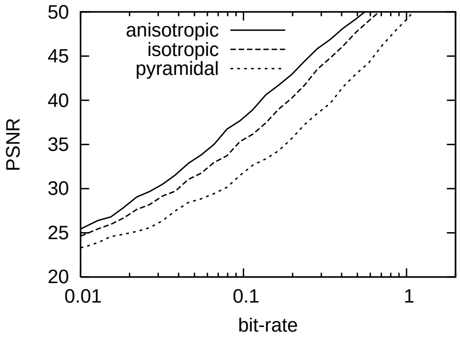
(a) Entropy (no AC)



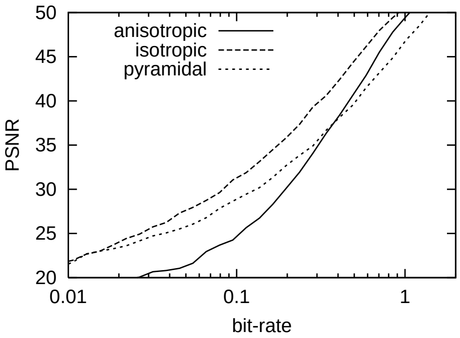
(b) Entropy (with AC)



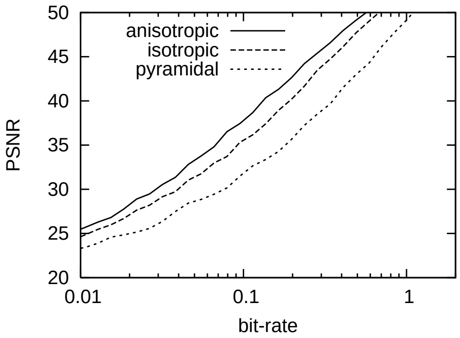
(c) Log. energy (no AC)



(d) Log. energy (with AC)

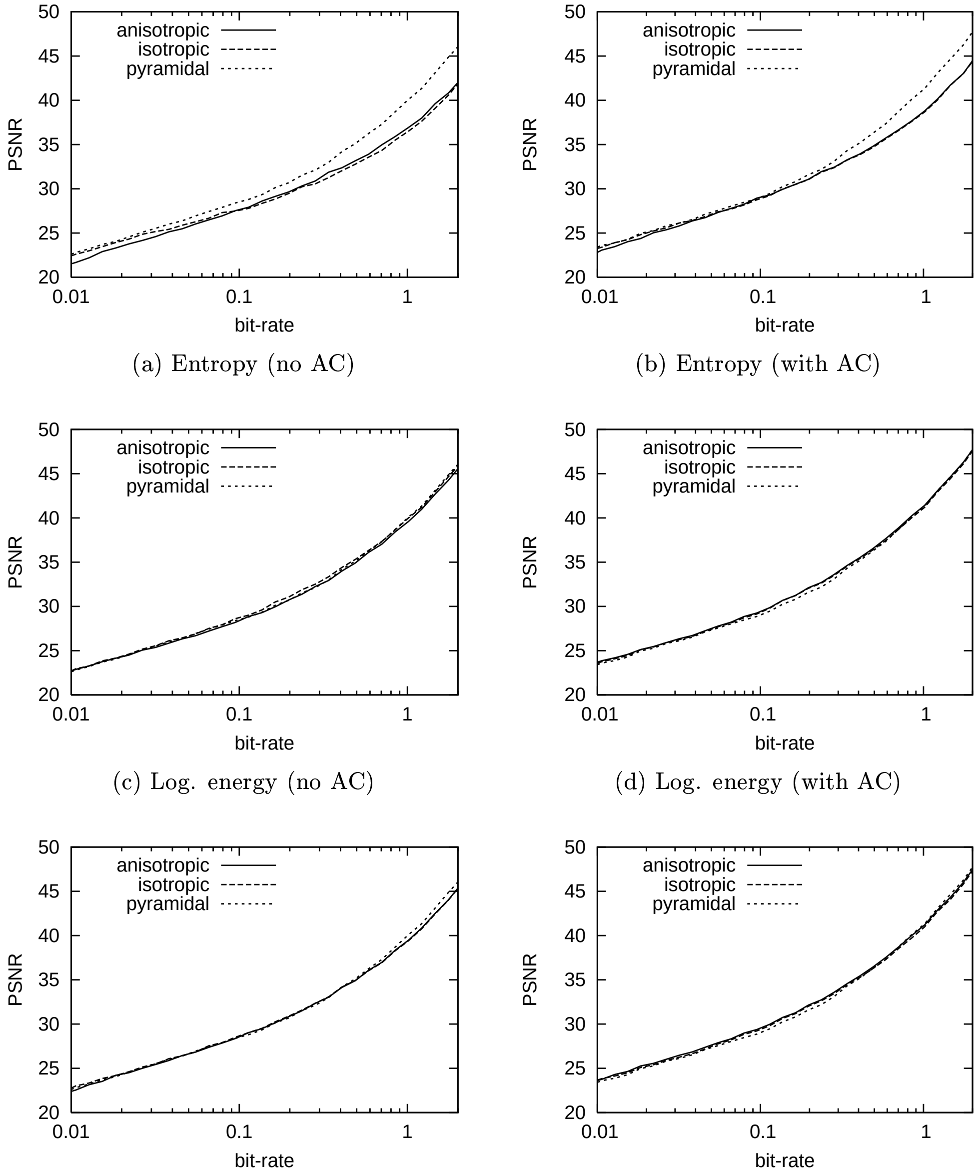


(e) Norm (no AC)



(f) Norm (with AC)

Figure 5.6: Comparison of cost functions for 3-D wavelet packet based SMAWZ for the *container* sequence



(a) Entropy (no AC)

(b) Entropy (with AC)

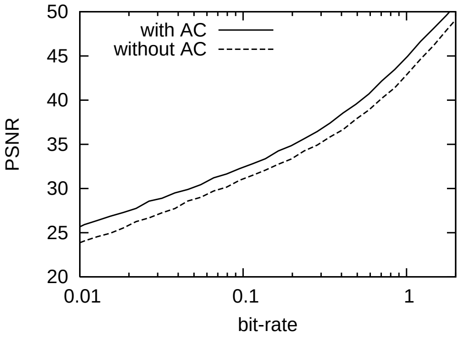
(c) Log. energy (no AC)

(d) Log. energy (with AC)

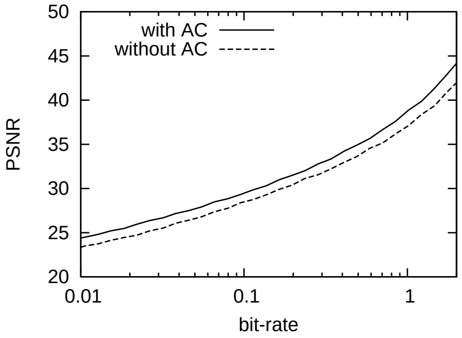
(e) Norm (no AC)

(f) Norm (with AC)

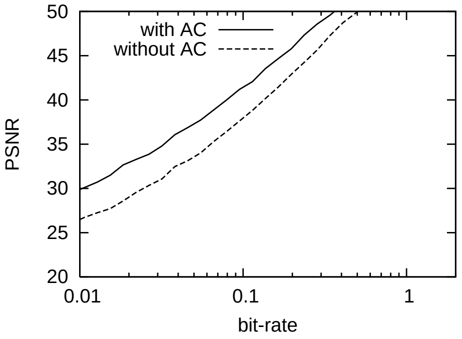
Figure 5.7: Comparison of cost functions for 3-D wavelet packet based SMAWZ for the *foreman* sequence



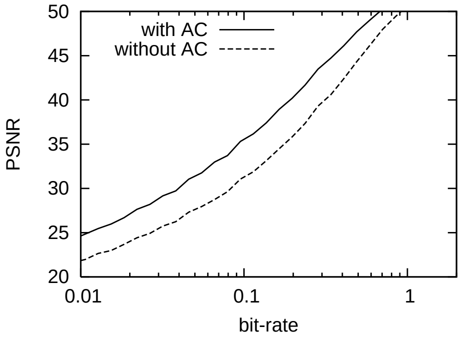
(a) carphone



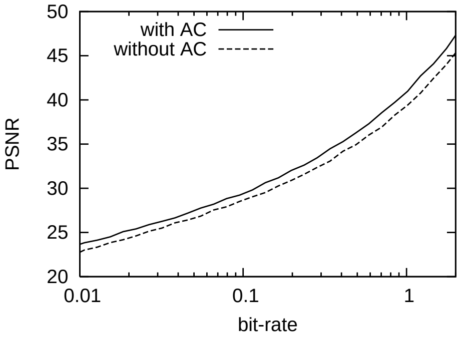
(b) coastguard



(c) claire

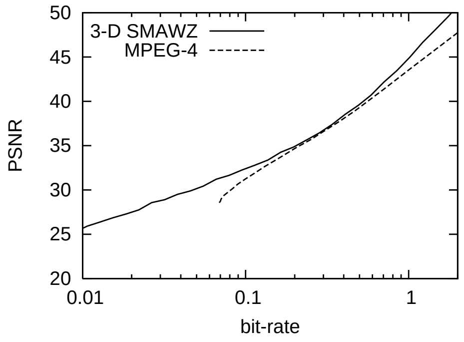


(d) container

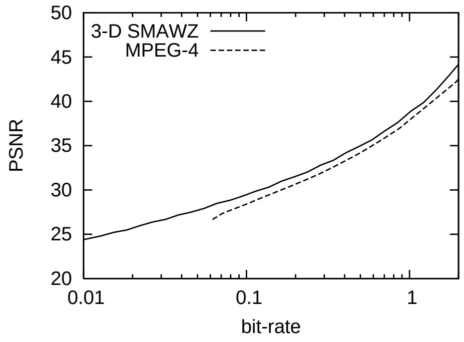


(e) foreman

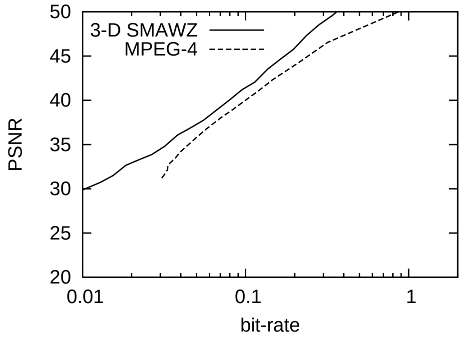
Figure 5.8: Quality comparison of different SMAWZ coding algorithms for 3-D wavelet packets. The logarithmic energy is used as cost function



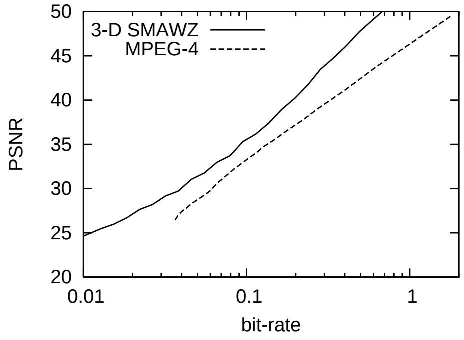
(a) carphone



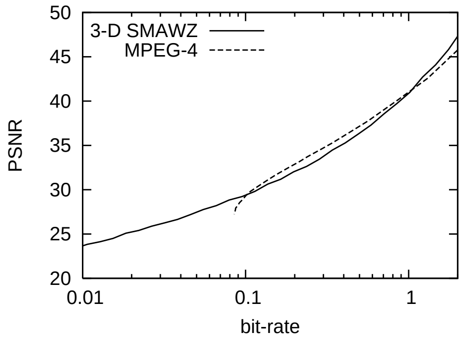
(b) coastguard



(c) claire



(d) container



(e) foreman

Figure 5.9: Comparison MPEG-4 against SMAWZ using 3-D wavelet packets. The norm is used as cost function

compensation is still better on high-motion content than a 3-D wavelet based approach.

