Context weighting in wavelet image compression

Brünken, L.J.A.

Award date:
1999

Eindhoven University of Technology
Department of Electrical Engineering
Telecommunication Technology and Electromagnetics

Context Weighting in Wavelet Image Compression

by L.J.A. Brünken

Graduation report
July 1998-June 1999
coach: dr. ir. Tj.J. Tjalkens
supervisor: Prof. dr. ir. G. Brussaard

Acknowledgements

In the beginning of July 1998, I asked for a master thesis project within the Information- and Communication Theory. Tjalling Tjalkens and Frans Willems provided me one within 1 week. I would like to thank both of them for their assistance and help to do my work here. I also would like to thank Paul Volf for his help, remarks and chocolate eggs during the last few months of my project. Also I would like to thank the rest of the group for providing a nice working environment.
Further I would like to thank my friends from IT92 for keeping up with me the last 7 years.

– Leon Brünken, June 5^{th} 1999

Contents

Chapter 1

Introduction

For well known reasons, image compression (the technique to describe an image with fewer bytes) is an important research area within signal processing and information theory. The goal of this research is to find methods/algorithms that describe an image with a high compression ratio and a good quality. Well known formats of images are GIF, JPEG and BMP. They can compress images for various qualities and features.

In every image coding method, an entropy coder is used. In an entropy coder, one wants to estimate the probability of a coefficient as good as possible. Then it can be compressed to its optimum (entropy). When one wants to estimate the probability, one generally uses the neighbouring coefficients as a indication for the value of the coefficient which has to be transmitted. This used neighbourhood is then called the *context*. Using only the neighbouring coefficients as the context is not necessarily the optimum. Also, the optimum context can change while processing the image.
The CTW algorithm [21] uses a weighting technique on its context to compress data coming from stationary sources. This weighting technique works very good for for instance text compression.

The goal of this project was to investigate whether the use of the context weighting techniques are useful to estimate the probability of a coefficient in an image.

The first part of this report is on image coding. The image coder known as "EZW" [14] was used for compressing images. The second part is about CTW and a derived form of it (MDL-algorithm). Finally, the last part is about the CTW and its application to images.

Chapter 2

Image Coding

In this chapter a general description of image coding is given. The subject will treated from an information theorist point of view. This chapter is provided to give an overview of the different techniques in image coding.

Image coding consists basically of three steps: the Image Transform, the Information Reduction, and the Information Compaction. See figure 2.1.

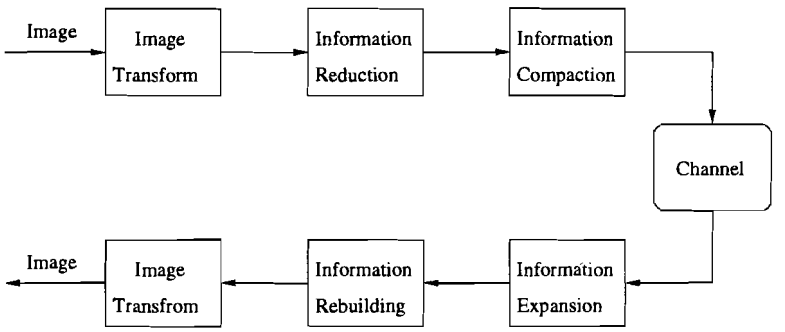

Figure 2.1: Image coding

A widely used coding scheme as JPEG has as image transform a DCT, as information reduction a quantiser with DPCM (Differential Pulse Code Modulation) and as information compaction a Huffman encoding and a QM-coder (see [12]).

The next three sections of this chapter will discuss the blocks in figure 2.1 in more detail.

2.1 Transform Coder

A transform coder is often used because it changes the characteristics of the data stream so that the stream can be better processed. For instance the Karhunen-Loève transform decorrelates random variables which are correlated. When the Gaussian variables are decorrelated, it is easier to build an optimal quantisiser.
In image coding however, the decorrelating of variables (or coefficients) does not work prefect. The side effect that a transform coder can localise the energy of the image into fewer coeffi-

cients, is where it is used for in image coding. One transform code used with image coding is the wavelet transform. It was already used in geological applications and numerous other, but since about 1992, when mathematicians build a solid mathematical basis for wavelets, it has been used for image coding also. Another transform coder is the DCT (Discrete Cosine Transform). The wavelet transform has been used in the research for this project. See chapter 3 for more on the wavelet transform and its advantages.

2.2 Information Reduction

In this block "information Reduction" of figure 2.1 knowledge and understanding of image coding is used. We distinguish two different goals for this block. The first is the quantisation to compress the image. If the transform code doesn't introduce errors in the image, the quantisation is the only parameter to decrease the quality (see below how to measure the quality). It usually increases the compression enormously. This quantisation is closely related to the rate-distortion theory. For a higher quantisation/distortion, less data has to be transmitted to the decoder.
The second goal has little to do with image compression; this block also takes care of all image features like "embedded image coding", sequential update, transparent colours etc.

This part is usually considered to be the most distinctive part of an image coder. For example the EZW (Embedded Zerotree Wavelet encoder) specifies only this block.
To compare the results of different coders with each other, the PSNR (Power Signal to Noise Ratio) is normally used as a measure for the quality. The formula for PSNR is:

$$PSNR = 10 \log_{10}(\frac{255^2}{MSE})$$

with MSE the Mean Squared Error, $\mathcal{E}((x-y)^2)$ (with x the original and y the copy (or vice versa)).
Many people argue that this is not a good measure for the quality. Artifacts like ringing, blocking etc, are not taken into account. So far however, there is no better simple objective measure for the quality of a picture. So the PSNR is most used (as it is here) for comparing different coders.

2.3 Information Compaction

After all image-specific coding, an information compaction coder can be used. The goal is purely to compact the information lossless.
Usually this is a sort of arithmetic coder which can be modelled as in figure 2.2. AE stands for Arithmetic Encoder, AD for the Decoder. The two smaller blocks are the Estimators. The estimators estimate the probability for the next symbol to be encoded, given the past (or another context).
With this conditional probability, the arithmetic encoder can efficiently encode the symbols. Question remains to built a good estimator and/or use a good context. The CTW algorithm is actually "just" an estimator which keeps the costs for the conditional probabilities low. See chapter 5 for more information about the CTW.

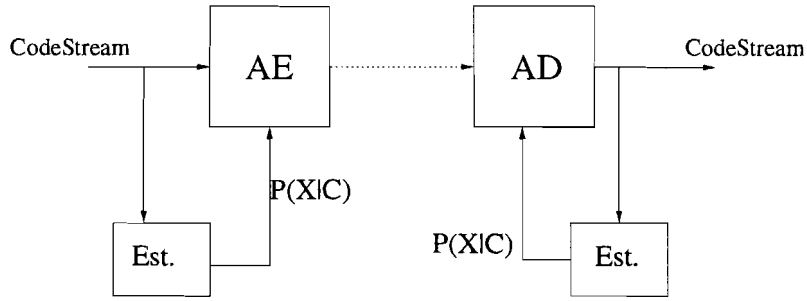

Figure 2.2: Arithmetic coding

In the JPEG image coding scheme, the information compaction is done by a QM- and a Huffman-coder. These coders also need estimated probabilities.
Because usually the conditional entropy $H(X|Y)$ is smaller than the entropy $H(X)$, the compaction is better when the coder can use good estimated conditional probabilities.

Chapter 3

Wavelets

In the last few years, a many image coding techniques use wavelets. This is not a coincidence. Wavelets are becoming more important and better understood. And not only for image techniques. They're in use in a lot of areas, for example in geological research, speech recognition and physics. Wavelets seem to recognise interesting features of data, while not introducing obvious artifacts of their own.
In general, there a two ways to explain the working of wavelets. One is the mathematical way, and the other is the electrical engineering way. Section 3.1 is an electrical engineering approach. A more mathematical approach can be found in [5].
Because we use wavelets with images, section 3.2 will go into that. Section 3.3 will give some practical real-life wavelet examples.

This chapter won't be a complete guide into the world of wavelets. Or even only image wavelets. For the master thesis project, some wavelets were implemented. With the information given here, one could implement a wavelet, not design or analyse one.

3.1 View from the electrical engineering side

Wavelets can be considered to be similar to bandpass filters. In a usual situation there are two filters; one low-pass filter (normally called $H(f)$) and one high pass filter ($G(f)$). This combination is also called a two-channel filter bank. When these filters are applied on some form of data (i.e. images, speech), two transformed streams of data are formed. These streams can be criticly sub-sampled and then reconstructed, using the inverse filters and up sampling, to the original data. See also figure 3.1. Here $E(f)$ and $F(f)$ are the inverse filters of $G(f)$ and $H(f)$ respectively.
A typical example of a set of bandpass filter is depicted in figure 3.2.
That this filtering, subsampling and up sampling can be done without making an error, has been known for a long time. See for example [15]. One cannot use every pair of filters of course. These filters have some restrictions. The restrictions (like orthogonality, gain etc) are not very interesting for this report. The wavelets used, have these special properties in various qualities. Of course, the quality is a subject of this report. It will be dealt with later. Note that it is not specified if the filters are digital or analog. How the digital wavelet is implemented for the project is subject of the next subsection.

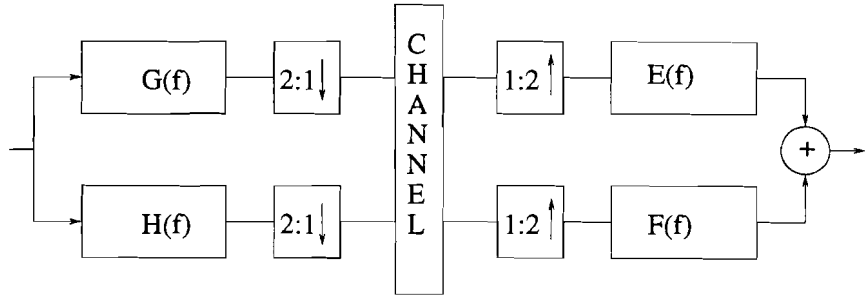

Figure 3.1: Using Low and High pass filters

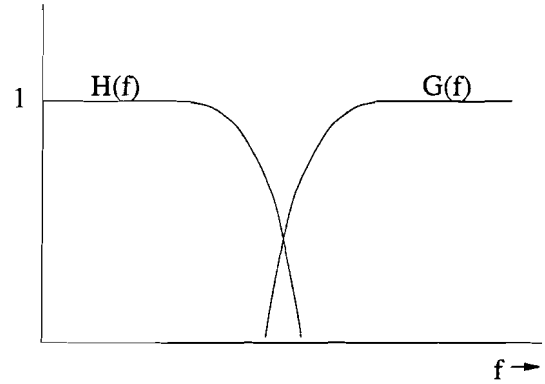

Figure 3.2: Low and High pass filters

3.1.1 Towards digital image wavelets

One property of the filters/wavelets used is the (bandpass) gain. If the gain of the filters is 1, figure 3.1 is correct. Often, the gain of a filter is $\frac{1}{\sqrt{2}}$. So for each filter, a multiplication by $\sqrt{2}$ is needed. This can be done by a multiplier at the filter itself or one multiplier at the end of a bandpass filter-combination.

Because images are in the digital domain, the filters are also digital. So for $H(f)$ we write $H[z]$ and $G(f)$ is $G[z]$. With the wavelets normally used in image coding, the inverse filters are transformations of the original filters. The inverse filter of the high-pass filter is the low-pass filter multiplied by $(-1)^n$ (n being the n^{th} coefficient in the filter). This is indicated by $-G[-z]$. The inverse filter of of the low-pass filter is $H[-z]$.
Incorporating all wavelet-specific conditions into figure 3.1 we get figure 3.3.
Wavelets are used because they localise the energy of the coefficients into fewer coefficients. For example with (natural) images, high frequencies don't occur very often. The data coming from the high-pass filter will not have much energy. The data from the low pass filter will have a high level of energy. (In drawings or photos of specific structures high frequencies can occur much more often, but it is still less often than low-frequencies.) Concluding, most energy will be in half the number of data.

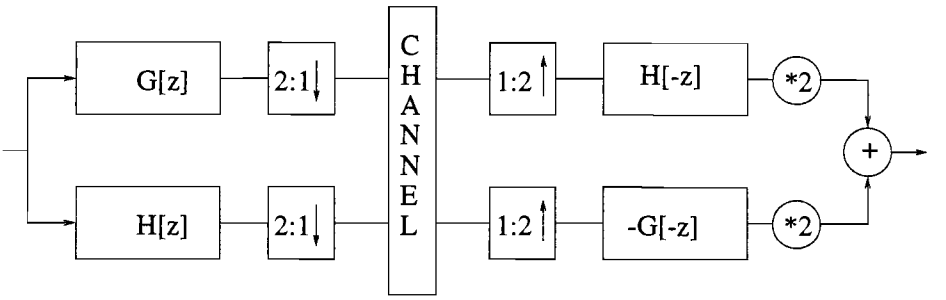

Figure 3.3: Using Low and High pass filters

Example 3.1 *This wavelet is called the Haar-wavelet. It is the mother of all wavelets. The two filters are (c[z] is incoming data, x[z] and y[z] are the two filtered coefficients):*

$$\textit{Low-Pass: } x[z] = \tfrac{1}{2}(c[2z] + c[2z+1])$$
$$\textit{High-Pass: } y[z] = \tfrac{1}{2}(c[2z] - c[2z+1])$$

Notice that the filters already subsample themselves. The inverse filters are somewhat different than depicted in figure 3.3. The original data can be found using: $c[2z] = x[z] + y[z]$ *and* $c[2z+1] = x[z] - y[z]$.
The low-pass filter is nothing more than a moving average. The high-pass is a moving difference filter.

Now, suppose we have the following coefficients[1] *(c[z]):*

`162 162 162 161 162 157 163 161 166 162 162 160 155 163 160 155 157 156`

These will be transformed to (x[z] and y[z]):
`162 161.5 159.5 162 164 161 159 157.5 156.5` *and* `0 0.5 2.5 1 2 1 -4 2.5 0.5`
Clearly, x[z] has much more energy than y[z]. When y[z] is not available, it is still possible to reconstruct c[z] by using only x[z]. A compression factor of 2 (half the data), MSE=4.1 and PSNR=42.0 is then reached for the whole Lena image.

Wavelets are often represented with their time-coefficients only. It is sufficient to state $(\frac{1}{2}, \frac{1}{2})$ are the coefficients for the low-pass filter instead of stating $x[z] = \frac{1}{2}(c[2z] + c[2z+1])$ or the frequency function. This is also used in the rest of this report.

3.1.2 Lifting

It is possible to implement a wavelet transform in a filter-bank as depicted in figure 3.4.
In this figure, a square with a "z" in it means a delay of 1 and the Greek symbols indicate the wavelet coefficients. The resulting transformed coefficients still need to be sub-sampled, so the picture isn't complete. Figure 3.4 represents then a imaginary 3/5 wavelet.
The use of the filters can also be implemented otherwise. The data-coefficients are not simply multiplied with the wavelet-coefficients, but the intermediate results are mixed with each other. See figure 3.5.

[1] The coefficients are the first 18 coefficients of the first row of the Lena 512x512 256-gray-scale image

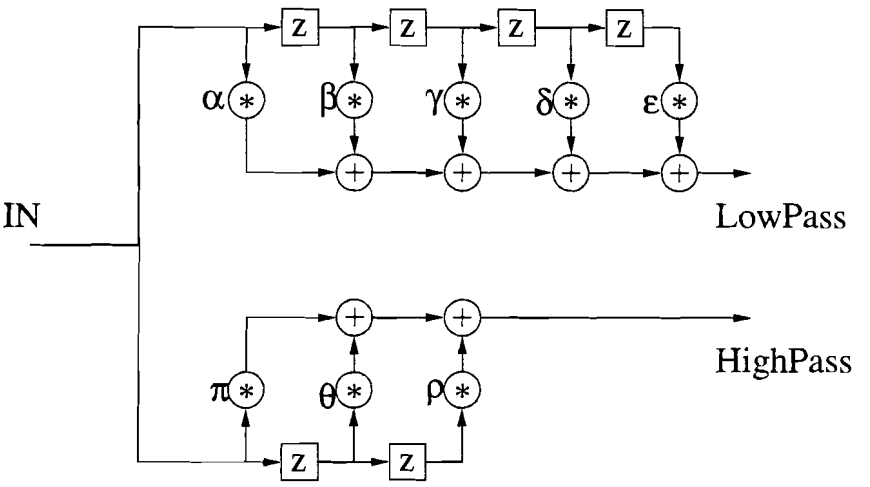

Figure 3.4: Wavelet filter-bank

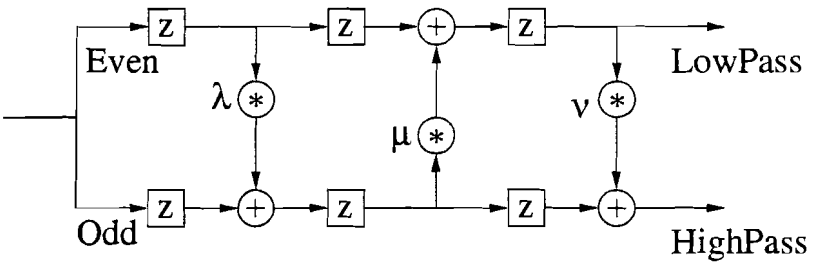

Figure 3.5: Wavelet lifting

In figure 3.5 the data stream is split up into the even and the odd coefficients. These are then multiplied by factors (λ, μ and ν) and then added to the other stream. This is of course not limited to three adders. The resulting two streams still have to be sub-sampled. The technique of transforming a simple wavelet to the scheme in figure 3.5 is called lifting [16]. The TT-transform (subsection 3.3.3) is defined like this. It can be implemented as in figure 3.4. The lifting structure can be easily implemented in hardware. Multipliers, adders and delays are standard building blocks. However the implementation of "ceilers" ($\lceil x \rceil$) is a bit more difficult. The method of lifting is not further discussed here, because it was not used except for implementing the TT-transform. For more information see [16].

3.2 Wavelets with images

In this section some specific problems/advantages/types of wavelets in combination with images are discussed. First a comparison with the DCT is given. After that, it is explained how a 1D filter will transform a 2D image, followed by a transcription of a problem when using a continuous filter with a finite length input stream.

3.2.1 DCT

In the widely used JPEG (Join Picture Experts Group) image format, the DCT (Discrete Cosine Transform) is used as the transform coder. This transform code has some drawbacks compared to wavelets. One of the drawbacks is that with a high compression (i.e. high quantisation), JPEG shows block artifacts. See for example figure 3.6.

Figure 3.6: JPEG Lena image (DCT block artifacts)

This is due to the fact that the DCT takes blocks of 8 by 8 coefficients and transform this block without regard to it's neighbouring blocks (larger blocks are possible but they blocking artifacts remain). If one quantisises these blocks, distinct differences occur between these blocks. With wavelets, this is not a problem. Because the wavelet is like a sliding window over the data, it does not have any borders. These blocking artifacts are important because the eye is sensitive to these kind of artifacts.
Because the transform coder is only a small part of an image coder, it is hard to compare wavelets with DCT by means of quality/compression graphs.

3.2.2 2D wavelets

The wavelet explanation presented so far, works in 1 direction. Images however are 2D. In order to transform an image, it is possible to use 2D wavelets or to use a 1D wavelet in both directions (introduced in [10]). Although it would seem that a 2D wavelet will cope better with a 2D image, the quality of 2D wavelets isn't as high yet as 1D wavelets. Also, the 1D wavelets emphasise on the horizontal and vertical directions in an image. This is a plus, because the human eye is more sensitive in these directions then in other directions ([11]).

So, we use 1D wavelets in 2 directions. See figure 3.7 and its resulting images on Lena[2] in figure 3.8.

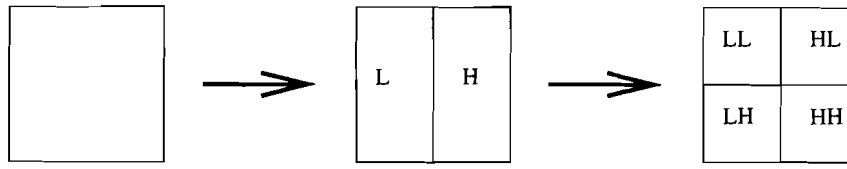

Figure 3.7: 1D wavelet on a 2D image

In figure 3.7 three images are depicted. The left most represents the original image. After a wavelet transform on the rows the low-pass coefficients and the high-pass coefficients are grouped together in an image with the same size. So each row is passed through the filters separately and the resulting output is the middle image of figure 3.7. The low-pass sector (or sub-band) is indicated with a "L" and the high-pass with an "H". On this middle image, the filters are re-applied but now on the columns of the image. The right most image in figure 3.7 is the resulting output. Now there are four sectors. "LL" is the sector with coefficients resulting from a horizontal and vertical low-pass filter. "LH" is the sector (or sub-band) with the coefficients resulting from a horizontal low-pass and a vertical high-pass filter etc.
In figure 3.8 Lena is transformed by the mechanism above. From left to right we have the original Lena (original 512x512, shown as 128x128), then Lena after a horizontal (row-wise) transform and the third is the output of a vertical transformation of the second image. The brightness of the second and third image are a little bit enhanced, so that not all high-pass-coefficients show up black. There is very little energy in the high-pass sub-bands.

Figure 3.8: 1D wavelet on Lena (enhanced)

It is possible to localise the energy in an image even more. This is simply done by re-applying above mechanism. See figure 3.9. The upper row is the same as figure 3.7. After the horizontal and vertical transform, the second row depicts another horizontal and vertical transform but only on the "LL" sub-band. Now most energy is localised in $\frac{1}{16}^{th}$ of the original number of coefficients. The resulting image is called a second-order transformed image, because the full

[2]Lena (or Lenna as Playboy incorrectly stated) is a widely used test image for image coding

wavelet transform has been applied twice. The quarter in the image which has been processed twice is also called the coarser band, compared to the other quarters.
This re-applying of the wavelet transform can go on unpunished. But only if a integer-to-integer lossless wavelet transform is used (see section 3.3). If a non-lossless (or lossy) wavelet transform is used, errors made in a high order sub-band can propagate and become a big problem when back-transforming the image. So there is a trade-off between the order of a transform and the best achievable quality (without quantisation) of the reconstructed image. Furthermore, with high orders, relatively more energy will "leak" through the high pass filter.

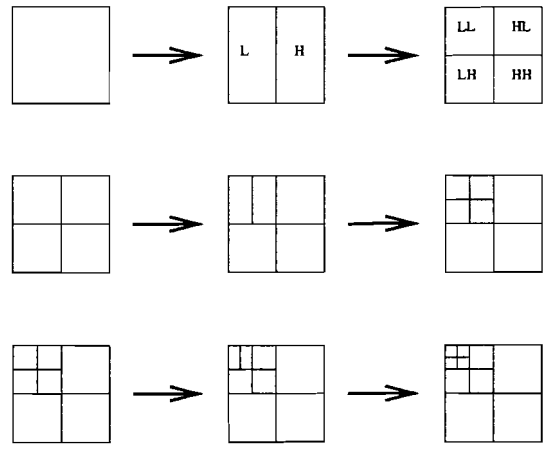

Figure 3.9: Higher order transform

3.2.3 Mirroring

When applying a 5-tap wavelet on a row of coefficients (for example 11 coefficients), it produces $\lceil \frac{11-5+1}{2} \rceil = 4$ transformed coefficients. To be able to reconstruct the original 11 coefficients, one wants 6 coefficients. This is because the first wavelet transform will be with the first 5 coefficients (not the first two). See figure 3.10. An other way to state this is to say that there is a delay between the beginning of the input and the beginning of the output, while the wavelet transform produces no more output when the input stops.

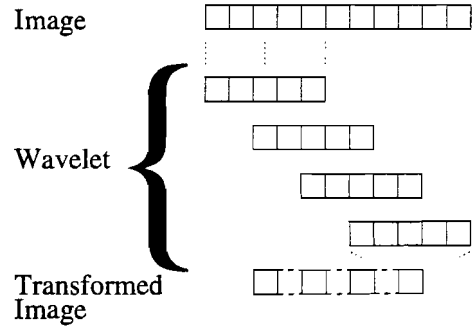

Figure 3.10: Start of wavelet transforming

In general, filtering a finite length data stream, will cause problems at the boundaries of the stream. There are four main methods of signal extension which solves these problems. See also [3]. The four solutions are:

1. zero padding
2. repeating of the boundary value
3. making the signal periodic (circular convolution)
4. mirroring the values of the signal with respect to the boundary

Because wavelet transforming should not introduce errors into the image, perfect reconstruction is a necessary demand. The criterion of whether a boundary extension allows perfect reconstruction can be stated as follows.
Suppose, the signal/stream of length N is extended to infinity at both boundaries, filtered, and down-sampled. If all the coefficients of both infinite length sub-bands can be determined from a subset of N/2 samples in each band then the extension enables perfect reconstruction.
For compression purposes, only N/2 coefficients are allowed to appear in each infinite length sub-band. With these constraints signal extension is only possible with the last two methods. With the first two methods more than N/2 coefficients are needed to reconstruct the N original coefficients [3].