5.3 Anisotropic Wavelet Packets

A motivation for the use of anisotropic wavelet packets is that they are actually used in a special way, for instance in [37]. Often, a set of frames is first decomposed in the time direction to get frames containing temporal low- and high-frequency components. These frames are then encoded in the usual way, i.e. by a 2-D wavelet transform followed by a 2-D wavelet codec. The decomposition structure that underlies this method is actually an anisotropic wavelet transform. This section will, among other things, deal with the question if this type of anisotropic wavelet transform is well suited for compression.

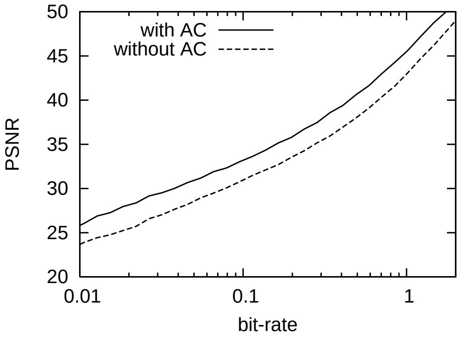
We can now concentrate on the solid lines in the Figures 5.3, 5.4, 5.5, 5.6 and 5.7 which we ignored in the last section. They depict the coding results for anisotropic wavelet transforms produced by the anisotropic best basis algorithm developed in Section 3.4.3.

Basically, the behaviour of the wavelet packet transform is repeated for anisotropic wavelet packets. There are slight improvements to the isotropic case. Again, the *entropy* cost does not work well and is counter-productive in many cases. Unfortunately, the coding algorithm appears to be very sensitive to sub-optimal decomposition structures. Figure 5.6 (c) and (e) show extremely bad results for the *container* sequence. The problem must lie in the difficulty to find appropriate similarity trees for anisotropic wavelet packet transforms.

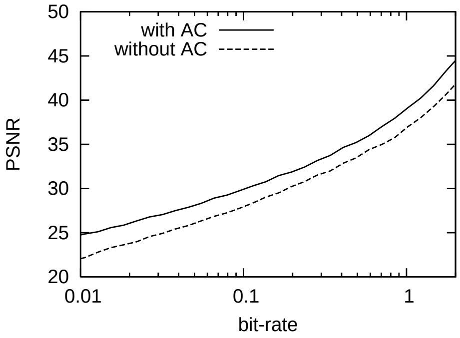
However, there is again the effect that the arithmetic coder benefits very much from this kind of wavelet transform. It manages to produce at least a small improvement in quality for all sequences. Again, the low-motion sequences give the greatest improvements, including the (problematic) *container* sequence (see above). This is also shown in Figure 5.10.

Finally, we can compare the results of SMAWZ using anisotropic wavelet packets with MPEG-4 (see Figure 5.11). The quality gain of SMAWZ appears to be about 1 dB for medium motion content (*carphone* and *coastguard*), 4 dB and above for low motion content and a quality loss of 0 to 1 dB for high motion content (*foreman*). The latter seems to be "invincible" as for 3-D wavelet compression.

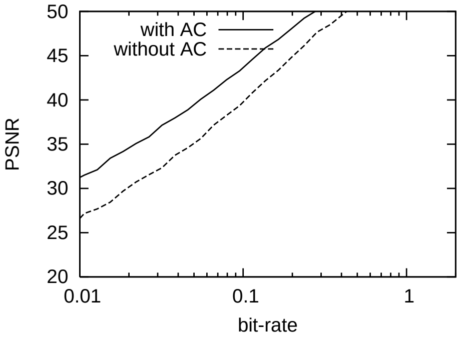
Let us now look at Figure 5.12 (a) which shows the decomposition structure for the *claire* sequence produced by the best basis algorithm using the *norm* as cost function. It seems that the temporal low-pass sub-bands are always decomposed to the maximum decomposition depth. This is also true for sub-bands that are invisible in Figure 5.12 because they are covered by others. This closely resembles the idea from the beginning of this section: a pyramidal temporal decomposition followed by a 2-D decomposition of the frames. Therefore, we will take another chance and evaluate this special kind of anisotropic decomposition.



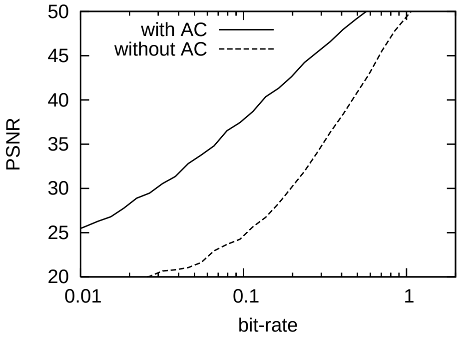
(a) carphone



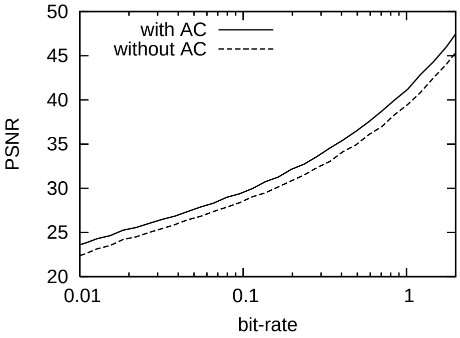
(b) coastguard



(c) claire

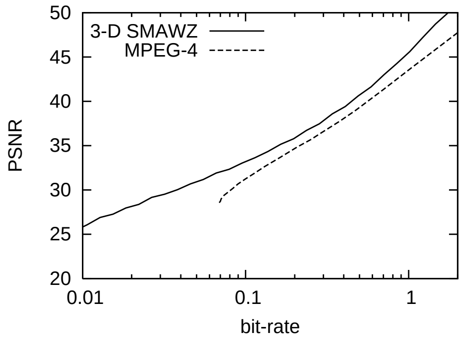


(d) container

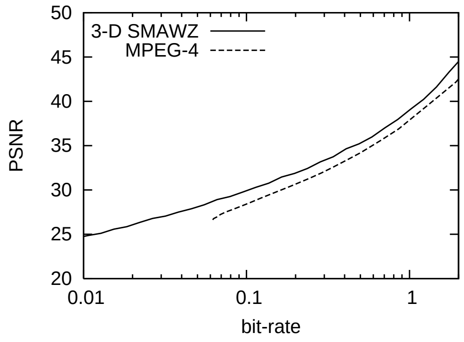


(e) foreman

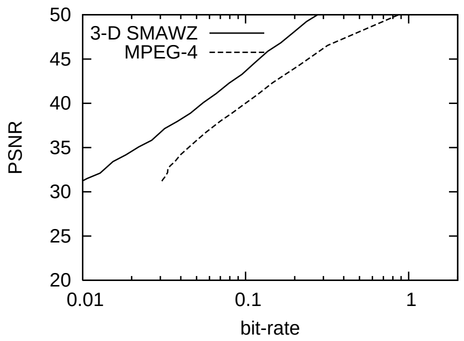
Figure 5.10: Quality comparison of different SMAWZ coding algorithms for anisotropic 3-D wavelet packets. The norm is used as cost function



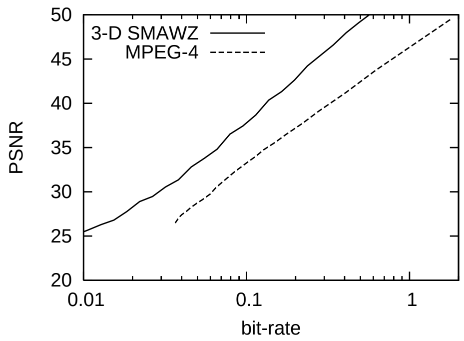
(a) carphone



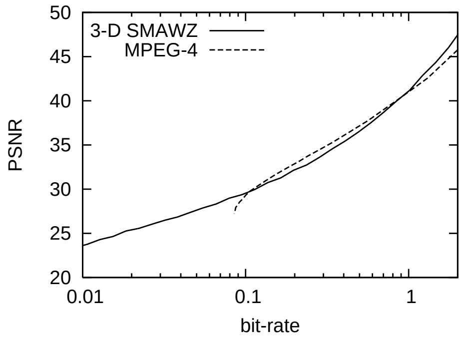
(b) coastguard



(c) claire

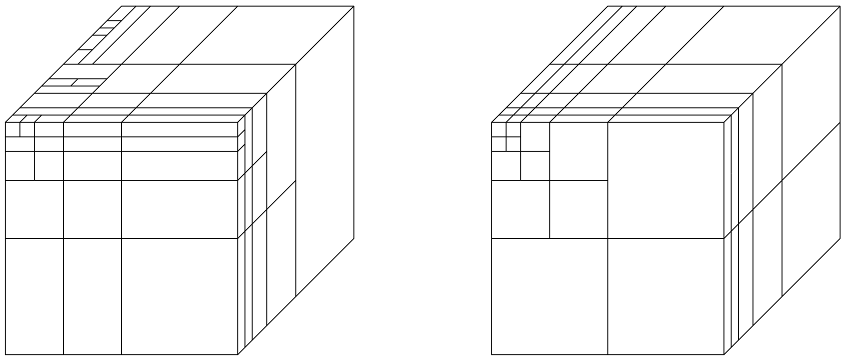


(d) container



(e) foreman

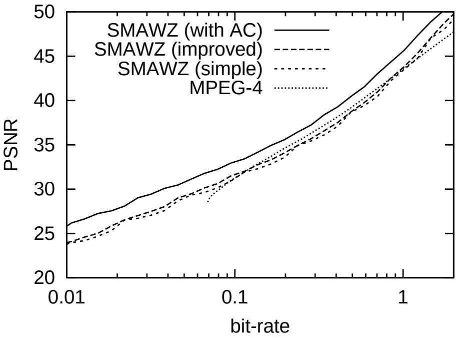
Figure 5.11: Comparison MPEG-4 against SMAWZ using 3-D wavelet packets. The norm is used as cost function



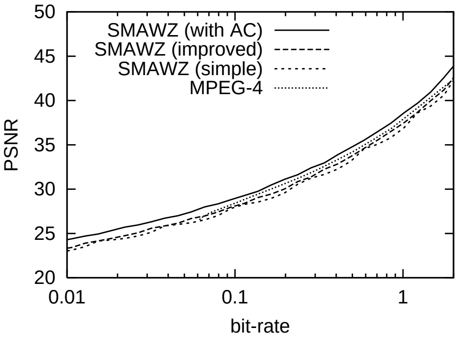
(a) best basis for claire (b) double pyramidal

Figure 5.12: The anisotropic decomposition structure for the *claire* sequence produced by the best basis algorithm using the *norm* as cost function (a) compared to the double pyramidal decomposition (b)

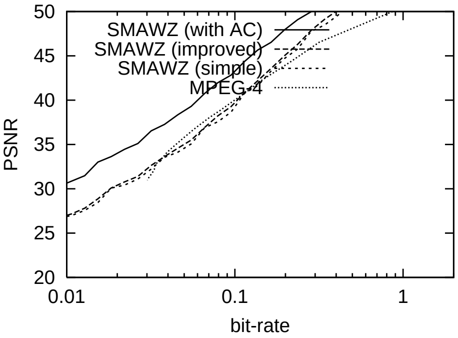
We choose a pyramidal decomposition for the 2-D part as well. Thus, we will call this decomposition *double pyramidal*. (See Figure 5.12 (b) for a visualisation.) The resulting decomposition needs slightly more computation time than the pyramidal decomposition. However, it has still linear complexity. To be precise, the pyramidal decomposition needs about $\frac{24}{7}fn \approx 3.43fn$ multiplies in the 3-D case, where f is the filter length and n is the data size. The double pyramidal decomposition needs about $\frac{14}{3}fn \approx 4.67fn$ multiplies. Figure 5.13 shows the coding results for the double pyramidal decomposition. We see that they are, up to neglectable differences, equal to the best bases for anisotropic wavelet packets (compare to Figure 5.11). Considering the near-optimal computational complexity of this transform, we may conclude that it is the most favourable choice for 3-D wavelet packet video compression.



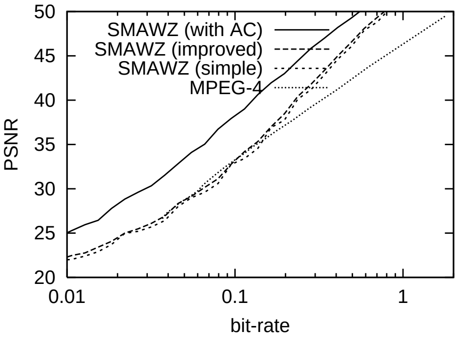
(a) carphone



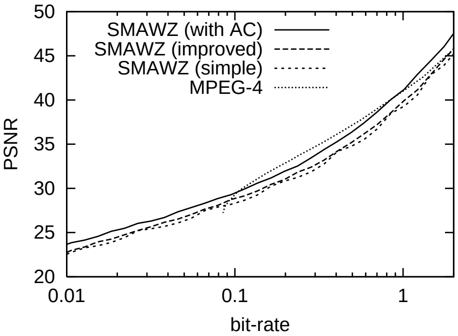
(b) coastguard



(c) claire



(d) container



(e) foreman

Figure 5.13: Evaluation of the double pyramidal decomposition for SMAWZ

Chapter 6

Video Coding by Motion Compensation in the Wavelet Domain

As mentioned in Chapter 1, wavelet based techniques did not have so much success in video coding. The problem is motion compensation which produces prediction residuals that are not quite suitable for wavelet compression. There are approaches for video coding which do not involve traditional motion compensation. [57, 14, 17, 63] are based on 3-D wavelet transform. [43, 61, 70] apply 1-D wavelet transform along motion threads followed by 2-D frame-wise wavelet transform.

However, motion compensation still seems to be the most practicable technique for video coding. [40] proposes an approach to perform motion compensation in the wavelet domain by using undecimated high-pass sub-bands. This approach is a loose approximation of motion compensation, though. [71] proposes a method of hierarchical backward motion compensation that predicts motion and high-pass coefficients from coarse scale approximations. This method has the advantage that no motion vectors have to be encoded.

In this section we will introduce a new technique of block motion compensation in the wavelet domain (**WBMC**). Groups of wavelet coefficients that are located within a spatial block are combined to wavelet blocks (**WB**). A method to shift such a block by arbitrary vectors directly in the wavelet domain is developed. By this it is possible to apply conventional block motion estimation/compensation techniques in the wavelet domain. The method is compared to conventional motion compensation schemes such as block motion compensation (**BMC**) or overlapped block motion compensation (**OBMC**) as well as to MPEG-4.

6.1 Motion Compensation in the Wavelet Domain

Wavelet compression is known not to gain optimal compression rates when applied to residual frames of conventional MC techniques (see Section 2.3). One reason is that these techniques introduce blocking artifacts which contain high-frequency components.

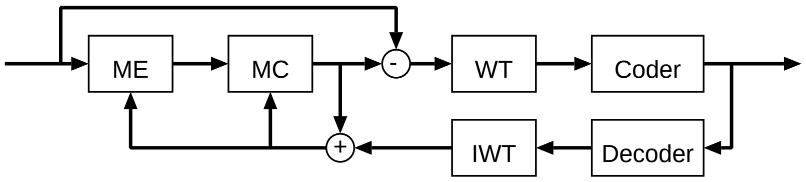
To fully eliminate blocking artifacts that disturb wavelet based compression, it is an ideal solution to assign a motion vector (**MV**) to each wavelet coefficient. If it is possible to predict a wavelet coefficient from this MV and the reference frame, the prediction does not include any blocking or other artifacts that usually result from motion compensation. A brute force method to get predicted wavelet coefficients would be to shift a sufficiently large part of the image (surrounding the corresponding wavelet) and to perform a wavelet transform on this part. This approach is clearly too time consuming and is, therefore, dropped immediately. On the other hand, shifting in the transform domain is difficult since the wavelet transform is known to be highly shift variant (see Section 3.5).

This chapter proposes a method to predict wavelets by a linear combination of neighbouring coefficients. Thus, a limited number of multiplications is needed per coefficient. In this case, "neighbouring" means neighbouring in space as well as in frequency, i.e. also coefficients in other sub-bands serve as source for coefficient prediction. Note that the use of shift-invariant transforms can limit the neighbourhood to a single sub-band [51]. However, these transforms necessarily contain redundancy and are, therefore, not suitable for compression purposes.

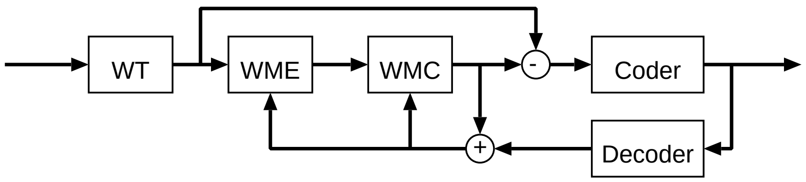
Of course, not every single wavelet coefficient gets a distinct MV because this would mean one MV per pixel. Therefore, wavelet coefficients have to be grouped together to form blocks (wavelet blocks, **WB**s) just as pixels are in conventional BMC. Each WB gets its own MV. This makes conventional motion estimation techniques applicable to the wavelet domain.

As the borders of a WB are not sharp in the pixel domain and wavelets located at the border of blocks overlap with wavelets from neighbouring blocks, WBs can be interpreted as special overlapped windowed blocks as used in OBMC. An important difference to overlapped blocks is the fact that the prediction error of WBs does not depend on neighbouring blocks/MVs. The reason for this is that the MSE is wavelet transform invariant. Therefore, minimising the prediction error of wavelet coefficients also minimises the MSE of the reconstructed image. Thus, it is sufficient to minimise the MSE of a given WB (calculated in the wavelet domain) independently of neighbouring blocks.

Another reason to perform motion compensation and estimation in the wavelet domain is shown in Figure 6.1. Usually, the motion prediction of a video coder is based on a decoded reference frame because this is the best that the decoder has and the difference between the predicted and the actual frame should be the same in the encoder and the decoder. Thus, the encoder has to decompress the frame it has just compressed in order to make it available for the subsequent



(a) Conventional



(b) Based on MC in the wavelet domain

Figure 6.1: Schematic data flow of video encoder

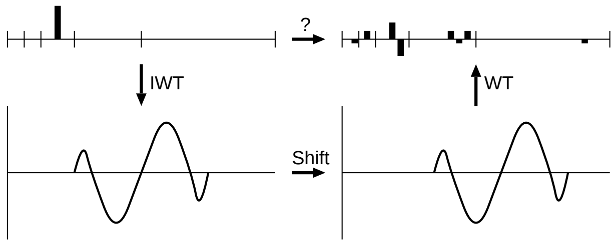


Figure 6.2: Shifting 1-D wavelets

frame prediction. As the wavelet transform is usually the most time consuming part in image compression, it is advantageous if we can avoid wavelet transforms. If the motion compensation is performed in the wavelet domain, the inverse wavelet transform (IWT) of the reference frame is not necessary. In this way, the encoder complexity can be reduced. In [52], this is called a uniform transform domain codec.

6.2 Shifting Wavelets

As part of the wavelet decomposition, sub-bands are scaled down by a factor of 2 in each decomposition step. Accordingly, a sub-band at decomposition level l is down-scaled by 2^l. Now, motion compensation in the wavelet domain involves moving wavelet coefficients by arbitrary vectors. It is easy to shift a wavelet coefficient at level l by a vector whose coordinates are multiples of 2^l because such a vector can be down-scaled by dividing it by 2^l. The coefficient is then simply shifted by the down-scaled vector within its sub-band.

Unfortunately, vectors without this property are more complicated to handle

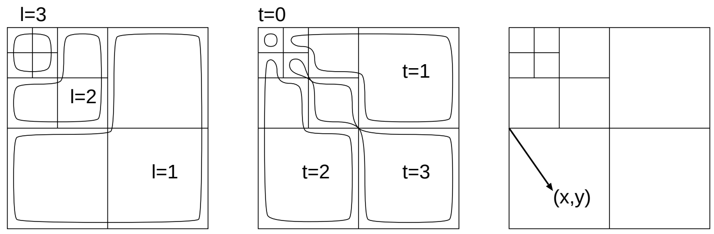


Figure 6.3: Locating wavelet coefficients

as is demonstrated in Figure 6.2 for the 1-D case. A shift by an arbitrary vector causes many neighbouring coefficients even in other sub-bands to be affected by a given coefficient. Its energy is spread over these coefficients. To incorporate this operation in the wavelet domain, the factors by which neighbouring coefficients are affected have to be known. For this purpose, we introduce a shift operator consisting of a set of such factors for all necessary shift vectors, decomposition levels and sub-band types.

First, we have to find a clear notation for coefficient positions. A coefficient can be located by the decomposition level l, the sub-band type t of the sub-band it is part of, and its position (x, y) within the sub-band (see Figure 6.3). Thus, (l, t, x, y) is a fully qualified coefficient position. Now, let $\chi_{(l,t,x,y)}$ be the set of wavelet coefficients that is 1 in (l, t, x, y) and 0 everywhere else. $\lambda_{a,b}$ be the shift of an image by the vector (a, b). A shift of an image in the wavelet domain can then be achieved by the concatenation $\mathrm{WT}\,\lambda_{a,b}\,\mathrm{IWT}$ of operations, i.e. inverse wavelet transform (IWT) to get the original image followed by a pixel-wise shift and a wavelet transform (WT) to change to the wavelet domain again. We can define:

$$M_{a,b}(l, t, x, y, l', t', x', y') := (\mathrm{WT}\,\lambda_{a,b}\,\mathrm{IWT}\,\chi_{(l,t,x,y)})(l', t', x', y')\,.$$

$M_{a,b}$ holds all factors necessary to shift a wavelet coefficient directly in the wavelet domain by a vector (a, b). We can use $M_{a,b}$ to calculate a shift of a set C of wavelet coefficients. This shift is defined as $\mathrm{WT}\,\lambda_{-a,-b}\,\mathrm{IWT}\,C$. The shift of C in the wavelet domain can be calculated as:

$$(\mathrm{WT}\,\lambda_{-a,-b}\,\mathrm{IWT}\,C)(l, t, x, y) = \sum_{l',t',x',y'} M_{a,b}(l, t, x, y, l', t', x', y')C(l', t', x', y')\,.$$

However, this is still not satisfying from a computational point of view for two reasons: First, $M_{a,b}$ has a size of N^2 (where N is the image size) which makes it unusable for video coding. Second, a summation over all coefficients (l', t', x', y') is too time consuming. Fortunately, $M_{a,b}$ owns some properties that are capable of reducing computational and memory requirements drastically.

1. $M_{a+p2^{l'},b+q2^{l'}}(l, t, x, y, l', t', x', y') = M_{a,b}(l, t, x, y, l', t', x' - p, y' - q)$
   This property assures that it is sufficient to keep a and b in the interval

$0 \ldots 2^n - 1$ (where n is the maximum decomposition depth). If e.g. $a \geq 2^n$ then a can be set to $a \mod 2^n$ plus a multiple of 2^n which can be down-scaled to the sub-band (l', t').

2. $M_{a,b}(l-1, t, x, y, l', t', x', y') = M_{2a,2b}(l, t, x, y, l', t', x', y')$
   This property assures that it is sufficient to keep only factors for coefficients at maximum decomposition level $l = n$. For lower levels, the shift operation can be reduced to a scaled shift at level n.

3. $M_{a,b}(l, t, x+p, y+q, l', t', x', y') = M_{a,b}(l, t, x, y, l', t', x'+p2^{l-l'}, y'+q2^{l-l'})$
   This property assures that it is sufficient to limit x and y to 0 for $l' \leq l$ and to $0 \ldots 2^{l'-l} - 1$ else. All other factors can be accessed through properly scaled shifts of (x', y').