With periodic extension the end of the incoming stream is extended with the beginning of the stream. In image coding, the beginning and the end of a stream (usually a row or a column) have in general little in common with each-other. When applying a wavelet on boundaries which have a sudden change in values, the transformed coefficients will have a sudden change too. This works negative for the compression. So, method 3 does not seem to for image coding. Method 4 is only suited for symmetric filters. Fortunately, most filters used in image coding are symmetric filters.
In figure 3.11 it is shown how the mirroring can be implemented. It is basicly the same figure as figure 3.1 but now rotated over -90 degrees. The four large blocks contain a pictorial description of the filtering. The left filter-combination is the low-pass filter. As is shown in figure 3.11, the input stream is mirrored on both sides, which gives smooth edges. Note that the symmetry-line is mostly antisymmetric (the line is on top of a coefficient), except for two edges in the synthesis filters. This is due to the odd-length filter but even-length input.
With this extended input stream, it is now possible to filter it, and reconstruct it with no errors.
Note: figure 3.11 only shows how to mirror for four symmetric odd-length filters. See [3] for mirroring with other filters.

3.3 Actual Wavelets

In this section some specific wavelets will be presented. The coefficients of the wavelets are all in table 3.1 except for the TT-wavelet. The TT-wavelet is given in sub-section 3.3.3. The wavelets are grouped into lossless (sub-section 3.3.1), lossy (sub-section 3.3.2) and other (sub-section 3.3.3).
Although the 9tap filter from Adelson ([1]) has 9 coefficients, only 5 are given in table 3.1. This is because the wavelet is symmetric and the coefficient with the highest value is the

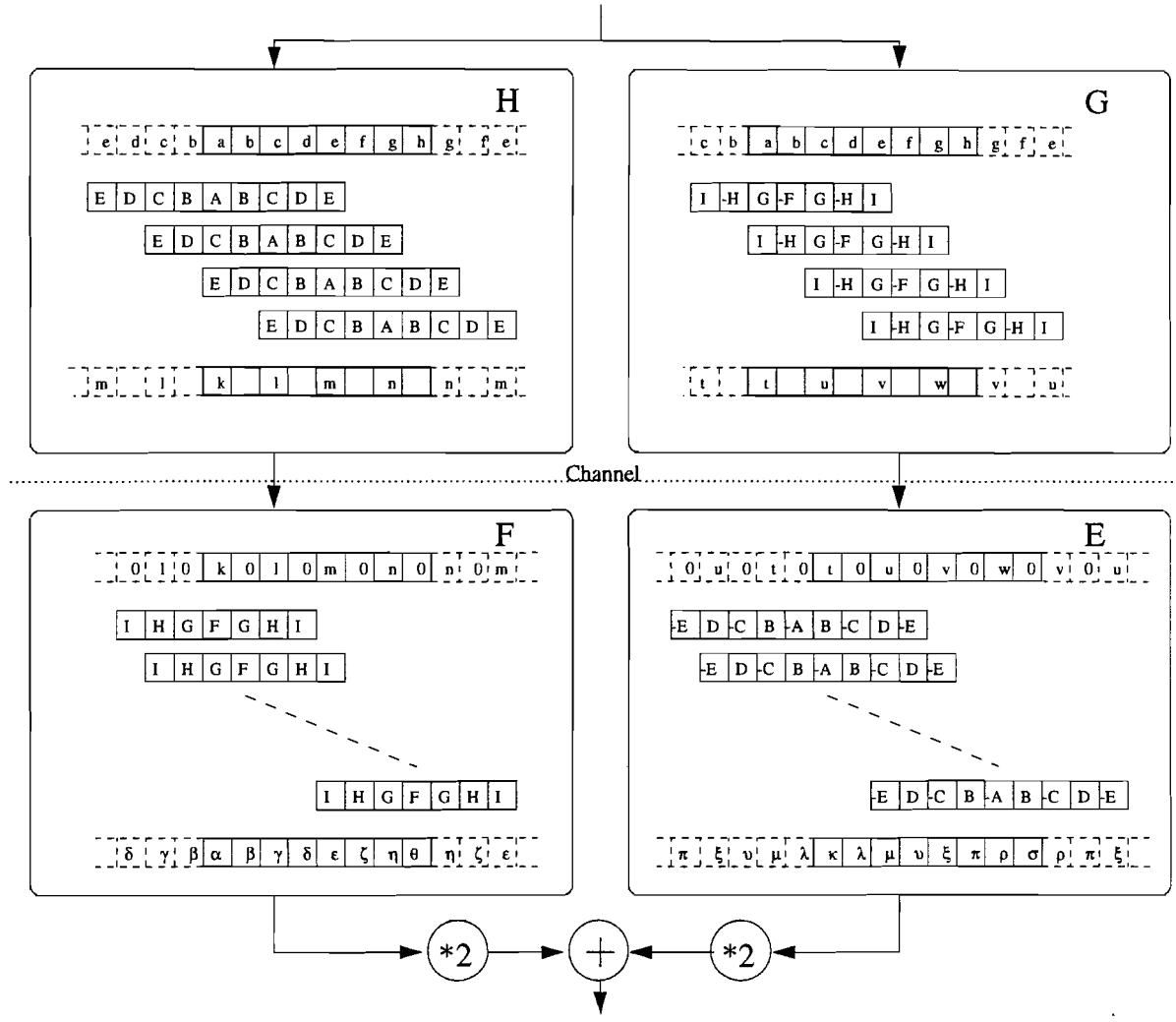

Figure 3.11: Symmetric extension (mirroring)

middle one. The same for the other filters.

Due to the multiplications of the coefficients with the wavelet-coefficients, the range of the image coefficients is extended. They no longer can be represented by 8 bit. So, transforming an image does extend the amount of data one has (but not the image information).

3.3.1 Lossless wavelets

It would seem that applying a lossless wavelet should give the best result. Only with a lossless image is it possible to reconstruct an image perfectly. Because the image coefficients and the transformed coefficients are integers, the wavelet should map integers to integers. Hence the name integer-to-integer wavelets. See also [4]. The transformed coefficients don't need to be integers by definition, but one can usually achieve enough quality with integers. One could use reals, but that raises the question with which precision one wants to transmit the coefficients.

Transform	Low-Pass	High-Pass
Daub 7/9 ([5], [2])	0.602949 0.266864 -0.078223 -0.016864 0.026759	0.557543 0.295636 -0.028772 -0.045636
9tap (from [1])	0.56458 0.29271 -0.05224 -0.04271 0.01995	low pass multiplied by $(-1)^n$

Table 3.1: Wavelet coefficients

The reason that lossless transforms are not the only transforms used, is that some other transforms localise the energy better. The reconstruction is not perfect then, but most of the time the human eye can't tell the difference between the original and the reconstructed image.
To demonstrate how good a wavelet localises the energy, see table 3.2. First, the image Lena is converted to a 3^{rd} order transform of it. This is done with three different wavelet transforms. The Daubechies 9/7, the 9tap from Adelson and the Two-Ten transform. Of this transformed image, the average and the variance ($\text{VAR}^2 = \mathcal{E}(X^2) - \mathcal{E}(X)^2$ with $\mathcal{E}(.)$ the expectation) of the coefficients in the lowest sub-band is calculated. The two calculations are also made for the rest of the coefficients.

Transform	Lowest Band		Other Bands	
	AVG	VAR	AVG	VAR
Daub 9/7	123.94	44.62	1.36	2.51
9tap	123.95	45.75	1.45	2.70
TT-transform	122.56	44.84	4.08	5.93

Table 3.2: Comparisons of wavelets (3^{rd} order . Lena)

The variance of the coefficients which were passed through a high-pass filter once or more (most right column), can be higher then the average because the coefficients can have a negative value. According to table 3.2 the Daubechies 9/7 transform and the 9tap transform have more or less the same energy localisation. The lossless TT-transform however has a significant higher average in the high-pass sub-bands, and a larger variance.
This means that the coefficients in the higher sub-bands probably won't be as good compressed as the same sort of coefficients of the other transforms. Because if the variance increases, the probabilities on a certain coefficient value will decrease which will work negative on the compression ratio.

3.3.2 Lossy wavelets

The best wavelets so far known are lossy. When a lossy wavelet is used to produce a higher order image, more precision will get lost while decoding (a quantisation error will propagate). This can even lead to worse qualities compared with quantised-lower-order pictures. Especially the 9tap wavelet seems to be influenced by this. Note that the quantisation of the reals to integers is only done as the last step. Between the multiple (inverse) transforms there is no quantisation.
For the rest of this report (as in many other articles) the Daubechies 9/7 transform is used.

3.3.3 Other "wavelets"

The TT-transform is specified as a lifting scheme. It looks a bit different than the already specified transforms.
The two analysis filters are:

$$s[n] = \lfloor \frac{x[2n] + x[2n+1]}{2} \rfloor,$$

$$d[n] = x[2n] - x[2n+1] + p[n],$$

with

$$p[n] = \lfloor \frac{3s[n-2] - 22s[n-1] + 22s[n+1] - 3s[n+2] + 32}{64} \rfloor$$

So the TT-transform has two outputs, s[n] and d[n]. The reconstruction is as follows:

$$x[2n] = s[n] + \lfloor \frac{d[n] - p[n] + 1}{2} \rfloor$$

$$x[2n+1] = s[n] - \lfloor \frac{d[n] - p[n]}{2} \rfloor$$

So the reconstruction produces the even and odd symbols. In a lifting scheme it looks as follows (figure 3.12):

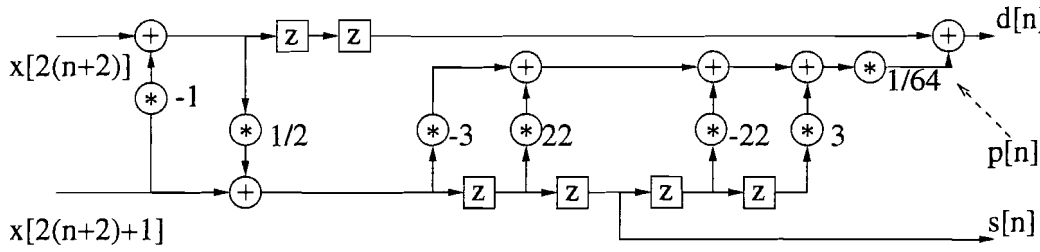

Figure 3.12: Two-Ten transform

All dividers (or multiplication by a factor) round the result to the nearest integer. This incorporates the +1 or +32 and floorers ($\lfloor x \rfloor$) in the TT-formulas. This scheme is not causal, but with mirroring that can be circumvented. For example in [20] the Two-Six transform is used (which is also lossless).

Chapter 4

Embedded Zerotree coding with Wavelets

In 1993 Shapiro presented an article [14] in which he presented a new method for image coding. He used a dependency between pixels which nobody had ever used before. This dependency manifests itself in the so-called Zerotree. The Zerotree indicates the relation between spatial adjacent coefficients across sub-bands in a transformed image. The method is called the Embedded Zerotree Wavelet (EZW) -method. In this context "embedded" means that the resulting output-stream can be stopped at any time without losing vital (structural) information about the image. One loses the precision and the perceived quality. The algorithm itself consists of two main passes. These will be explained in section 4.3, and the Zerotree in section 4.2. First an introduction into successive approximation to explain the embedded coding principle.

4.1 Successive Approximation Quantisation

Suppose one wants to send the number pi (π) to somebody else. It would be logical to start with the number "3", then "1","4","1" etc. Until the receiver has the precision he needs. This system of successive approximation can be applied to image coding as well. Suppose one wants to send an integer in the range of 0 to 255 (2^8 numbers). This can be done by converting the integer to its binary value, and then transmitting the bits. First the MSB so the receiver can decrease the range the integer is in the most (if the MSB is a "0", then the integer is within [0..127]). Next of course the MSB of the remaining bits of the integer. If all 8 bits are send, the receiver knows the exact value. If only seven bits could be send (for whatever reason), the receiver knows the value within a precision of "1". This method of sending the most important bit first, and then refining is called the Successive Approximation method.

The best guess the receiver can make with his so far received bits is the value in the middle of the known range i.e. if the known range is [64..127], the best guess is 95 or 96. This value (96) is also reached if the integer value is quantised with threshold 32. This successive approximation can be used like this as an quantisation method.
If one imagines an image as a stream of bits, "most important" bits at the front, one can use this SAQ as an encode scheme. The only problem is then how to distinct the important bits from the not-so-important bits. The following section will help with that classification.

4.2 Important bits first

In this section it will be explained how the order of bits in the bit-stream is constructed. As seen in the previous section, the most important bit(s) should be in the first part of the bit-stream.

4.2.1 Zerotrees

A Zerotree can only exist in images which have been processed by a wavelet more than once. A tree is defined as the relationship between coefficients in two or more frequency bands as depicted in figure 4.1.