4. $M_{a,b}$ is sparse. Therefore, it is advisable to store elements of $M_{a,b}$ in a set of lists indexed by a, b, t, x, y and l'. l is not needed as index because it can be kept equal to n (see above). l' is introduced as index although it is variable. This is to be able to restrict the range of x and y which depends on l'. Therefore, l' has to be varied within a sufficient range when accessing $M_{a,b}$.

5. Computational and memory requirements can further be reduced by discarding elements below a threshold. This reduces list sizes and accelerates calculation. However, it also introduces errors in the results.

After taking these properties into account, the handling of $M_{a,b}$ and, thus, applying shift operations in the wavelet domain is a feasible process. As we will see, it can be used in wavelet domain based block motion compensation.

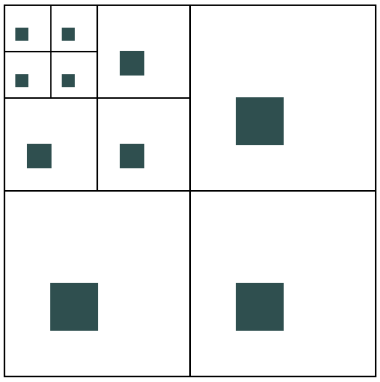
6.3 Wavelet Blocks

The idea behind BMC is that a group of pixels is combined to form a block and to assign a single MV to the whole block. This reduces the bit-rate for the encoding of MVs. If we want to apply motion compensation in the wavelet domain, we have to do something similar. We define a wavelet block (WB) as

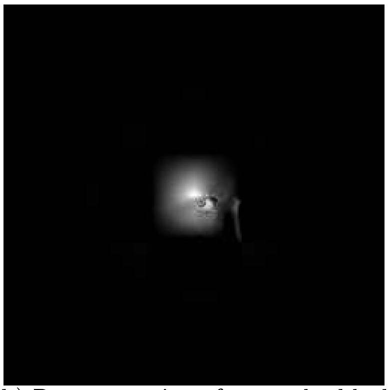
$$B(x_0, y_0) := \left\{ (l, t, x_0 2^{n-l} + x, y_0 2^{n-l} + y) \,|\, 1 \leq l \leq n,\, 0 \leq x, y < 2^{n-l} \right\}.$$

In this way, the block resolution is equal to the resolution of the sub-band at maximum decomposition level, i.e. a WB has a size of 1 coefficient at maximum decomposition level. A WB is visualised in Figure 6.4.

To predict a WB from a reference frame, the following calculation has to be



(a) Wavelet Block (1,2) in a depth 3 decomposition of a 32×32 image



(b) Reconstruction of a wavelet block

Figure 6.4: Example wavelet blocks

executed:

$$\begin{aligned}\tilde{C}(B(x_0, y_0)) &:= (\mathrm{WT}\,\lambda_{-a,-b}\,\mathrm{IWT}\,C)(B(x_0, y_0)) \\ &= \left(\sum_{p'} M_{a,b}(p, p')C(p')\right)_{p \in B(x_0, y_0)} .\end{aligned}$$

If C_k are the coefficients of frame k, then the prediction of $C_k(B(x_0, y_0))$ is $\tilde{C}_{k-1}(B(x_0, y_0))$ and the prediction error is $\mathrm{MSE}(C_k(B(x_0, y_0)) - \tilde{C}_{k-1}(B(x_0, y_0))$. The task of wavelet block motion estimation (WBME) is to find the MV (a, b) that minimises the prediction error. All conventional BME methods (full search, 3-step search and iterative methods) can be applied in WBME.

In this work, only full search and a variant of 3-step search is used which can be named m-step search because it uses search grids of 3×3 MVs and m refinement steps to get an overall search range of $\{-(2^m - 1) \dots 2^m - 1\} \times \{-(2^m - 1) \dots 2^m - 1\}$. In the r-th step the search grid has a granularity of 2^{m-r}. Therefore, it is not necessary to calculate prediction errors for coefficients at a level $l \leq m - r$ because these coefficients contain high-frequency components that cannot be perceived by such a coarse search grid.

Note the relationship to the sampling theorem (Nyquist, Shannon) which states that the frequency content of a signal that is sampled at a certain sampling rate can be up to a frequency corresponding to the half sampling rate. In our case, sub-bands at level l contain frequencies of up to $2^{-l}\,\mathrm{pixel}^{-1}$ (except for the approximation sub-band). The search grid of granularity 2^{m-r} pixel implies a maximum frequency of $2^{-m+r-1}\,\mathrm{pixel}^{-1}$ due to the sampling theorem. Thus, $2^{-l} \leq 2^{-m+r-1}$ from which follows that $l > m - r$ which is equivalent to the statement above.

Experiments verify that removing the coefficients in question from error calculation does not degrade the prediction accuracy. The benefit of this is that,

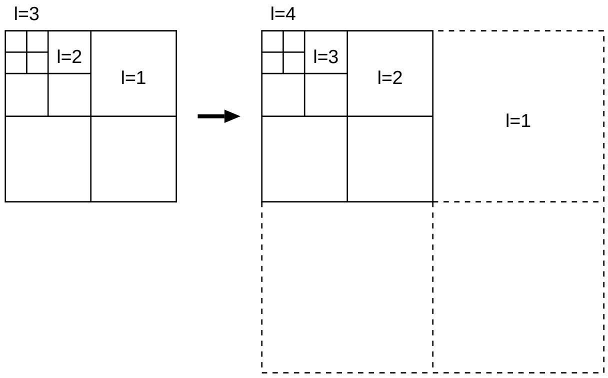


Figure 6.5: How to get half-pixel precision in WBMC

in this way, the complexity of the m-step search in WBME can be decreased.

6.4 Half-Pixel Precision

Half-pixel precision (i.e. motion in $\frac{1}{2}$ pixels) can be achieved easily in WBMC. As each coefficient can be shifted by an arbitrary vector, it can also be shifted by a half pixel. All we need is the proper factors for $M_{a,b}$.

To achieve this formally, we virtually extend the sub-band structure by one high-frequency level (see Figure 6.5). In this way, the level of each sub-band is increased by 1. Therefore, the range of shift vectors doubles and has to be interpreted in $\frac{1}{2}$ pixel steps.

The coefficients of the virtual sub-bands at level 1 are supposed to be 0 before and after the shift. Therefore, they are not included in any calculation and do not increase computational or memory demands.

6.5 Experimental Results

The first 100 frames of the "foreman" video sequence are used for experiments throughout this section. Other video sequences have been tested, with very similar results. The frames of this sequence are of the size 176×144 pixels. The graphs in this section show the quality of the prediction of the k-th frame using the $k-1$-th frame as reference frame. Experiments were conducted on a 1 GHz Pentium III.

WBMC is compared to conventional BMC and OBMC. For OBMC the motion estimation method $RAST$ [53] is used which is non-iterative. It employs a raster-scan on blocks and includes all neighbouring blocks that have previously been accessed in the scan into the calculation of the overlapped prediction error. A bilinear window function is used.

	full search	4-step search
BME	0.48	0.06
OBME (RAST)	6.70	0.84
WBME (threshold 0.1)	11.60	0.58
WBME (threshold 0.01)	44.50	2.40
WBME (threshold 0.1, half-pixel)	12.00	1.00
WBME (threshold 0.01, half-pixel)	46.00	4.50

Figure 6.6: Timing of wavelet block motion estimation (in seconds)

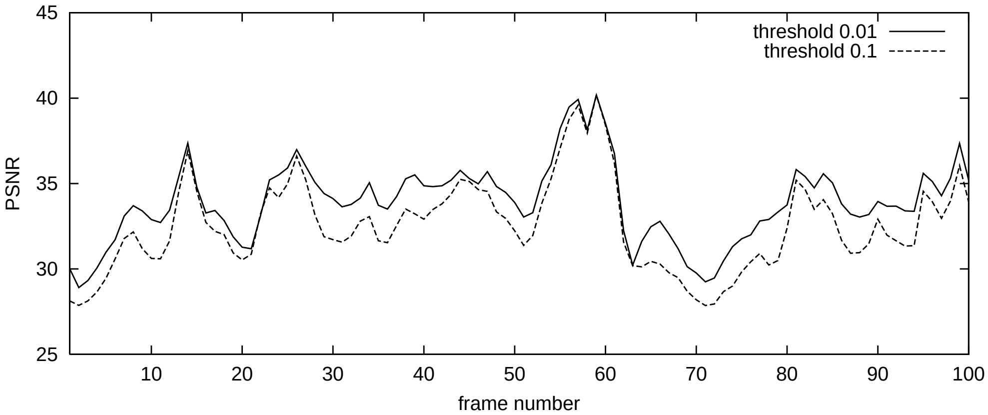


Figure 6.7: Influence of the shift factor threshold on the prediction PSNR

The search range for MVs is $\{-15\ldots15\} \times \{-15\ldots15\}$. Thus, the m-step search needs 4 steps. This corresponds to the wavelet decomposition depth 4.

6.5.1 Timing Performance

First, let us have a look at the execution times of the used algorithms (see Figure 6.6). Unfortunately, WBME seems to be slow, especially for a low threshold for the shift factors $M_{a,b}$. Having a look at Figure 6.7 tells us that a low threshold is necessary to achieve good prediction quality. This increases the number of factors used in shift operations, which is the reason for higher execution times.

6.5.2 Prediction Quality

Figure 6.8 compares the quality of the three motion compensation methods for the full search motion estimation. Unfortunately, WBMC loses again. However, the advantage of WBMC should be that residuals can be compressed more easily by a wavelet codec. Figure 6.9 shows similar results for the 4-step search. Note that the quality difference between full search and 4-step search is not significant. Therefore, we will use 4-step search from now on.

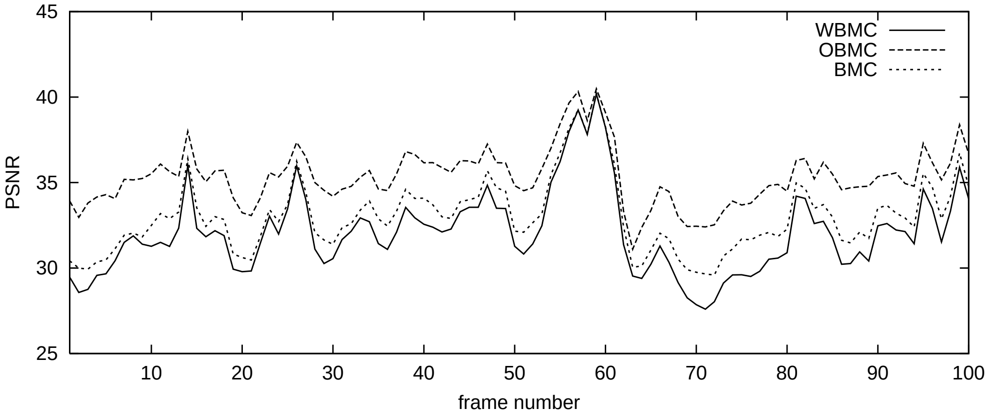


Figure 6.8: Prediction quality for full search of MVs

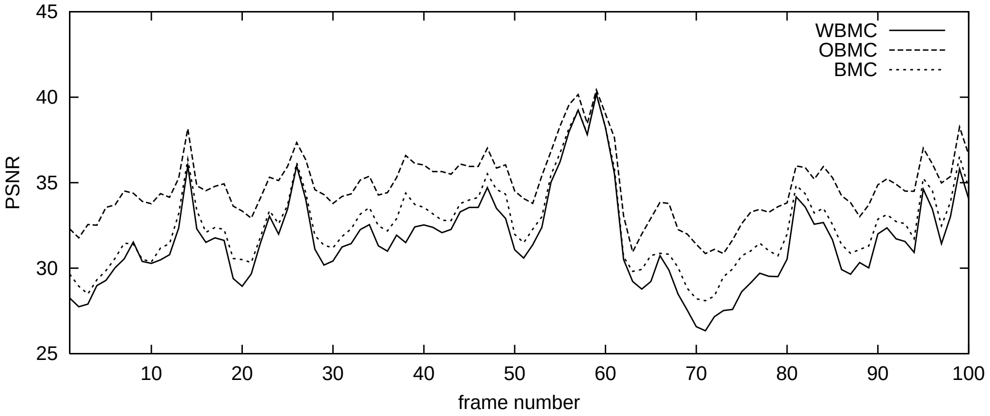


Figure 6.9: Prediction quality for 4-step search

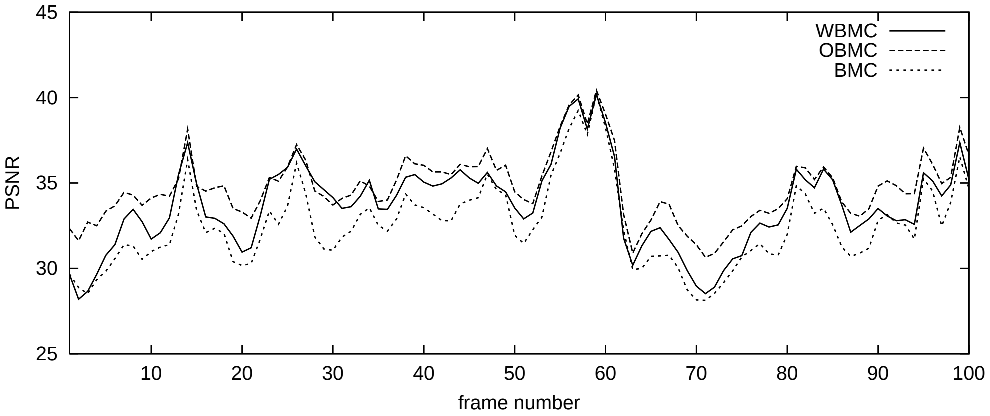


Figure 6.10: Prediction quality with half-pixel precision (4-step search)

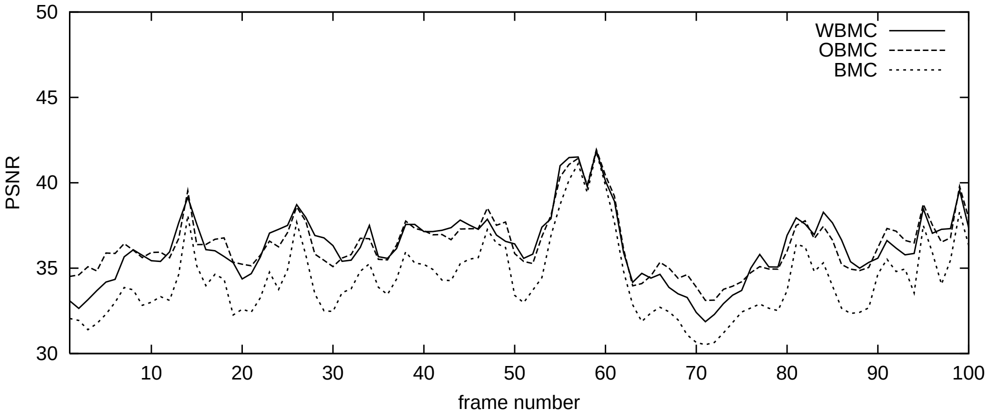


Figure 6.11: Coding performance of residuals at 0.1 bpp

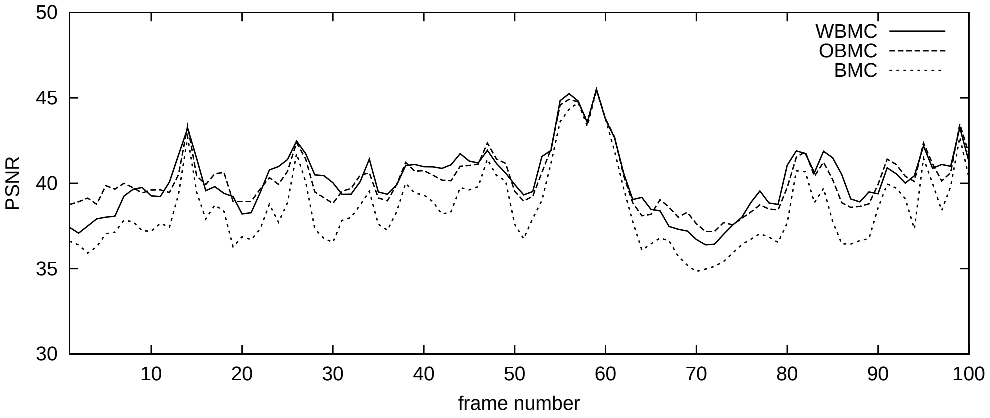


Figure 6.12: Coding performance of residuals at 0.5 bpp

Results for half-pixel precision are much better, as one can see in Figure 6.10. Therefore, we will use half-pixel precision from now on.

6.5.3 Coding Performance

To show the performance of a wavelet image codec when used on WBMC residuals, we use the SMAWZ codec (see Chapter 4). The difference between an actual frame and a predicted frame is encoded at a certain bit-rate. It is then decoded again and added to the predicted frame. The result is compared to the actual frame.

Figure 6.11 and Figure 6.12 show the results for bit-rates of 0.1 and 0.5 bpp. Very often, the PSNR of WBMC is higher than the PSNR of OBMC for frames where it was lower before encoding the residuals. This shows that WBMC residuals can be encoded more efficiently by a wavelet codec than OBMC residuals.

However, Figure 6.13 shows that SMAWZ is not able to encode motion residuals efficiently enough. The plots represent the coding performance for an IPPPP scheme. The PSNR is calculated over the $176 \times 144 \times 64$ pixel array of the first 64 frames of each video sequence (see Section 2.4.1). As MPEG-4 codec, the implementation of the verification model from December 1999 is used. Because we only consider monochrome (grey) videos, the MPEG-4 codec is modified not to encode any colour data, i.e. only Y- but no U- and V-components are encoded. For both codecs the PSNR of I- and P-frames is kept approximately equal – which should maximise the overall PSNR. We see that MPEG-4 outperforms SMAWZ/WBMC for all types of videos (low-motion *claire* and high-motion *foreman*).

6.6 Future Work

Considering the fact that, although optimal from a theoretical point of view, WBMC cannot outperform conventional OBMC and produces bad coding performance compared to MPEG-4 in a rate-distortion sense as well as in a timing sense, there are several ideas to improve the situation.

- There are modified wavelet transforms [19] with better shift invariance. This should make the handling of shift operations in the wavelet domain easier and faster.

- There exist motion estimation techniques based on wavelets which could be used to find MVs [52, 38, 39].

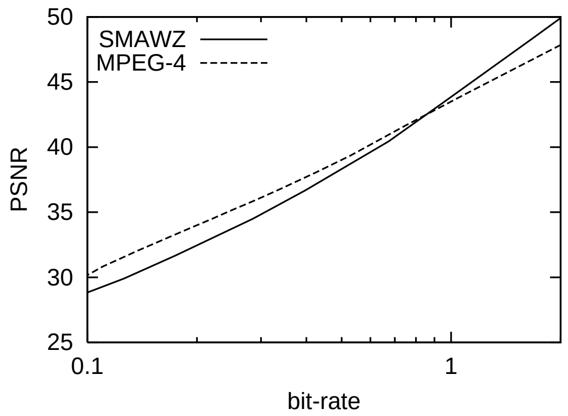
- Zero-tree coding does not seem to be a good choice for encoding residual frames because it requires high-frequency sub-bands to contain less energy, which is not true for residuals. Other wavelet codecs (e.g. JPEG2000) could still perform better.

- Throughout this work, the Daubechies 7-9 tap biorthogonal wavelet is used. Different wavelets might produce better results. Note that the WBMC scheme is adaptable to all kinds of wavelets.

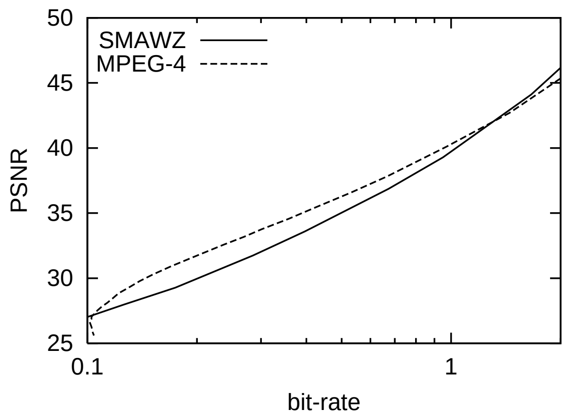
- The hierarchical scan order of m-step search is very similar to the way zero-tree image codecs work. They could be combined to a video codec that integrates motion compensation and residual encoding into an embedded bit-stream.

6.7 Conclusions

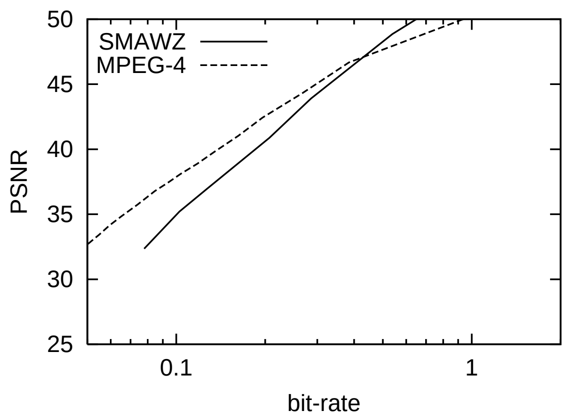
A novel motion compensation method (WBMC) was presented. The motion compensation is solely based on blocks of wavelet coefficients. For this purpose, a



(a) carphone



(b) foreman



(c) claire

Figure 6.13: Comparison of rate-distortion performance of SMAWZ with WBMC against MPEG-4

method to shift wavelet coefficients directly in the wavelet domain was developed.

Unfortunately, WBMC cannot achieve the timing performance and prediction accuracy of overlapped motion compensation. However, an easy method to include half-pixel precision can be applied, which makes WBMC competitive. Moreover, experiments show that the prediction residuals are more suitable for wavelet based compression than residuals of conventional motion compensation. Nevertheless, WBMC together with SMAWZ is not able to outperform MPEG-4.

Chapter 7

Parallelisation

Video compression often shows prohibitive computational and memory demands (especially for real-time applications). Therefore, MIMD (multiple instruction multiple data) architectures seem to be an interesting choice for such algorithms.

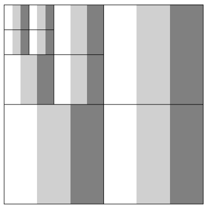
A significant amount of work has already been done on parallel wavelet transform algorithms for all sorts of high performance computers. We find various kinds of suggestions for 1-D, 2-D and 3-D algorithms on MIMD computers for decomposition only [67, 55, 45, 13, 22, 20, 30, 31, 32, 23] as well as in connection with image compression schemes [33, 11]. Therefore, this chapter concentrates on the parallelisation of the coding algorithm that follows the wavelet transform.

7.1 Parallelisation of Zero-Tree Codecs

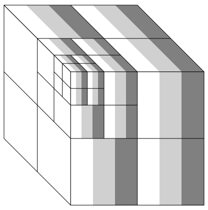
A parallelisation of the EZW algorithm – an important zero-tree coding scheme – is presented in [11] where two approaches are proposed: One is a straight-forward parallelisation which performs the EZW algorithm locally on each processing element (PE) for distinct blocks. This makes the resulting bit-stream (BS) incompatible with the sequential algorithm. The other approach dedicates one PE to the collection of the symbols that have to be encoded. This PE performs a reordering of the symbols before it encodes them. This approach is similar to the approaches presented in this work and (as far as the author understands) is compatible with the sequential EZW.

An improvement of the EZW is the SPIHT algorithm [48]. However, this algorithm makes use of lists of coefficients, which makes it hard to parallelise. This means that although consecutive list entries initially point to neighbouring wavelet transform coefficients, the algorithm jumbles the entries after a while. Thus, there is no easy data driven parallelisation.