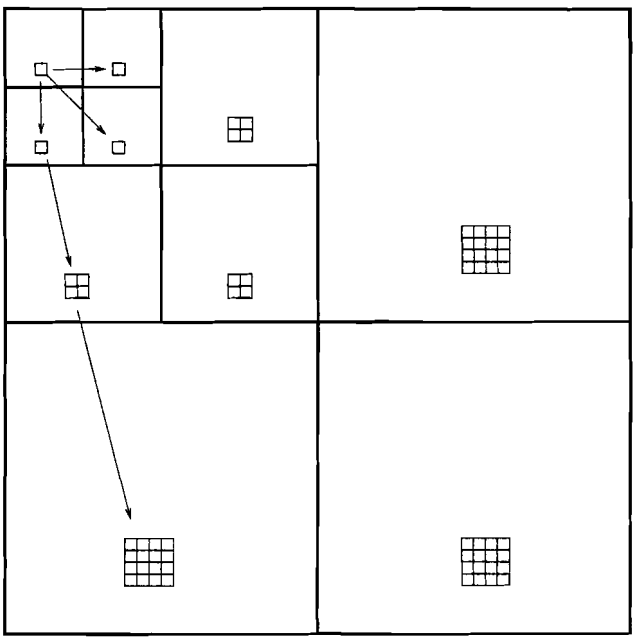

Figure 4.1: Tree in a wavelet-transformed image

Now define for each coefficient in the tree, a parent and three or four children as follows. The parent of a coefficient is the coefficient with the same spatial place in the original image, but in the scale which has been processed one more time. This scale does not have to exist, in which case the coefficient has no parent. So, for a coefficient on coordinates (i,j) in an N by N image which had been transformed k times, the parent is

$$\begin{cases} (\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor) & \text{if i} \in [\frac{N}{2^{k-1}}, N] \text{ and j} \in [\frac{N}{2^{k-1}}, N], \\ (i - \frac{N}{2^k}, j - \frac{N}{2^k}) & \text{if i} \in [\frac{N}{2^k}, \frac{N}{2^{k-1}}] \text{ and j} \in [\frac{N}{2^k}, \frac{N}{2^{k-1}}] \\ (i - \frac{N}{2^k}, j) & \text{if i} \in [\frac{N}{2^k}, \frac{N}{2^{k-1}}] \text{ and j} \in [0, \frac{N}{2^k}] \text{ and} \\ (i, j - \frac{N}{2^k}) & \text{if i} \in [0, \frac{N}{2^k}] \text{ and j} \in [\frac{N}{2^k}, \frac{N}{2^{k-1}}]. \end{cases}$$

For the coefficients in the lowest sub band, there is no parent of course.
The children of a coefficient are the three or four coefficients with the same spatial place, but in a scale which has been processed one time less. For a coefficient with coordinates (i,j) in

the same image as stated above, the three or four children are:

$$\begin{cases} (2i,2j),(2i+1,2j),(2i,2j+1)\,\text{and}\,(2i+1,2j+1) & \text{if i and j} \in [\frac{N}{2^k},\frac{N}{2}], \\ (2i,j),(i,2j)\text{and}(2i,2j) & \text{if i and j} \in [0,\frac{N}{2^k}]. \end{cases}$$

The coefficients in lower sub bands usually have a smaller value. As explained in chapter 3, the values of the coefficients in higher sub bands is usually much lower than those in the lower bands. This gives rise to the following "hypothesis" from Shapiro.

> "If a coefficient at a coarse scale is insignificant with respect to a threshold then all of its descendants are also insignificant."

Although not always true, this "hypothesis" can be used to compress an image very efficiently; if the descendants (all children and its children descendants) are insignificant, it is sufficient to encode the parent as a ZeroTreeRoot (see below), and not specify all the descendants. Depending on the number of descendants, this allows a great increase in the compression factor. The subject of quantisation/thresholding will be handled in subsection 4.2.2.
A ZeroTreeRoot is specified as that coefficient that has all its descendants below the threshold, and its parent or siblings above the threshold.
When encoding an image, and one specifies the coarser scales (or the lower sub bands) first, the Zerotree root can reduce much of the length of the bit stream. In this sense, it is more important the send the lower sub bands scales first.
By using the Zerotree, one uses context information in order to transmit less data (information compaction). The EZW-CTW algorithm will use besides the context used by the Zerotree, also the neighbouring coefficients (spatial context) and the values of the coefficients in the zerotree.

4.2.2 Bit Plane Coding

The combination of SAQ and images is as follows:

1. For all coefficients, determine if it is significant with respect to a certain threshold (i.e. determine if the coefficient value is higher than the threshold). Transmit for all coefficients in the image a "1" if the coefficient is significant and a "0" if not.

2. If the coefficient is above the threshold, decrease the coefficient value with the threshold value.

3. Now, decrease the threshold and repeat the procedure.

Gradually, the receiver learns more and more about the image. If the threshold is a power of 2, this encoding scheme is the same as just sending the bit planes of the image. See figure 4.2. In the left upper corner of figure 4.2, a 4 by 4 image is given, which has three coefficients other than zero. If one now wants to send this image with the bit plane encoding, one needs three bit planes. These are depicted on the right in figure 4.2. The first bit plane consists of 1 one, and 15 zeros. This bit plane is transmitted first. After receiving this bit plane, the receiver knows that there is 1 coefficient which has a value between 4 and 7, and 15 coefficients which have a value between 0 and 3. After transmitting the second bit plane, the receiver now knows: 1 coefficient is 4 or 5, 1 coefficient is 2 or 3, and the rest is 0 or 1. After receiving the

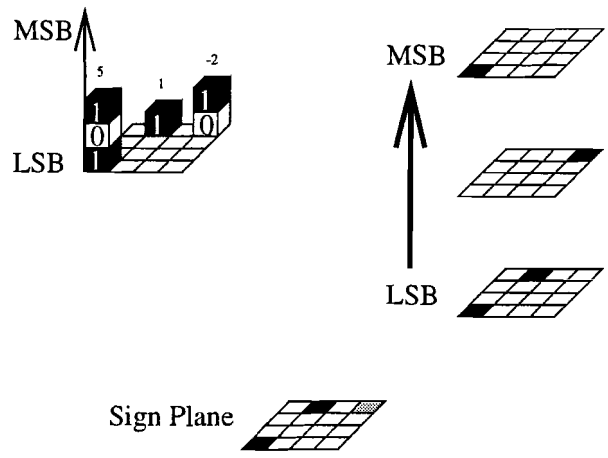

Figure 4.2: Bit plane encoding

third and last bit plane, all coefficients are known. If however, the receiver didn't get the last bit plane, it still has a pretty good idea what the image looks like.
One last plane remains actually. This is the sign plane. In a wavelet-transformed image, negative values can occur. One usually transmits the sign immediately after the first "1" for that coefficient has been transmitted.

The EZW-algorithm uses a threshold for determining the significance of coefficients. After the whole image has been transmitted according with respect to this threshold and the Zerotree mechanism, the threshold is decreased. So, in this way, gradually all information of all coefficients is transmitted to the receiver.

4.2.3 Coefficient Ordering

In order to make use of Shapiro's hypothesis, the parent of a coefficient should be transmitted first. Otherwise, the information that the coefficient is insignificant because it is a ZeroTreeRoot-Descendant would not be known to the decoder. In order to do this, the so called Z-scan was used to determine the order in which the coefficients should be transmitted. See figure 4.3.
Other scans are the raster-scan (row by row or column by column) and the Peano-Hilbert scan [13]. Besides the fact that the parents should be encoded before the children, there is another thing to keep in mind. In order to estimate the probabilities, one can use the neighbouring coefficients. These have to be known to the decoder too. When one uses the raster scan and transmits the coefficients of the first row (or column), the coefficients north of the coefficient are not known. This is also true if one uses the Z- or Peano-Hilbert-scan. In addition, if one uses the Z-scan, the coefficient north-east of the to-be encoded coefficient has not always been transmitted yet. When using the raster scan, the north-east coefficient has already been transmitted most of the time.
In our implementation, we used the Z-scan in the sub bands and used the Z-scan for ordering the sub bands. Shapiro only specifies the ordering of the sub bands, and does probably use the raster-scan for the coefficients. This could work better due to the above mentioned problem.

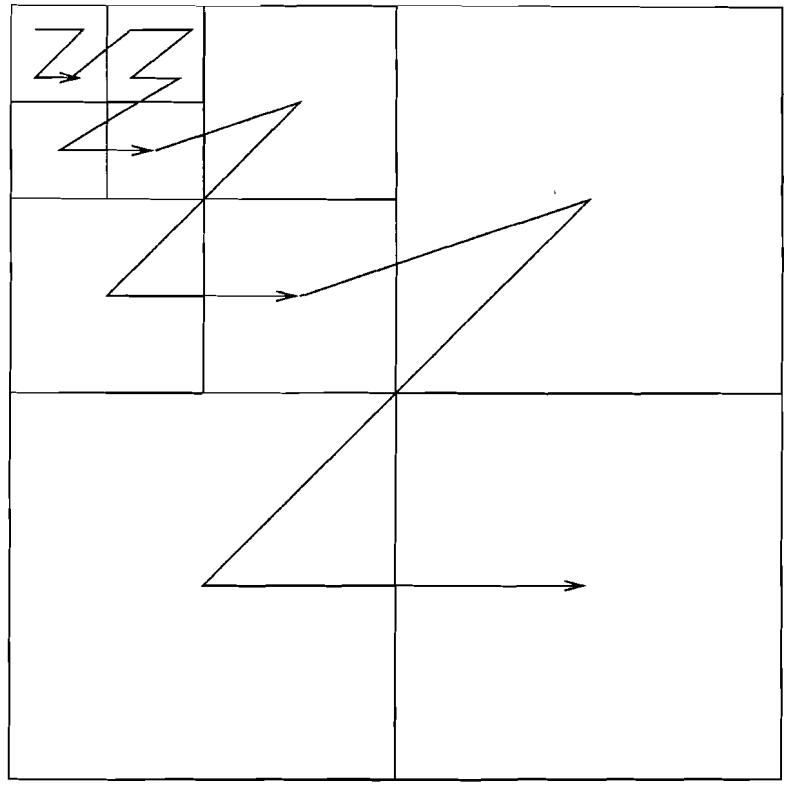

Figure 4.3: Z-scan of coefficients.

This problem was discovered too late to improve our implementation.

4.3 Algorithm

The algorithm consists of two main passes. They are called the Dominant Pass and the Subordinate Pass. Each pass has its own list of coefficients, and every coefficient is in one (and only one) of them. In the starting situation, all coefficients are in the Dominant List. Gradually all coefficients move to the Subordinate List. The two passes are intermittent executed until the threshold is 0. See the first piece of pseudo-code.

```
void main(){
  Threshold=128;                          /* range of coefficients is 0..255 */
  Dominant_List= all_coefficients;
  Subordinate_List = empty;
  while (Threshold>=1) do {
    Dominant_Pass();
    Subordinate_Pass();
    Threshold=Threshold/2;
  }
}
```

The dominant pass will evaluate all coefficients in its dominant list. This evaluation determines if the coefficient is a ZeroTreeRoot, a ZeroTreeRootDescendant, a ISOlated zero or a SIGnificant coefficient. If a coefficient is found to be significant, the coefficient is removed from the dominant list and put in the subordinate list. At the same time a symbol is transmitted

to the decoder (or arithmetic encoder) indicating what kind of a coefficient it is.
The subordinate pass will refine all coefficients in its list. It transmits the next bit of the coefficients value. See the pseudo code of the *Dominant_Pass()* and the *Subordinate_Pass()*:

```
void Dominant_Pass(void){
  for_all_coefficients_in_Dominant_List{
    switch(Coefficient){
      case Positive_Significant : SendD(P-SIG);
      case Negative_Significant : SendD(N-SIG);
      case ZeroTreeRoot         : SendD(ZTR);
      case Isolated_Zero        : SendD(ISO);
      case Descendant_of_ZTR    : Do_not_Send();
    }
  }
}

void Subordinate_Pass(void){
  for_all_coefficients_in_Subordinate_List{
    if (Coefficient>=Threshold/2) SendS(1); else SendS(0);
  }
  Sort_Subordinate_List();
}
```

with

```
void SendD(int to_send){
  Write(to_send);
  Move_Coefficient_from_Dominant_List_to_Subordinate_List();
}

void SendS(to_send){
  Write(to_send);
  if (to_send==1) Coefficient-=Threshold/2;
}
```

The switch/case in the dominant pass is maybe easier to understand with the flow-graph in figure 4.4. It is easy to encode the 4 different symbols (ISO,ZTR,P-SIG,N-SIG) into bits. This has however no influence as the bits/symbols are passed through an arithmetic encoder. As depicted in figure 4.4, if a significant coefficient is transmitted, it is specified whether it is a positive or a negative one.
One procedure hasn't been described yet. That is the *Sort_Subordinate_List()*. In order to receive the most for the eye important bits first (large valued coefficients), the Subordinate List is ordered so that the coefficients with the highest values are processed first. So, coefficients with high values will be processed earlier than coefficients with a low value. The decoder has to be able to order its Subordinate-List too, but it does only know in which range the coefficients are. Therefor, the Sort_Subordinate_List() procedure orders the list based on the known range.
By using the Z-scan, coefficients near to each other in the image, are transmitted not far apart. By the re-ordering, this is no longer true. Coefficients from the whole image can be transmitted at any given point. This is not good for the compression ratio.