An approach to *packetise* the SPIHT algorithm is presented in [65]. This is similar to the first approach in [11]. The output of several executions of SPIHT on spatial blocks is multiplexed based on a rate allocation technique that approximates distortion reduction by fast SPIHT-specific statistics. However,



(a) 2-D case



(b) 3-D case

Figure 7.1: Distribution of coefficients or list entries after parallel wavelet transform. Different colours indicate different PEs

the resulting bit-stream (BS) is not compatible with sequential SPIHT.

This chapter concentrates on the parallelisation of the SPIHT and SMAWZ algorithms (which are quite similar). Several "tricks" have to be found to overcome parallelisation difficulties for the SPIHT algorithm while SMAWZ involves a more spatially oriented coefficient scan order and, thus, avoids the problems of SPIHT parallelisation.

7.1.1 Parallel Wavelet Transform

To perform the wavelet transform in parallel, the data has to be distributed among the PEs in some way. In this work, data is splitted into slices (in the time domain). The filtering is performed in parallel on local data. Border data has to be exchanged before each decomposition step between neighbouring PEs due to the filter length. After that, transformed data is found distributed as shown in Figure 7.1.

In contrast to the parallelisation of the wavelet transform as presented in [29], the parallel wavelet transform used here dispenses with video data distribution as well as collection of transformed data. Initial data distribution is not necessary because input is performed in parallel (i.e. each PE reads its own part of the video data). Note that the speedups reported in this work do not include I/O operations as I/O is not viewed as a part of the algorithm. Another advantage of this is that we can drop the host-node paradigm because there is no extra single PE responsible for data distribution. Likewise, the collection of transformed data is not necessary because data is passed on to the coding part of the algorithm which is also performed in parallel. There is no redistribution of data required as we will see in Section 7.2 and Section 7.3.

7.1.2 Zero-Trees in Parallelisation

Zero-trees can be viewed as a collection of coefficients with approximately equal spatial position. While this fact implies that the coefficient's significances are statistically related, which is exploited by the SPIHT and SMAWZ algorithms, this also means that zero-trees are local objects corresponding to the data dis-

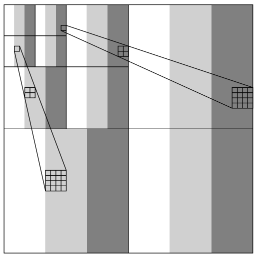


Figure 7.2: After parallel decomposition, data is distributed in a way so that each zero-tree resides on a single PE

tribution produced by the parallel wavelet transform (see Figure 7.2). This can be exploited by the parallelisation of the zero-tree algorithms (see Section 7.2 and Section 7.3).

7.2 SPIHT Parallelisation

When parallelising the SPIHT algorithm (Section 4.4), we have to face the problem that it uses lists of coefficient positions and is, therefore, inherently sequential. The reason is that it is hard to perform general list operations on distributed lists. Nevertheless, the set of distinct list operations involved in the SPIHT algorithm is limited. This enables us to develop an efficient way to manage distributed lists of coefficients.

7.2.1 Separators

The basic operations of the algorithm are: Moving an iterator all through a list, deleting elements at iterator position and appending elements at the end of a list. The aim is to distribute the list so that each PE-local entry corresponds to a local coefficient where coefficients are distributed among the PEs as shown in Figure 7.1.

This is a simple task for initial distribution. However, as coefficients are appended to the end of lists, one has to provide a mechanism to indicate which parts of a list belong to which PE – or from a PEs view: where a sequence of local coefficients ends and parts of another PEs list should be inserted. This work is done by separators (see Figure 7.3).

The idea is to insert a separator at the end of each part of the list which entirely belongs to a single PE. Initially, the approximation sub-band is split into equal slices. Each slice is assigned to a single PE. On each PE, the (local) lists LIS and LIP are filled with all coefficients from the PE's (local) slice, and a separator is appended to the end of each list. From here on, the sequential algorithm is performed locally with one exception: Each time the iterator meets a separator, the separator is copied to the end of each destination list. A destination list

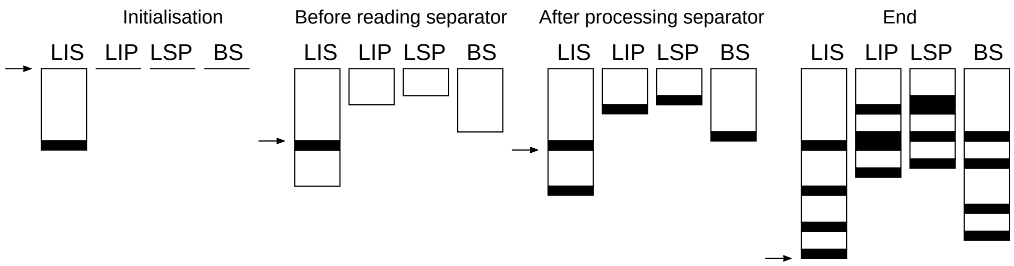


Figure 7.3: Functionality of separators. Four states of the three lists and the bit-stream while processing the LIS.

is a list into which entries are potentially inserted during the current list processing (see Figure 4.5). Applying this principle, the lists L_i on PE_i are split by separators into parts L_{ij} such that the assembled list $L_{11}L_{21}L_{31}\ldots L_{12}L_{22}\ldots$ is identical to the list the sequential algorithm would produce. The same is true for the bit-stream. This enables the parallel algorithm to assemble the bit-stream correctly after each PE has encoded its part of the wavelet coefficients.

An important question is when the processing of a list (i.e. a refinement step) is completed. Essentially, the procedure can stop if it has processed the last non-separator entry in the list. Unfortunately, this does not guarantee that each PE produces the same number of separators. However, this is a necessary condition for the correctness of the parallel algorithm because, otherwise, the correct order of the list-parts would be lost. Therefore, the global maximum number of separators has to be calculated (which unfortunately synchronises the PEs), and the lists have to be filled up with separators before the algorithm continues with the next refinement step (in fact even before each processing of a list).

As a matter of fact, the number of separators grows exponentially with the number of refinement steps, and very often separators appear in a row together in the list. To avoid unnecessary memory demands, consecutive separators should be kept together in a single entry associated with a counter. This means, if a separator entry (containing a counter) is inserted at the end of a list where a separator entry (also containing a counter) is already sitting, their counters can simply be added.

7.2.2 Algorithm Termination

Another problem is the termination of the whole algorithm. In the sequential case, the process terminates when the required number of bits have been written to the bit-stream. In the parallel case, this test (as it is a global test) can only be executed at the end of a refinement step. Thus, the parallel algorithm potentially generates too much bits which of course decreases the speedup in inconvenient cases. If necessary, superfluous bits can simply be cut off after assembling the bit-stream (due to the nature of the SPIHT algorithm).

The procedure of assembling the bit-stream (after collecting the PE-local bit-streams) is the only sequential part of the algorithm. Unfortunately, it gets more complicated and, therefore, consumes more calculation time when the number of PEs is increased. The result will be a significant decrease in speedup.

7.2.3 Parallel SPIHT Decompression

Note that for the reverse algorithm – the reconstruction of video data from the bit-stream – the methods described above are not applicable. Although it is not part of this work, we can shortly outline the ideas how to implement a parallel SPIHT decoder: First of all, the whole bit-stream has to be copied to all PEs. Also, each PE has to process all bits of the bit-stream, independent of whether they belong to local coefficients. Therefore, the global lists have to be kept at each PE. The only speedup potentials are:

- Not adjusting non-local coefficients which involves floating point operations.
- Keeping consecutive non-local list entries (entries belonging to non-local coefficients) together in a single entry associated with a counter (similar to separators). This is possible because position information is not needed for non-local entries.

Although this seems to be a very simple approach, it does not imply the necessity of process synchronisation and it does not contain a sequential part.

7.3 SMAWZ Parallelisation

The approach described above reveals some drawbacks as e.g. complicated bit-stream handling, additional communication needs and non-neglectable sequential code parts. This is a direct consequence of the fact that the SPIHT algorithm is inherently sequential.

In contrast to the original SPIHT algorithm, the parallelisation of the SMAWZ algorithm (see Section 4.5) is easy. Again, it is based on the fact that after the parallel wavelet transform, data is distributed in a way so that zero-trees are local objects (see Figure 7.2).

All we have to do is to parallelise the inner loop in the procedure ProcessAll (which reads "for p in approximation-subband", see Pseudo code 4) according to the data distribution of the approximation sub-band (see Figure 7.1). All other computations within a refinement step are localised, i.e. PE-local computations do not depend on data of neighbouring PEs. So, no communication is required within a refinement step.

Each PE produces one continuous part of the bit-stream for each refinement step. At the end, these parts have to be collected by a single PE and assembled

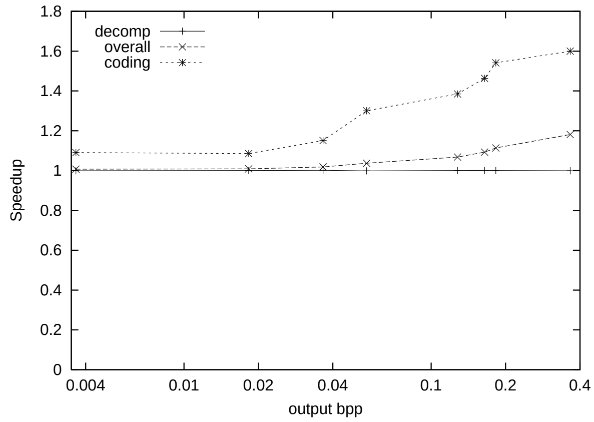


Figure 7.4: Sequential speedup of SMAWZ with respect to SPIHT

properly (i.e. in an alternating way). As in the direct SPIHT parallelisation, this is a major bottleneck. However, the number of bit-stream parts is reduced significantly, which should speed up the bit-stream assembly.

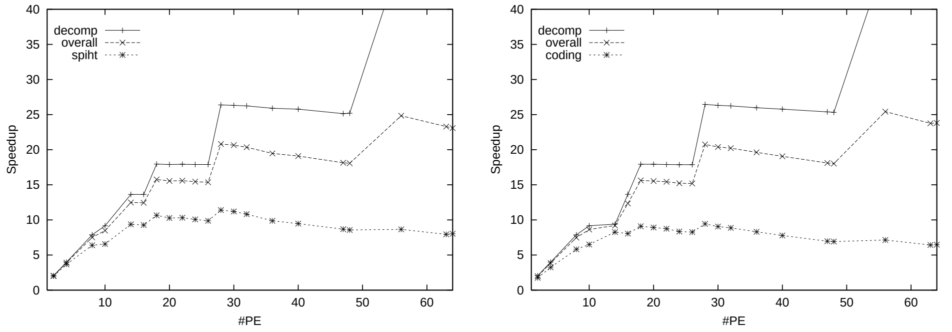
Because of the termination problem (see section 7.2.2) the PEs again have to synchronise at the end of each refinement step to determine if the global number of bits produced so far is sufficient. However, this synchronisation can be dropped if the termination condition is not the bit-stream size but a fixed number of refinement steps. Note that the direct SPIHT parallelisation could not drop the synchronisation in this case because it still has to equalise the number of separators after each refinement step.

7.4 Experimental Results

Experimental results were conducted on a Cray T3E-900/LC at the Edinburgh Parallel Computing Centre using MPI. Video data size is always 864 frames with 88 by 72 pixels. The video sequence used here is the U-part of “grandma”. The wavelet transform is performed up to a level of 3.

Note that the data size is limited by memory constraints in the case when the number of PEs is 1 and a single PE has to hold all video data. The number of frames has to be high to enable uniform data distribution for parallelisation as well as down-scaling for the wavelet transform. Thus, the frame size is small. In a real-world scenario, however, the frame size can be bigger. For the same reason, frame size scalability is difficult to measure. Nevertheless, linear scalability is assumed due to the authors experiences and the fact that the execution time of the coding part does not depend on the video data size but on the number of output bits only.