4.4 Optimisations

To this scheme several optimisations can be applied.

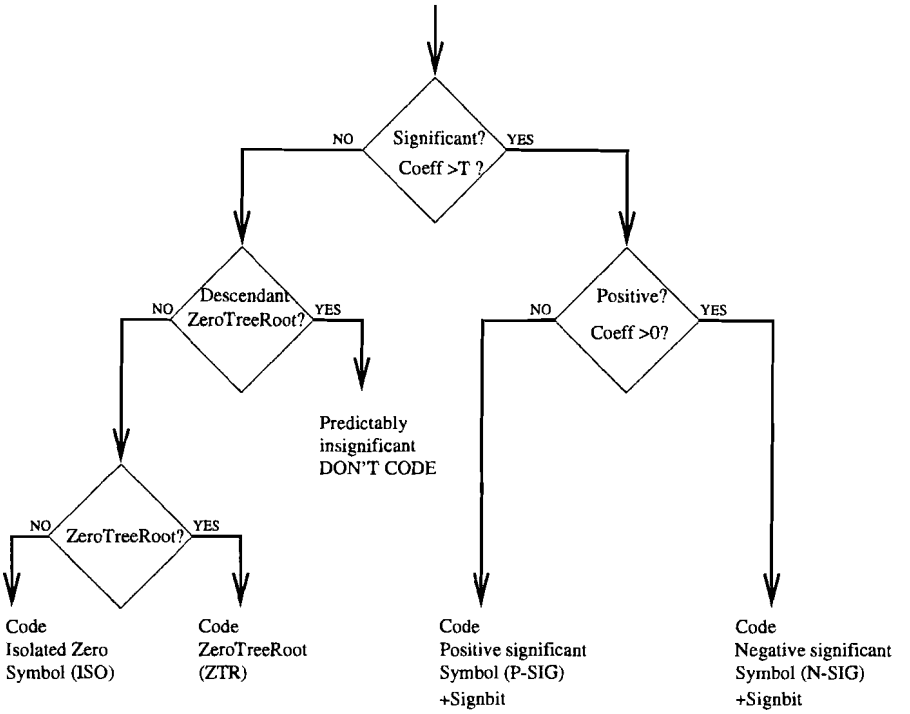

Figure 4.4: Flow diagram for determining coefficient-class

- **Average deduction**
  To lower the average value of the coefficients, the average value of the coefficients in the lowest sub band is calculated and deducted from those coefficients. The average is then transmitted as the first value to the decoder.

  This will increase the number of Zerotrees, because more coefficients will be insignificant with respect to the threshold. But also more coefficients will become negative and thus the symbol for negative significant coefficients will be often used. This increases the probability with which a negative coefficient appears, which in turn will not benefit the compression.

- **Threshold start level**
  If the range of the values of the coefficients is [0..255], then a good threshold starting level is 128 (2 times the threshold is higher than the maximum value). But if all values are lower than this, it is possible to start with 64 or even 32. This optimum starting threshold is calculated and transmitted to the receiver. Of course the receiver has to know the starting threshold in order to decode correctly.

- **Adding zeros**
  Once a coefficient is removed from the dominant list, the value of the coefficient in the image is set to zero. This way, future passes will determine the coefficient as insignificant, and more/larger Zerotrees can be formed. The subordinate pass however will remember the original value and transmit it accordingly

4.5 Results

In this section, some results that Shapiro got with EZW are compared to our EZW results. In figure 4.5 the compression results are plotted for the images Lena and Barbara.

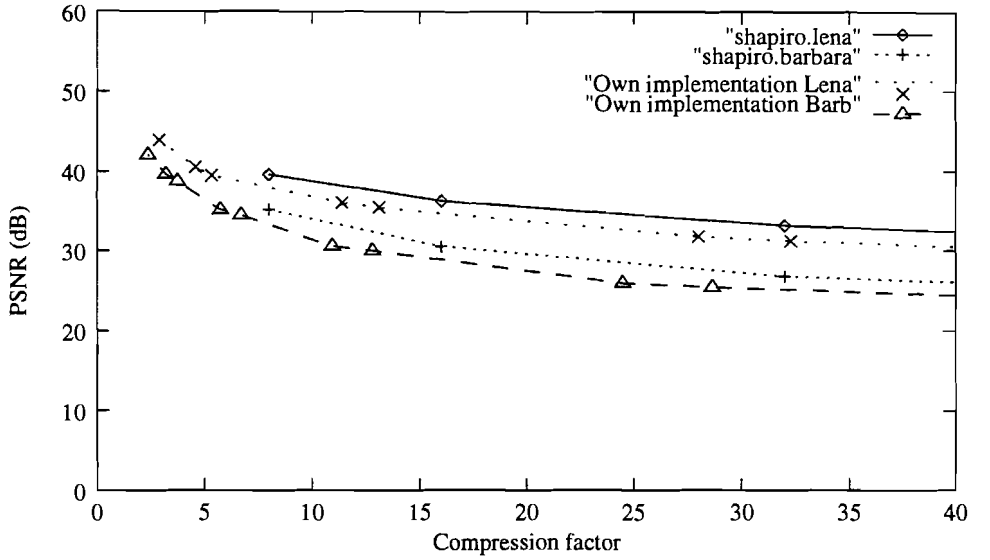

Figure 4.5: Compression ratio for Lena and Barbara

As can be seen in this figure, our EZW-implementation does not reach the compression achieved by Shapiro. Probable causes are:

- We used the same arithmetic encoder ([25]), however, we did not use three different coders as Shapiro did for binary, ternary or quandary data. We used 1 which could handle a 6-valued data.
- Shapiro uses for the Dominant pass four different histograms in the arithmetic coder. We use one histogram. However, according to Shapiro, this will only account for a 0.12 dB loss for Lena when compressed by a factor 16. This is obviously less than the difference shown in figure 4.5.
- We used the Z-scan ordering also for the image coefficients. Shapiro did not specify exactly what he used. He probably used a raster-scan for the ordering of the coefficient.

Chapter 5

Context Tree Weighting

This is the second part of this project report. Here, the more information theoretical CTW-algorithm (this chapter) and an application (chapter 6) of it will be given.

In 1993, Willems, Shtarkov and Tjalkens presented the context tree weighting algorithm during the International Symposium on Information Theory [21], and later in 1993 they presented an article using more general context models during a symposium on information theory in the Benelux[23]. This algorithm, a sequential universal source coding algorithm, estimates the probability of the next symbol on the basis of a general context based on all previously processed data or any other side information available to or computable by the en- and decoder. In order to do this it weights all models of a certain depth using a context tree. The model that matches the real model best, will become dominant compared to the other ones, as the number of processed bits increases. The CTW uses the model that gives the shortest description of the model and the data.

The theory behind this algorithm will be discussed in three steps. First the handling of memoryless sources will be explained. Then the modelling of a known source with unknown parameters, followed by the handling of an unknown source with unknown parameters.

5.1 The memoryless source

One of the most elementary sources is the binary memoryless source with parameter θ (the probability that the output is a 1). If the parameter θ is known then one can easily calculate the probability of a sequence $x_1 \dots x_T$: $P_a(x_1 \dots x_T) = (1-\theta)^a \theta^b$, with a the number of zeros and b the number of ones in $x_1 \dots x_T$. One can feed such a probability distribution to an arithmetic coder like the Elias-algorithm or the one in [25]. This arithmetic coder will then produce a codeword based on this probability distribution $P_c(x_1 \dots x_T)$. This resulting code is a prefix-code for which the codeword length $L(X_1 \dots x_T)$ satisfies (for the Elias code):

$$L(x_1 \dots x_T) \leq \log_2 \frac{1}{P_c(x_1 \dots x_T)} + 2, \text{for all } x_1 \dots x_T \tag{5.1}$$

Thus the coding redundancy (the maximum number of extra bits needed in order to make the code) is always less than 2 bits (for the Elias code).

In the remaining sections the term *cost* of a probability will be used often. It means the length of the codeword corresponding to this probability. This length can be computed with equation 5.1, but the two bits are omitted because they only depend on the arithmetic coder which has to be prefix-free.

But how can one calculate the probability of a sequence, generated by a memoryless source, for which the parameter θ is not known? One could process the sequence, symbol by symbol, and estimate every time the probability distribution of the next symbol, on the basis of the previous ones. It is obvious that if one is at a specific point in the sequence, and so far a zeros and b ones have been seen, that a good estimate of the probability of the next symbol being a zero is:

$$\Pr\{\text{next symbol } = 0 \,|\, \text{after } a \text{ zeros and } b \text{ ones}\} \approx \frac{a}{a+b}. \tag{5.2}$$

A problem with formula 5.2 is that the probability of the first symbol is not defined (0 divided by 0). Also, for $a = 0$ and $b > 0$ (or vice versa) the probability the the next symbol is a 0 is zero. For a sequential algorithm this will cause problems. The Krichevski-Trofimov estimator [7] has the best minimax redundancy and does not suffer from the above mentioned problems:

$$P_e(a,b) \triangleq \frac{\frac{1}{2} \cdot \frac{3}{2} \cdot \ldots \cdot (a-\frac{1}{2}) \cdot \frac{1}{2} \cdot \frac{3}{2} \cdot \ldots \cdot (b-\frac{1}{2})}{1 \cdot 2 \cdot \ldots \cdot (a+b)} \tag{5.3}$$

And $P_e(0,0) = 1$ follows from the exact definition of the KT-estimator:

$$P_e(a,b) \triangleq \int_0^1 \frac{1}{\pi\sqrt{(1-\theta)\theta}} (1-\theta)^a \theta^b \, d\theta. \tag{5.4}$$

It is important to note that the order in which the bits appear in the sequence doesn't influence its probability. In [24] the following upper-bound for the redundancy has been proven:

$$\log_2 \frac{P_a(x_1 \ldots x_T)}{P_c(x_1 \ldots x_T)} \leq \frac{1}{2} \log_2 T + 1, \text{ for all } \theta \in [0,1], \tag{5.5}$$

in which $P_c(x_1 \ldots x_T)$ is the coding probability and $P_a(x_1 \ldots x_T)$ the actual probability, as defined before. This redundancy caused by not knowing the parameter θ is called the *parameter redundancy* (see also section 7.4 and figure 7.22). Thus the total redundancy ρ for the coding of a sequence from a memoryless source is (x_1^T is written instead of $x_1 \ldots x_T$):

$$\rho(x_1^T) \leq \frac{1}{2} \cdot \log_2 T + 3, \text{ for all } x_1^T. \tag{5.6}$$

An important property of the Krichevski-Trofimov estimator is that it can be used sequentially. This is important for the context weighting algorithm and for the image coding. In order to use it sequentially, the current estimated probability and the number of ones and zeros have to be stored. With equation 5.3, the probabilities are:

$$P_e(a+1,b) = P_e(a,b) \cdot \frac{a+\frac{1}{2}}{a+b+1} \text{ and} \tag{5.7}$$

$$P_e(a,b+1) = P_e(a,b) \cdot \frac{b+\frac{1}{2}}{a+b+1}. \tag{5.8}$$

5.2 Known model, unknown parameters

If the encoder and the decoder both know the model and the parameters, it is not so complicated.
First a mapping is defined which maps the output symbols $x_1 \dots x_T$ of the source to the different finite context states. Each context state s consists of a context $u_t(1) \dots u_t(D_s)$ (D_s being the length of the context associated with state s). With every possible context state s, a memoryless source with parameter θ_s (with $\theta_s \in [0,1]$) is associated. If the current context is s then the source will generate a one with probability θ_s. The set of context states s must form a proper and complete set S. If the set is not proper and complete then it is not possible to find a parameter θ_s for every possible context s. It is possible to arrange the context states $s \in S$ in a tree, and because often the mapping maps the last few bits of the output of the source to the context states (with the last symbol of the sequence the first symbol of the context) it is convenient to make it a postfix tree in which one can find in every leaf the parameter belonging to that context. In that case the context state s is $s = u_t(1) \dots u_t(D_s) = x_{t-1} \dots x_{t-D_s}$, otherwise one needs a function σ which maps the output sequence $x_{t-1} \dots x_1$ to the context of s : $s = \sigma(x_{t-1} \dots x_1)$. Thus:

$$Pr\{x_t = 1 | x_1^{t-1}, S, \theta_s\} \triangleq \theta_{\sigma(x_1^{t-1})}. \tag{5.9}$$

Example 5.1 *The tree belonging to the postfix-set* S=*{000,100,10,01,11}* *and* $\Theta_s = \{\theta_{11} = \alpha, \theta_{01} = \beta, \theta_{10} = \gamma, \theta_{100} = \delta, \theta_{000} = \epsilon\}$ *had been shown in figure 5.1.*

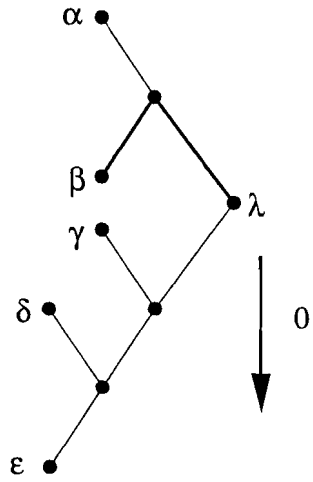

Figure 5.1: An example of a postfix tree

If the context bits are 10, then the next symbol will be a 1 with probability β (see the thicker line in figure 5.1). And suppose that the context is indeed the tail of the sequence then the probability of the sequence 010 (with ...10 as the previous two bits) is :

$$P_e(010|\dots 10) = (1-\theta_{10}) \cdot \theta_{100} \cdot (1-\theta_{01}) \tag{5.10}$$

$$= (1-\gamma) \cdot \delta \cdot (1-\beta). \tag{5.11}$$

The encoding of the tree would cost 2·|S|-1 bits. Here that is 9 bits.

Now, if the encoder and the decoder both know the model but not the parameters, it will get a bit more complicated. Another way to look at it is to say that there are several memoryless sources and the context (known to the encoder and the decoder) will be used to determine which source will be used for the next symbol. One can imagine the entire sequence split in various subsequences, one for every parameter, and each consisting of the symbols following the context of that parameter. In section 5.1 it was described how to deal with memoryless sources. The encoder and the decoder simply use the Krichevski-Trofimov estimator in every leaf. The probability of the sequence can then be computed by taking the product of all estimated probabilities P_e in the leaves.
Thus the probability of a sequence x_1^t, generated by a model S can be expressed with the following formula:

$$P_c(x_1^t|S) \triangleq \prod_{s \in S} P_e(a_s(x_1^t), b_s(x_1^t)), \text{ for all sequences } x_1^t,\ t = 1, 2 \ldots T, \tag{5.12}$$

and with $a_s(x_1^t)$ $(b_s(x_1^t))$ the number of zeros (ones) that followed context s. See [21] for the proofs of the two following upper bounds. The redundancy ρ of a model S with unknown parameters can be upper-bounded by:

$$\rho(x_1^T|S, \theta_s) \leq |S| \cdot \gamma\left(\frac{T}{|S|}\right) + 2, \tag{5.13}$$

with: $\gamma(z) \triangleq \begin{cases} \frac{1}{2} \cdot \log_2 z + 1 & z \geq 1, \\ z & 0 \leq z < 1. \end{cases}$ The following bound on the codeword length is a direct consequence of equation 5.13:

$$L(x_1^T) < \min_{\theta_s} \log_2 \frac{1}{P_a(x_1^T|S, \theta_s)} + |S| \cdot \gamma\left(\frac{T}{|S|}\right) + 2, \text{ for all } x_1^T \in \{0,1\}^T. \tag{5.14}$$

A last remark is that because this is just the multiple use of Krichevski-Trofimov estimators, it is obvious that this approach can be done sequentially. In every leaf of the postfix tree, the number of zeros and ones and the estimated probability are stored. In order to process a new bit, one has to walk through the tree to the leaf which corresponds to the context, and then update the estimated probability P_e with equation 5.7.

5.3 Unknown source, unknown parameters

First the situation is examined in which there are only two possible models, model I and II (see figure 5.2). Both the encoder and the decoder know the models (but not the parameters). The question is now, how to encode the sequence $x_1 \ldots x_T$, generated by one of these models, if one does not know the model. Suppose one has a (good) coding distribution for I and II. If the source was model I, the probability distribution for the sequence $x_1 \ldots x_T$ is $P_c(x_1 \ldots x_T|M_I)$. But if the source was model II, then one should use the probability distribution $P_c(x_1 \ldots x_T|M_{II})$. The *weighted* probability distribution is now:

$$P_w(x_1 \ldots x_T) \triangleq \frac{P_c(x_1 \ldots x_T|M_I) + P_c(x_1 \ldots x_T|M_{II})}{2}. \tag{5.15}$$

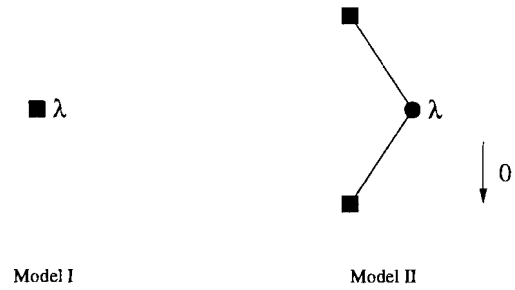

Figure 5.2: Two simple models

This is a good probability distribution for both sources. Call the actual source i (meaning *I* or *II*), then the extra redundancy is at most 1. This follows from:

$$\log_2 \frac{1}{P_w(x_1 \ldots x_T)} \leq 1 + \log_2 \frac{1}{P_c(x_1 \ldots x_T | M_i)}. \tag{5.16}$$

Thus not knowing the model will not cost more than 1 bit. This means that with this method we need not more bits than by identifying which model gives the highest probability for the sequence, encode the sequence with this model and use one extra bit to indicate which model the encoder chose.
If one generalises this result for a set of M models, in which every model $m_i \in M$ has an a priori probability $P(m_i)$ (often $\frac{1}{|M|}$), the weighted probability is defined as:

$$P_w(x_1 \ldots x_T) = \sum_{m_i \in M} P(m_i) \cdot P_c(x_1 \ldots x_T | m_i). \tag{5.17}$$

If $m_a \in M$ is the actual model, then for the *model redundancy* one obtains the following bound:

$$\log_2 \frac{P_c(x_1 \ldots x_T | m_i)}{P_w(x_1 \ldots x_T)} \leq -\log_2 P(m_a). \tag{5.18}$$

Thus not knowing the actual model and using the weighted probability 5.17 costs not more bits than by identifying the best fitting model and transmitting its code (of which the length depends on the probability) followed by the data encoded with this model.

Depending on how the model classes are chosen, there are various versions of the context weighting algorithm (one considers four classes: class I - class IV [22]). In the remaining part of this chapter model class IV will be used. Here the models have the form of a postfix tree as discussed earlier. The number of trees $\#_d$ with a maximum depth D can be recursively calculated with: $\#_d = \#_{d-1}^2 + 1$. To give an idea, of how fast the number of possible models grows, see table 5.1, in which for trees with a maximum depth up to 9, this number is shown. If one wants to compress a sequence, in general the source is not known. That's why one wants to weight over all possible models (of sources) with a maximum depth D (to keep the algorithm simple) using equation 5.17. A major disadvantage is that the number of possible models grows rapidly. It is virtually impossible to calculate the block probability P_c of a sequence for all possible models separately and then weight them with formula 5.17. But, and this is the power of this method, all these models can be weighted *simultaneously* in one

Maximum depth of the tree	Total number of possible models	Maximum depth of the tree	Total number of possible models
0	1	5	$4.6 \cdot 10^{3}$
1	2	6	$2.1 \cdot 10^{11}$
2	5	7	$4.4 \cdot 10^{22}$
3	26	8	$1.0 \cdot 10^{45}$
4	677	9	$3.8 \cdot 10^{90}$

Table 5.1: The number of possible models, given the maximum depth of the tree

simple context tree with sequential updating! A context tree is a full tree of a certain depth D. In every node of the tree one calculates the following equation for the weighted probability:

$$P_w^D(x_1 \dots x_T, s) = \begin{cases} \frac{P_e(a_s,b_s)+P_w^D(x_1 \dots x_T,0s)\cdot P_w^D(x_1 \dots x_T,1s)}{2} & \text{if } l(s) < D \\ P_e(a_s, b_s) & \text{if } l(s) = D \end{cases} \tag{5.19}$$

in which *l(s)* denotes the length of the context s. In the root λ (*l(s)=0*) the coding probability will be [21]:

$$P_c(x_1^T) = P_w^D(x_1 \dots x_T, \lambda). \tag{5.20}$$

Before an upper-bound is given for the redundancy of this algorithm, a function Γ for the cost of a model is defined:

$$\Gamma_D(s) \triangleq |S| - 1 + |\{s : s \in S, l(s) \neq D\}|. \tag{5.21}$$

Now, the total redundancy ρ is upper-bounded by [21]:

$$\rho(x_1^T|S, \theta_s) < \Gamma_D(s) + |S| \cdot \gamma\left(\frac{T}{|S|}\right) + 2. \tag{5.22}$$

So with the context tree weighting method, the model redundancy is not more than the description of the actual model with the natural tree encoding!
A last important bound is the upper-bound on the codeword length. It is obtained by rewriting equation 5.22 and taking the minimum over all the actual source models and parameters (an elaborate proof can be found in [24]).

$$L(x_1^T) < \min_{s \subset T_D} \left(\min_{\theta_s} \log_2 \frac{1}{P_a(x_1^T|S, \theta_s)} + \Gamma_D(S) + |S|\gamma\left(\frac{T}{|S|}\right) \right) + 2 \tag{5.23}$$

Thus even if there is another model that would perform better than the actual model, then the context tree weighting algorithm will still be better than this model.

5.4 Implementation

Like in the previous sections, it is important to notice that the context tree weighting algorithm can be implemented sequentially. In order to do this, the number of zeros and ones, the estimated probability (for the Krichevski-Trofimov estimator) and the weighted probability are stored into each node (or leaf). The actual context tree weighting algorithm with sequential updating can be described now in the following steps.

1. Walk recursively through the tree to the leaf which corresponds to the context.

2. Update the number of zeros and ones and the estimated probability P_e with equation 5.3. And if the current state is an internal node of the context tree, then update the weighted probability with equation 5.19. This can be done fast because one level higher in the tree the weighted probabilities of both subtrees are already available.

3. If not in the root then track one step back in the tree and continue with step 2.

As one can see the algorithm searches a path through the tree, and along this path the various results will be updated, but every change is local. If the tree has a maximum depth D, then only 1 leaf and D internal nodes have to be updated. This makes the execution of this algorithm relatively fast. It is possible to generate only the parts of the context tree that are really used, by starting with an empty tree, and make a node (or leaf) every time the updating algorithm wants to access a node (or leaf) that doesn't exist yet. This is more memory efficient (remember that the context tree can grow very large, especially when the depth is large). It is even possible to save more memory, because nodes of which the context becomes unique do not have to be split any further. But this has the disadvantage that if during a later update in this node a context with a different tail appears, that then the previous context has to be examined again to determine its tail in order to extend the node. This means that a lot of extra administration has to be done, every time an update has not used his entire context. The remaining part of the context and the corresponding value of the class have to be stored in that node, in order to make this result available, if the node has to be extended at a later update.

Chapter 6

Minimum Description Length

In order to get a good selection and ordering for the context symbols, an algorithm is used to measure the compression for each ordering. This is done by the Minimum Description Length (MDL)-algorithm. The algorithm is based on the CTW-algorithm but then on the class III sources. First the transition from CTW to MDL is explained (section 6.1), then the results and measurements are given (section 6.2).

6.1 From CTW to MDL

In [18] two main modifications to the CTW-algorithm were made in order to derive MDL-decision trees. From these trees several things can be concluded. One of them is which context bits to use and how they should be ordered to gain the most from the CTW-algorithm. That is what it is used for here.
The two main modifications are detailed in the following subsections.

6.1.1 From weighted to maximising probabilities

We now want to select a source model that assigns a high probability to the source sequence while taking into account the model cost. For this purpose we need an algorithm that explicitly selects a model. We shall discuss an adaptation of the CTW algorithm where we replace the weighting with a maximising operation.
In order to determine these maximising probabilities, now only two models are considered. First the probability for the sequence $x_1 \dots x_T$, assuming model I has been used : $P_c(x_1 \dots x_T|M_I)$ will be calculated. Then the probability of the sequence assuming model II had been used: $P_c(x_1 \dots x_T|M_{II})$ will be calculated. Independent of some a priori probabilities (even if one model is more likely then the other, still 1 bit is needed to determine which model gives the lowest description length) the description length will be minimised with the following maximised probability:

$$P_{\max}(x_1 \dots x_T) = \max(\frac{1}{2} \cdot P_c(x_1 \dots x_T|M_I), \frac{1}{2} \cdot P_c(x_1 \dots x_T|M_{II})), \tag{6.1}$$

in which the factor $\frac{1}{2}$ is needed to account for the extra bit necessary for identifying the best model. The generalisation of this idea to a set M with $|M|$ possible models, in which every possible model $m \in M$ has an a priori probability $P(m)$ is (see also 5.17):

$$P_{\max}(x_1 \dots x_T) = \max_{m \in M}(P(m) \cdot P_c(x_1 \dots x_T|M)). \tag{6.2}$$

As one can see, the coding probability *plus* the probability which is inverse proportional to the cost for identifying the model, is maximised. With the same steps as used in the previous chapter, and with the same definition for the a priori probabilities (which account for the number of bits needed to identify the model), the following equation for the maximised probability is obtained, to be used in a context tree (see also 5.19):

$$P^D_{\max}(x_1 \dots x_T) = \begin{cases} \frac{\max(P_e(a_s,b_s),P^D_{\max}(x_1 \dots x_T,0s)\cdot P^D_{\max}(x_1 \dots x_T,1s))}{2} & \text{if } l(s) < D, \\ P_e(a_s,b_s) & \text{if } l(s) = D. \end{cases} \tag{6.3}$$

This probability will give us the model which minimises the description length as proved in [18].