First of all, we have to look at the sequential performance of the SMAWZ algorithm because if it was slower than the original SPIHT, its parallelisation



(a) SPIHT (b) SMAWZ

Figure 7.5: Speedups for varying #PE and fixed compression rate (0.14 bpp).

would not make sense. However, Figure 7.4 shows that it outperforms the original SPIHT especially for higher bit-rates. On the other hand, this means that it is even harder to get reasonable speedups.

Figure 7.5 shows speedups for a fixed compression rate: 0.14 bpp (bits per pixel, pixels in different frames are counted as different pixels). The fact that the speedup curves are not smooth, i.e. have discontinuities at #PE = 9, 18, 27 and 54, is caused by the divisibility of 864 – the length of the video sequence which determines the size of the local data sub-sets. Note that due to the depth of the wavelet transform (3), this size is divided by 8 and the resulting number is then divided by the number of PEs which is not always possible without remainder.

The sequential bit-stream assembly takes more and more execution time for higher numbers of PEs. Its share in execution time gets higher than 50% of the coding part. Note at this point that in a particular hardware implementation, the bit-stream assembly can be integrated in the output module and separated from the actual coding.

The speedups of the two different algorithms are about the same. This shows that the complicated bit-stream assembly - which is the main problem of a direct SPIHT parallelisation - could be solved efficiently. Some dominant problems, such as PE synchronisation and sequential code parts, are present in both approaches. However, the SMAWZ algorithm is expected to gain a lower parallelisation efficiency for two reasons:

- It owns the same communication overhead while the sequential algorithm is faster.

- There is less potential for positive caching effects in the parallelisation because optimal cache utilisation (due to the spatially oriented coefficient scan) is supposed to be the main reason that the SMAWZ algorithm is faster.

The fact that the parallelisation efficiency is about the same strongly suggests that the bit-stream assembly in parallel SMAWZ is more efficient.

Figure 7.6 shows speedup curves for fixed #PE and varying compression rate. Of course, the execution time of the wavelet decomposition does not depend on the compression rate. The reason for the speedup breakdowns at certain compression rates is the termination problem of the parallel algorithm (see Section 7.2.2). The parallel algorithm is optimal only at compression rates achieved at the end of a refinement step.

Note that although the speedup of the coding part increases with the bit-rate, the overall speedup remains constant or drops slightly because the share in execution time of the coding part increases with the bit-rate.

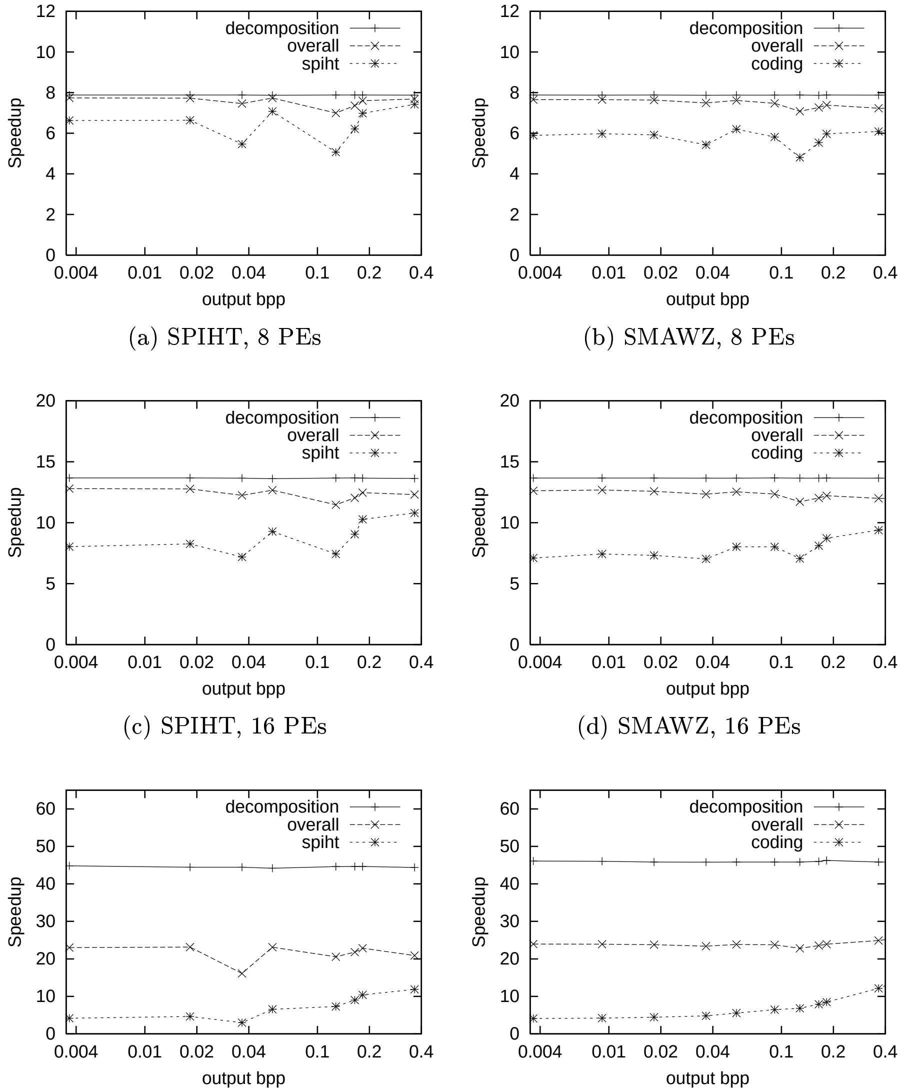
The problem of unevenly distributed complexity is illustrated in Figure 7.7. Here, approximately the first half of the video sequence is substituted by the *carphone* sequence which contains much more motion than the *grandma* sequence. This causes bigger coefficient values at higher frequency sub-bands for the more complex video parts. Thus, more coefficients have to be processed within a refinement step, which makes the algorithm consume more computation time. A load balancing problem is the consequence. One can clearly see that the necessity of process synchronisation at several points in the algorithm leads to an increase of idle times of PEs waiting for other PEs. Figure 7.8 shows the speedups that are – compared to Figure 7.5 – slightly reduced because of this problem.

7.5 Conclusions

We have seen how the inherently sequential zero-tree coding algorithm SPIHT can be parallelised. Although the speedups are not overwhelming, the presented way of parallelisation prevents the necessity to perform the coding sequentially. Thus, reasonable speedups are possible for higher numbers of processing elements and the whole range of compression rates.

There are two methods for parallelisation: Either by the use of so called separators or by applying the alternative SMAWZ algorithm. The first method is more complicated but guarantees compatibility with original SPIHT bit-streams. The second method shows very similar speedup results but better execution times.

Although unevenly distributed image/motion complexity can decrease the speedup potential, this effect seems to keep within limits.

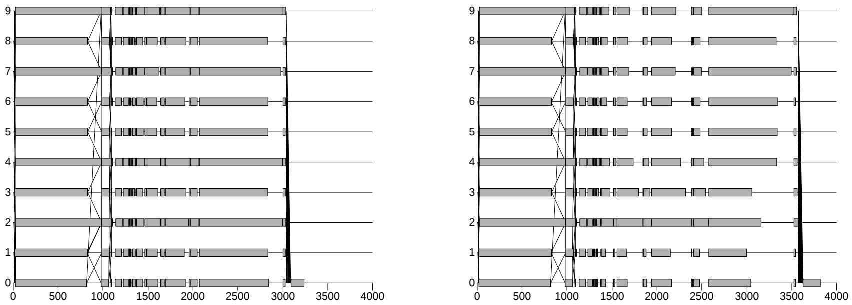


(a) SPIHT, 8 PEs (b) SMAWZ, 8 PEs

(c) SPIHT, 16 PEs (d) SMAWZ, 16 PEs

(e) SPIHT, 64 PEs (f) SMAWZ, 64 PEs

Figure 7.6: Speedup for decomposition, coding and overall speedup for varying compression rate.



(a) evenly distributed complexity

(b) unevenly distributed complexity

Figure 7.7: Execution scheme of decomposition and SPIHT coding for 10 PEs. Time (on horizontal axis) is measured in milliseconds. (Nearly) vertical black lines indicate data transfer. Horizontal grey bars indicate calculation phases.

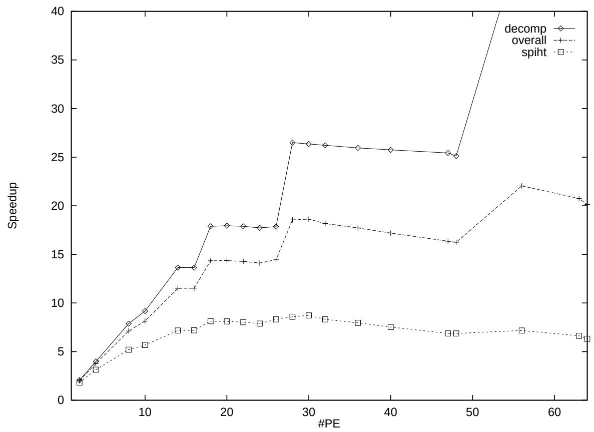


Figure 7.8: Speedups for video with unevenly distributed complexity (at fixed compression rate (0.14 bpp)).

Acknowledgements

The author was supported by the Austrian Science Fund FWF, project no. P13903. The author would also like to acknowledge the support of the European Commission through TMR grant number ERB FMGE CT950051 (the TRACS Programme at EPCC).

Bibliography

[1] K.U. Barthel, G. Ruhl, and T. Voyé. Combining wavelet and fractal coding for 3-D video coding. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'96)*, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society.

[2] C. P. Bernard. Discrete wavelet analysis for fast optic flow computation. *Harmonics Analysis*, 11(1):32–63, July 2001.

[3] Jonathan N. Bradley, C. M. Brislawn, and T. Hopper. The FBI wavelet/scalar quantization standard for gray-scale fingerprint image compression. In *SPIE Proceedings, Visual Information Processing II*, volume 1961, pages 293–304, Orlando, FL, USA, April 1993.

[4] B.-B. Chai, J. Vass, and X. Zhuang. Significance-linked connected component analysis for wavelet image coding. *IEEE Transactions on Image Processing*, 8(6):774–784, June 1999.

[5] L. Chiariglione-Convenor. MPEG-2: Generic coding of moving pictures and associated audio information. ISO/IEC JTC1/SC29/WG11, July 1996.

[6] S.J. Choi and J.W. Woods. Three-dimensional subband/wavelet coding of video with motion compensation. In J. Biemond and E.J. Delp, editors, *Visual Communications and Image Processing '97*, volume 3024 of *SPIE Proceedings*, pages 96–104, San Jose, February 1997.

[7] Charilaos Christopoulos, Athanassios N. Skodras, and Touradj Ebrahimi. The JPEG2000 still image coding system: an overwiew. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, November 2000.

[8] R. R. Coifman, Y. Meyer, S. Quake, and Mladen Victor Wickerhauser. Signal processing and compression with wavelet packets. In Y. Meyer and S. Rocques, editors, *Proceedings of the International Conference Wavelets and Applications*, pages 77–93, Toulouse, France, 1993. Editions Frontieres.

[9] R.R. Coifman and M.V. Wickerhauser. Entropy based methods for best basis selection. *IEEE Transactions on Information Theory*, 38(2):719–746, 1992.

[10] M.L. Comer, K. Shen, and E.J. Delp. Rate-scalable video coding using a zerotree wavelet approach. In *Proceedings of the 9th Workshop on Image and Multidimensional Signal Processing*, pages 162–163, Belize City, 1996.

[11] C.D. Creusere. Image coding using parallel implementations of the embedded zerotree wavelet algorithm. In B. Vasudev, S. Frans, and S. Panchanathan, editors, *Digital Video Compression: Algorithms and Technologies 1996*, volume 2668 of *SPIE Proceedings*, pages 82–92, 1996.