6.1.2 Changing Classes

The CTW algorithm described so far is based on the class IV context weighting algorithm (the context *tree* weighting algorithm). But that is just one of the possibilities to classify different models. For the application of finding the most efficient order of context-bits, the class III algorithm is needed. The order of the context bits is important because if important coefficients would be high in the tree, these nodes/leafs would be present many times. Which does not give a better estimated probability.
The class III algorithm finds the best context tree by weighting all possible trees in each possible order. This search space is vast. See for example the tree for depth 3 in figure 6.1. It is obvious that the experiments/measurements with this algorithm can take a long while (some measurements took up to 500 hours to complete).
The extra determination of the best order of context bits is accomplished by the following equation for the weighted probability as presented in [22] (a shorter notation is used):

$$P^D_w(x_1 \dots x_T, S, s) = \frac{P_e(a_{S,s},b_{S,s}) + \sum_{p \notin S} P^D_w(x_1 \dots x_T, \{p\} \cup S, 0s) \cdot P^D_w(x_1 \dots x_T, \{p\} \cup S, 1s)}{D - |S| + 1}. \tag{6.4}$$

The set S contains the positions which have been used already in the tree. The corresponding contexts have been collected in the state s (which has a value for every position in S, and a "don't care" for the others, see the notation in figure 6.1). In this node, a weighting should be done over the empty split (P_e) and all possible remaining split positions p (all $\{p \notin S\}$). For the maximised probability algorithm the equation is then:

$$P^D_m(x_1 \dots x_T, S, s) = \frac{\max_{\{p\} \notin S} \left(P_e(a_{S,s},b_{S,s}), P^D_m(x_1 \dots x_T, \{p\} \cup S, 0s) \cdot P^D_m(x_1 \dots x_T, \{p\} \cup S, 1s) \right)}{D - |S| + 1}. \tag{6.5}$$

Now, the CTW-algorithm has been adjusted for use with the class III algorithm and it will give the model (decision tree) with the minimum description length.
The next section will go into more detail with measurements taken with the MDL-algorithm.

6.2 Measurements

In order to investigate the most efficient ordering of the contexts, and what contexts are more important than others, the MDL-tree was calculated for several parts of image data

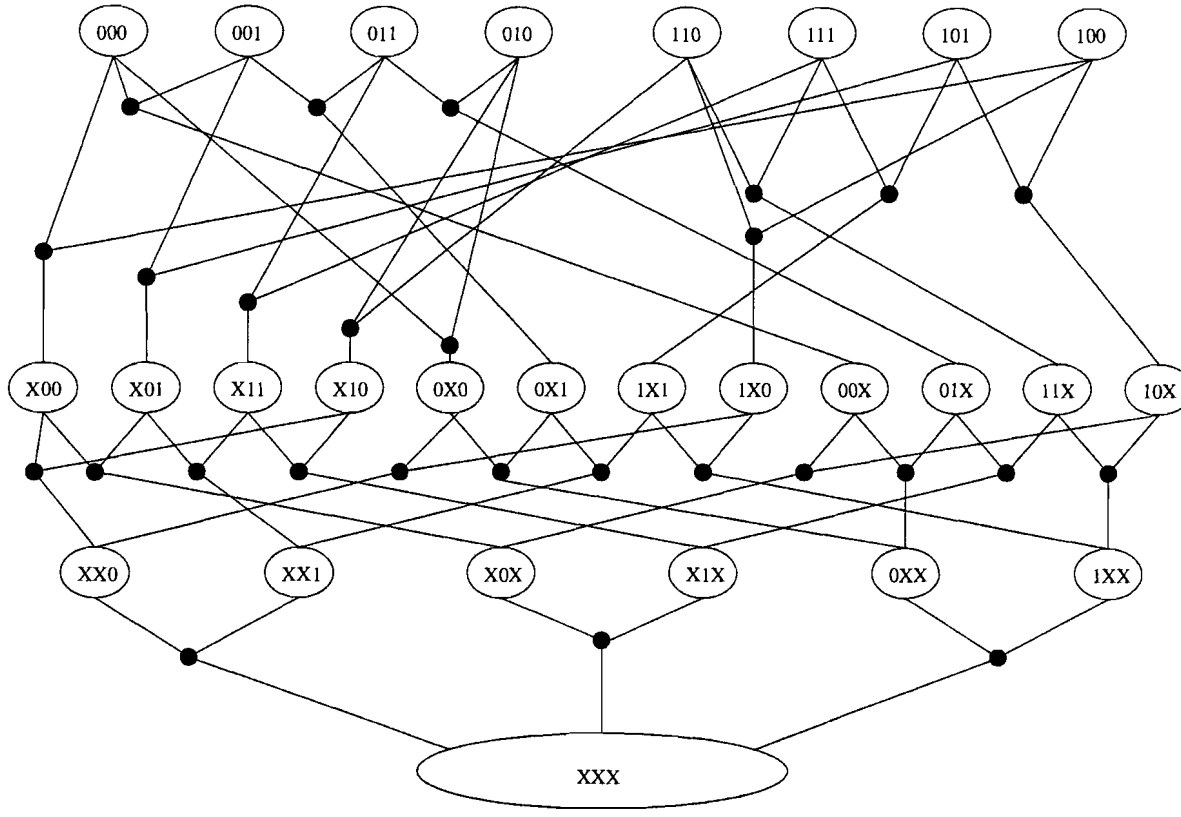

Figure 6.1: A full depth 3 search tree

from Lena and Barbara with their defined context. In the next two subsections, two different contexts were used. With these measurements, we hoped to get a better understanding of which contexts are important (for these images and probably images in general).

6.2.1 First measurement

The context used for these measurements consists of 10 bits. For each coefficient near the to-be encoded coefficient, X, (W,NW,N and NE (see figure 6.2)) and the parent of the to-be-encoded coefficient, 2 bits are assigned to the context. The 2 bits specify if the coefficient is ZeroTreeRoot (0 0), ISOlated zero (0 1), Significant (1 0) or ZeroTreeRootDescendant(1 1). The to-be encoded coefficient can also be one of those four values (there are of course, no parent coefficients that are ZTRD).
Conclusive, the context for the first measurement is:

```
  P    W    NW   N    NE
1 2  3 4  5 6  7 8  9 10
```

For each image, the EZW program creates three context-files for the dominant pass coefficients. Each with the context specified above and one to-be-encoded bit. Why this is done is explained in section 7.1. With each of the three context-files, a MDL-tree is calculated. A fourth MDL-tree is calculated from the binary output of the subordinate pass.

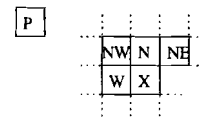

Figure 6.2: Context coefficient for the first measurements

From the measurements with Lena and Barbara, it looks like the coefficients W and N, as well as the Parent coefficient are the most important. These context bits were always present near the root of the decision tree. Because the wavelet transform works in horizontal and vertical direction only, added with the fact that the N and W coefficient are nearest to the to-be encoded bit, it is not that strange that these coefficients are important to estimate the probabilities. That the Parent coefficient is also important, is a confirmation of the correctness of the Zerotree, but also indicates that there is more dependency that can be used and is not covered by the Zerotree.

6.2.2 Second Measurements

To further investigate other contexts, new measurements were done with the following context:

```
 P    W   NW N NE   PW    PN   W==N PW==PN
1 2   3   4   5 6   7 8   9 10   11     12
```

Here the bit of W,NW,N and NE is the second bit of the previous used context. So it indicates whether the context-coefficient is a ZeroTreeRoot or a significant symbol, or if it is an isolated zero or a ZeroTreeRoot descendant. We also added the information of the West and the North coefficient are equal, and the same information for their parents.
This proved to be too complicated to be executed within reasonable time (a few days per context file of 30.000 contexts). So two bits were left out, and the following context was used:

```
 P    W N   PW    PN   W==N PW==PN
1 2   3 4   5 6   7 8    9      10
```

We still can compare the important bits from the first measurements with the new context bits.
The new context bits are: the parent of the northern and the western coefficient, the information that specifies if these two parents are equal, and the information that specifies if the western and the northern coefficient are equal. Because the northern and western coefficient are important, we thought it could be that only one of them could be sufficient.

From these measurements we conclude that these new context bits are less important. The decision trees were much shorter (an indication that not much important context bits were found) and the first 4 bits (P W and N) were always present near the root of the tree if there was an other context bit in the tree. The added context bits which we included to investigate whether only one of the northern or western coefficient was sufficient, were sometimes also present in the MDL-tree. This could mean that specifying only the western coefficient would be enough. We did not look at this further.

Chapter 7

Transforming from EZW to CTW

In this chapter specific information is given how the EZW bit stream is converted and used with the CTW algorithm. After that, some CTW fine tuning is done with different parameters in the CTW algorithm.

7.1 Dominant Pass

In section 6.2, it was said that for each image, three different context files were created. This is because the MDL algorithm can only be decisive on 1 bit (note: the CTW-algorithm can handle quadruples, but that feature was not used here). So the quadruple that Shapiro specifies is binary decomposed into 3 decision points. See figure 7.1

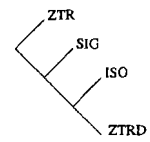

Figure 7.1: Binary decomposition of a quadruple

This is done is this way because ZTR's are the most common type of coefficient, then the SIG's, followed by the ISO's. ZTRD is the least common kind of coefficient, because it does not have to be encoded. So, decompositioned like this, it will produce the least amount of data to be compressed. With this binary decomposition, there are three decision points. The first is the decision whether it is a ZTR or not. If it is not, the the next decision has to be taken, is it a SIG or not? If it is not, the last decision has to be taken. Is it a ISO or a ZTRD? These three contexts files were further compressed with the CTW algorithm. This is detailed in section 7.3 and further.

7.2 Order of the wavelet used

It was already clear that the Daubechies 9/7 wavelet was the best (chapter 3). For 4 different orders it was measured how good the compression/quality ratio was.
In figure 7.2 the horizontal scale is the compression ratio (compared to the 512x512 original image of 256kb). The vertical scale is the quality of the compressed image.

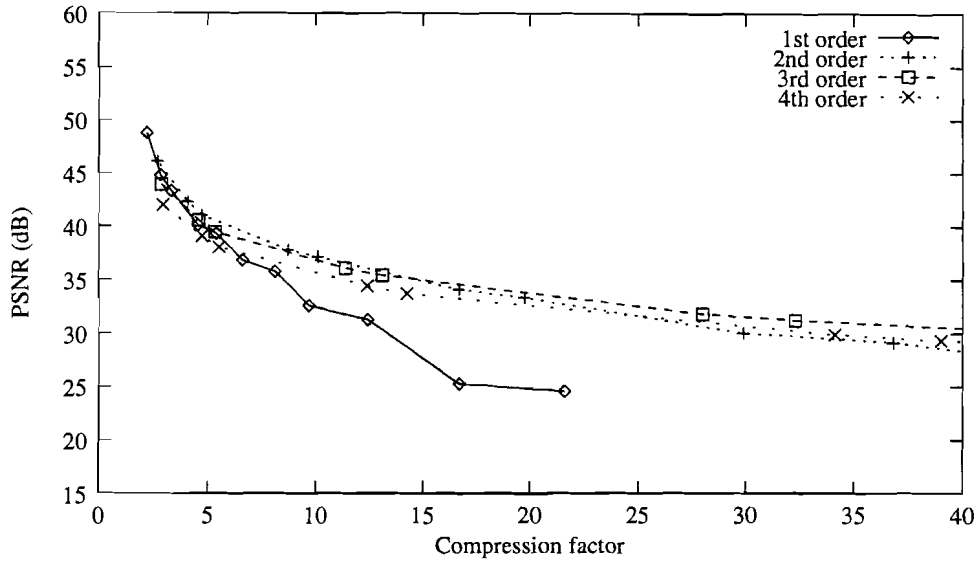

Figure 7.2: Different orders with Daub 9/7 wavelet on Lena

As can been seen from this plot, a 3^{rd} order decomposition will work the best, although a 4^{th} order decomposition achieves similar results.

7.3 Non-stationaries

While transmitting the coefficients to the decoder, the probabilities of the coefficients change. For instance, when processing the shoulder-part of the Lena-image the coefficients gradually change. While processing the feather-part of the Lena-image, the characterise and probabilities are totally different. This changing of the probabilities is called a non-stationarity. The CTW-algorithm however uses all the previous processed data to estimate the probabilities. So after a while it uses the old and therefor incorrect context. This has a negative effect on the compression ratio.