[12] M. Feil, R. Kutil, R. Norcen, and A. Uhl. ParWave: Parallel wavelet video coding. In E. Krause and W. Jäger, editors, *High Performance Computing in Science and Engineering 2001*, pages 512–526, Stuttgart, Germany, 2002. Springer-Verlag.

[13] J. Fridman and E.S. Manolakos. On the scalability of 2D discrete wavelet transform algorithms. *Multidimensional Systems and Signal Processing*, 8(1–2):185–217, 1997.

[14] K.H. Goh, J.J. Soraghan, and T.S. Durrani. New 3-D wavelet transform coding algorithm for image sequences. *Electronics Letters*, 29(4):401–402, 1993.

[15] W.L. Hsu and H. Derin. Video compression using adaptive wavelet packets and DPCM. In H.H. Li, S. Sun, and H. Derin, editors, *Video Data Compression for Multimedia Computing*, pages 55–94. Kluwer Academic Publishers Group, 1997.

[16] ISO/IEC JPEG committee. JPEG 2000 image coding system — ISO/IEC 15444-1:2000, December 2000.

[17] B.J. Kim and W.A. Pearlman. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings Data Compression Conference (DCC'97)*, pages 251–259. IEEE Computer Society Press, March 1997.

[18] S.H. Kim and N.C. Kim. Image coding using wavelet-based fractal approximation. In *Proceedings of the International Picture Coding Symposium (PCS'97)*, volume 143 of *ITG-Fachberichte*, pages 451–456. VDE-Verlag, Berlin, Offenbach, September 1997.

[19] Nick G. Kingsbury. Complex wavelets for shift invariant analysis and filtering of signals. *Applied and Computational Harmonic Analysis*, 10(3):234–253, May 2001.

[20] C. Koc, G. Chen, and C. Chui. Complexity analysis of wavelet signal decomposition and reconstruction. *IEEE Trans. on Aereospace and Electronic Systems*, 30(3):910–918, July 1994.

[21] R. Koenen. Overview of the MPEG-4 standard. ISO/IEC JTC1/SC29/WG11, July 1996.

[22] D. Krishnaswamy and M. Orchard. Parallel algorithm for the two-dimensional discrete wavelet transform. In *Proceedings of the 1994 International Conference on Parallel Processing*, pages III:47–54, 1994.

[23] R. Kutil. High dimensional wavelet transforms on MIMD architectures. Master's thesis, Dept. of Computer Science, Salzburg University, Austria, February 2000.

[24] R. Kutil. Parallelisable zero-tree image coding with significance maps. In R. Sakellariou et al., editors, *Parallel Processing. Proceedings of EuroPar'01*, volume 2150 of *Lecture Notes on Computer Science*, pages 674–677. Springer-Verlag, 2001.

[25] R. Kutil. Zerotree based video coding on MIMD architectures. In S. Panchanathan, V. Bove, and S.I. Sudharsanan, editors, *Media Processors 2001*, volume 4313 of *SPIE Proceedings*, pages 61–68, January 2001.

[26] R. Kutil. Approaches to zerotree image and video coding on MIMD architectures. *Parallel Computing*, 2002. accepted.

[27] R. Kutil. The graph structure of the anisotropic wavelet packet transform. In *Proceedings of the 7th international scientific conference devoted to the 25th anniversary of civil engineering faculty and 50th anniversary of technical university Kosice*, pages 41–47, May 2002.

[28] R. Kutil. A significance map based adaptive wavelet zerotree codec (SMAWZ). In S. Panchanathan, V. Bove, and S.I. Sudharsanan, editors, *Media Processors 2002*, volume 4674 of *SPIE Proceedings*, pages 61–71, January 2002.

[29] R. Kutil and A. Uhl. Hardware and software aspects for 3-D wavelet decomposition on shared memory MIMD computers. In Peter Zinterhof, M. Vajtersic, and Andreas Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 347–356. Springer-Verlag, 1999.

[30] R. Kutil and A. Uhl. Optimization of 3-d wavelet decomposition on multiprocessors. *Journal of Computing and Information Technology (Special Issue on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia)*, 8(1):31–40, 2000.

[31] R. Kutil and A. Uhl. Parallel adaptive 3-d wavelet analysis for fast and efficient video coding. In E.H. D'Hollander, G.R. Joubert, F.J. Peters, and H.J. Sips, editors, *Parallel Computing: Fundamentals and Applications.*

Proceedings of the International Conference ParCo 1999, pages 316–323. Imperial College Press, 2000.

[32] R. Kutil and A. Uhl. Parallel adaptive wavelet analysis (Invited Paper). *Future Generation Computer Systems (Special Issue on High Performance Numerical Methods and Applications)*, 1(18):97–106, 2001.

[33] G. Lafruit and J. Cornelius. Parallelization of the 2D fast wavelet transform with a space-filling curve image scan. In A.G. Tescher, editor, *Applications of Digital Image Processing XVIII*, volume 2564 of *SPIE Proceedings*, pages 470–482, 1995.

[34] R.L. Lagendijk, F. Bosveld, and J. Biemond. Subband video coding. In A.N. Akansu and M.J.T. Smith, editors, *Subband and Wavelet Transforms*, pages 251–286. Kluwer Academic Publishers Group, 1996.

[35] A.S. Lewis and G. Knowles. Video compression using 3D wavelet transforms. *Electronics Letters*, 26(6):396–398, 1990.

[36] W.-K. Lin and N. Burgess. Listless zerotree coding for color images. In *32nd Asilomar Conference on Signals, System and Computers*, CA, USA, November 1998.

[37] W.-K. Lin and N. Burgess. 3d listless zerotree coding for low bit rate video. In *IEEE International Conference on Image Processing*, Kobe, Japan, October 1999.

[38] Julian F. A. Magarey. *Motion Estimation using Complex Wavelets*. PhD thesis, University of Cambridge, UK, February 1997.

[39] Julian F. A. Magarey and Nick G. Kingsbury. Motion estimation using a complex-valued wavelet transform. *IEEE Transactions on Signal Processing, Special issue on wavelets and filter banks*, 46(4):1069–1084, April 1998.

[40] F.G. Meyer, A.Z. Averbuch, and R.R. Coifman. Motion compensation of wavelet coefficients for very low bit rate coding. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'97)*, Santa Barbara, October 1997.

[41] Francois G. Meyer, Amir Z. Averbuch, and Jan-Olov Strömberg. Fast wavelet packet image compression. In J.A. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC'98)*, page 563, Snowbird, UT, USA, March 1998. IEEE Press.

[42] Francois G. Meyer, Amir Z. Averbuch, and Jan-Olov Strömberg. Fast adaptive wavelet packet image compression. *IEEE Trans. on Image Process.*, 9(5):792–800, May 2000.

[43] J.-R. Ohm. Three-dimensional subband coding with motion compensation. *IEEE Transactions on Image Processing*, 3(5):559–571, September 1994.

[44] M.T. Orchard and G.J. Sullivan. Overlapped block motion compensation: A estimation-theoretic approach. *IEEE Transactions on Image Processing*, 5:693–699, September 1994.

[45] J.N. Patel, A.A. Khokhar, and L.H. Jamieson. Scalability of 2-D wavelet transform algorithms: analytical and experimental results on coarse-grain parallel computers. In *Proceedings of the 1996 IEEE Workshop on VLSI Signal Processing*, pages 376–385, San Francisco, CA, USA, October 1996.

[46] N. M. Rajpoot, Francois G. Meyer, R. G. Wilson, and R. R. Coifman. On zerotree quantization for embedded wavelet packet image coding. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'99)*, Kobe, Japan, October 1999.

[47] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Trans. on Image Process.*, 2(2):160–175, 1993.

[48] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–249, June 1996.

[49] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. on Signal Process.*, 41(12):3445–3462, December 1993.

[50] M.J. Shensa. Wedding the à trous and Mallat algorithms. *IEEE Trans. on Signal Process.*, 40(10):2464–2482, 1992.

[51] E.P. Simoncelli, W.T. Freeman, W.H. Adelson, and D.J. Heeger. Shiftable multi-scale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607, March 1992.

[52] Kamakshi Sivaramakrishnan and Truong Nguyen. A uniform transform domain video codec based on dual tree complex wavelet transform. In *Proceedings of the 2001 International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, Salt Lake City, UT, USA, May 2001.

[53] J.K. Su and R.M. Mersereau. Non-iterative rate-constrained motion estimation for OBMC. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'97)*, pages II:33–xx, Santa Barbara, CA, USA, October 1997.

[54] J.K. Su and R.M. Mersereau. Motion estimation methods for overlapped block motion compensation. *IEEE Transactions on Image Processing*, 9(9):1509–1521, September 2000.

[55] S. Sullivan. Vector and parallel implementations of the wavelet transform. Technical report, Center for Supercomputing Research and Development, University of Illinois, Urbana, 1991.

[56] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158 – 1170, 2000.

[57] D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 5(3):572–588, September 1993.

[58] L. Thornton, J. Soraghan, R. Kutil, and M. Chakraborty. Unequally protected SPIHT video codec for low bit rate transmission over highly error-prone mobile channels. *Signal Processing: Image Communication*, 17(4):327–335, April 2002.

[59] J. Vass, B.-B. Chai, K. Palaniappan, and X. Zhuang. Significance-linked connected component analysis for very low bit rate wavelet video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(4):630–647, June 1999.

[60] J. Vass and X. Zhuang. Efficient three-dimensional wavelet codecs for networked video communications. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'99)*, Kobe, Japan, October 1999.

[61] P. Waldemar, M. Rauth, and T. A. Ramstad. Video compression by three-dimensional motion-compensated subband coding. In *Proceedings of the Norwegian Signal Processing Symposium*, pages 227–232, Stavanger, Norway, September 1995.

[62] G.K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[63] A. Wang, Z. Xiong, P. A. Chou, and S. Mehrotra. Three-dimensional wavelet coding of video with global motion compensation. In *Proceedings Data Compression Conference (DCC'99)*, pages 404–413, Snowbird, UT, March 1999.

[64] Frederick W. Wheeler and William A. Pearlman. SPIHT image compression without lists. In *Proceedings of the IEEE ICASSP 2000*, Istanbul, Turkey, June 2000.

[65] F.W. Wheeler and W.A. Pearlman. Low-memory packetized SPIHT image compression. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, October 1999.

[66] M.V. Wickerhauser. INRIA lectures on wavelet packet algorithms. Lecture notes, INRIA, 1991.

[67] M-L. Woo. Parallel discrete wavelet transform on the Paragon MIMD machine. In R.S. Schreiber et al., editor, *Proceedings of the seventh SIAM conference on parallel processing for scientific computing*, pages 3–8, 1995.

[68] Zixiang Xiong, Kannan Ramchandran, and Michael T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Transactions on Image Processing*, 6:677–693, May 1997.

[69] Zixiang Xiong, Kannan Ramchandran, and Michael T. Orchard. Wavelet packet image coding using space-frequency quantization. *IEEE Transactions on Image Processing*, 7(6):892–898, June 1998.

[70] J. Xu, Z. Xiong, S. Li, and Y.-Q. Zhang. Three-dimensional embedded subband coding with optimized truncation (3-D ESCOT). *Applied and Computational Harmonic Analysis*, 10(3), May 2001.

[71] X. Yang and K. Ramchandran. Hierarchical backward motion compensation for wavelet video coding using optimized interpolation filters. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'97)*, Santa Barbara, CA, USA, October 1997.

Curriculum Vitae

Name: Rade Kutil

Date of birth: June 17, 1971

Place of birth: Salzburg, Austria

Education and Employment:

- **1977–1981** Elementary school in Bischofshofen
- **1981–1985** General secondary school in St. Johann
- **1985–1990** Secondary school for electronics and communications engineering
- **1991–1998** Computer engineer at Eisenwerk Sulzau-Werfen
- **1994–2000** Studying informatics at Salzburg University (master's degree)
- **2001–2002** Lecturer at Salzburg University
- **2000–2002** PhD student at Salzburg University