To investigate how the non-stationary characteristics in images should be dealt with in the CTW-algorithm, several measurements of different solutions were taken. There is the local divide (subsection 7.3.1), the global divide (subsection 7.3.2) and its combination (subsection 7.3.3). After that there is a subsection on dividing by more than two. For an overview of the figures in this section, see table 7.1 on page 52.

7.3.1 Local divide

The *local divide*-method consists of a division in a leaf or node when the number of contexts corresponding to that leaf/node reach a maximum. The number of occurrences of zeros and ones is then divided by two (rounding to the nearest higher integer). Nothing is changed to other nodes in the tree. By doing this, the divided node will be more adaptive, while not losing all its characteristics. A compromise is expected when there are so much divisions that the estimated probabilities can't follow the actual probabilities of the image anymore and the

compression increases.

Measurements were taken with several images for several orders of wavelet transforms . The used wavelet was the Daubechies 9/7, because this wavelet seems to be the best. Of most measurements a plot was made. These are the following (7.3 to 7.8):

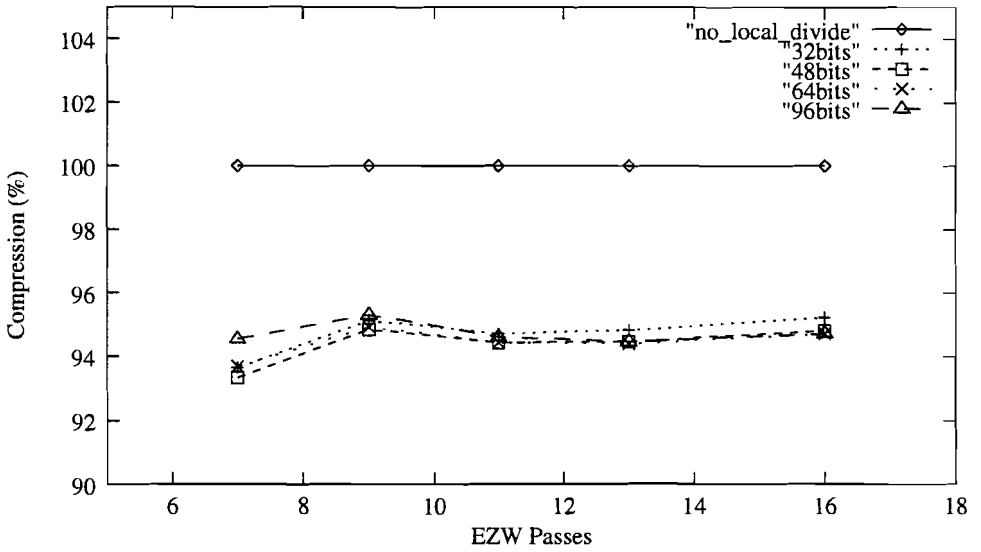

Figure 7.3: Lena order 3 (local)

The vertical scale in figure 7.3 is the compression rate which is achieved by using local division compared to the compression rate with no local division (in percentages). This way we'll get a reasonable feeling how good this change in the CTW algorithm is, and we can compare it with global and other divisions.
The horizontal scale is the number of EZW-passes that are processed. This stands in direct correlation with the quality of the image. With 16 EZW passes (8 times a dominant and subordinate pass) the quality of the resulting image is at its maximum (for the wavelet used). The data used for compression was taken from the dominant pass only. This is because the data from the subordinate pass has other characteristics. Subordinate data is the subject of section 7.5. So, at pass 7, the data (if the coefficient is a ZTR, ISO or ZTRD plus its context) comes from the first four dominant passes. For the other plots the same horizontal and a similar vertical scale is used.
There are 5 different lines in figure 7.3. The *no_local_divide* is compression reached when there are no divisions made for implementing non-stationaries compared to itself. So this line is the same as the a horizontal line on 100%. The *32bits*-line is the compression when the maximum number of counts in a leaf or node before it is divided is 32, compared to the method with no divisions. The maximum number of occurrences of ones or zeros is expressed in bits because when the algorithm divides, there have been that maximum number of binary symbols that were either 1 or 0.
As can be seen in figure 7.3, the difference between a maximum at 48 or 64 is very small. For compression purposes it does not really matter if one chooses 48 bit or 64 bit (or even something in between).

For different orders and also for the test image Barbara, the same measurements were taken and those were plotted in figure 7.4 till 7.8.

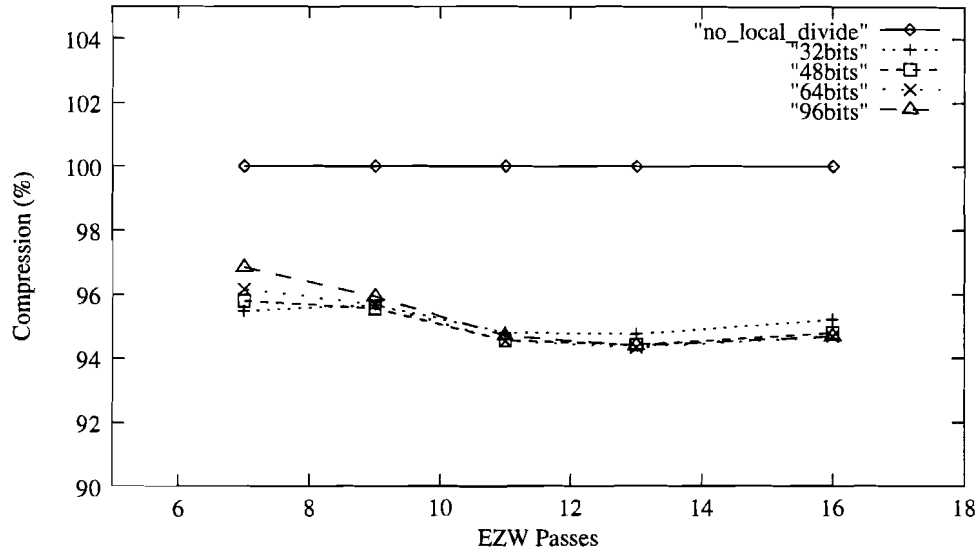

Figure 7.4: Lena order 4 (local)

For the Barbara and Lena image (with the exception of the 3^{rd} order Lena measurement) it can be said that taking 48 as the maximum number at which is divided gives generally the best result. Of one wants to take a slightly higher maximum (for some reason), the compression is not compromised.

7.3.2 Global divide

Another way to compress non-stationary sources better is to reset all statistics present in the tree when a certain maximum has been reached. This maximum will be reached in the root of the tree (because all contexts apply to the root). Shapiro however, resets his arithmetic encoder when the EZW-algorithm starts a new pass. This was also incorporated into the CTW-algorithm, but first the more general approach was taken.

When a certain maximum has been reached, the whole tree has its characteristics divided by two. Again there is the same compromise of reseting the tree too frequent, or too less. Measurements were taken for the Lena and Barbara test images.
From figure 7.9 till 7.14 we can conclude that a maximum of about 750 will work the best. This number is pretty low. It was expected that it would be much higher (approximately. 3000 bits (400kbit of information divided over 16 passes)).

As said, Shapiro resets his arithmetic coder totally at each pass. So far here the coder was not reset totally, just more generalised (number of 1's and 0' divided by two). Here, also at each pass, the tree is totally reset, divided by 2, 4 or 16. See figures 7.15 and 7.16.
As can be seen from this image, resetting the whole tree at each pass does not work very well. (note: the figures have a very small vertical range. (98%-102%), so the figures may look very

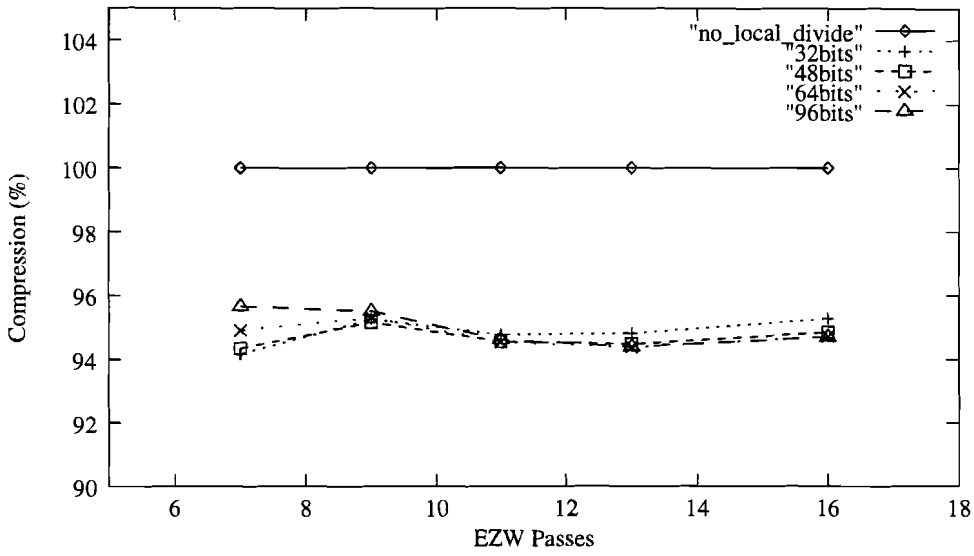

Figure 7.5: Lena order 5 (local)

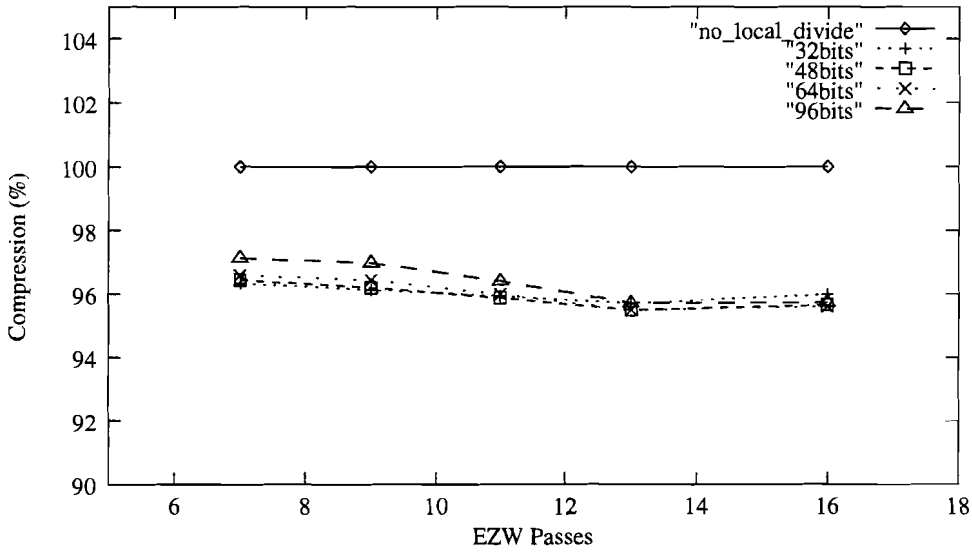

Figure 7.6: Barbara order 3 (local)

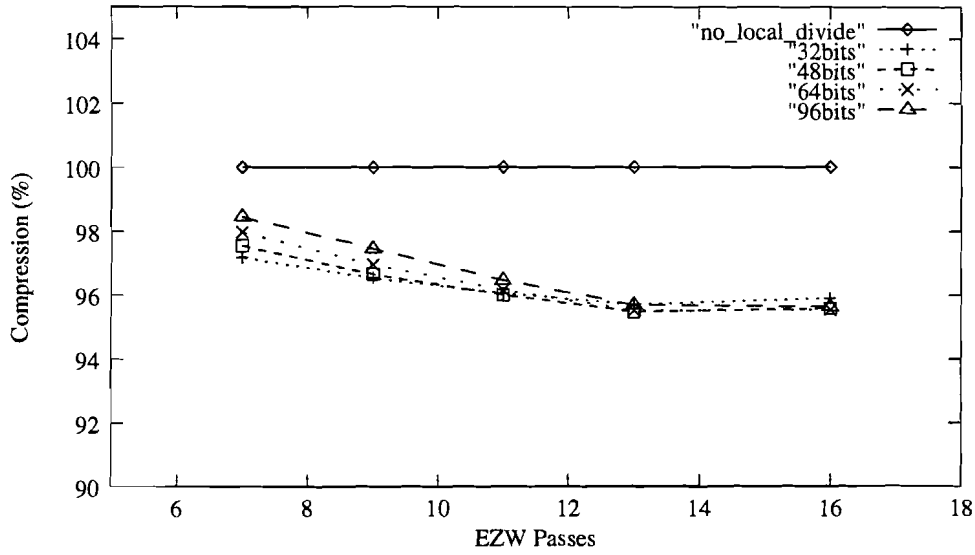

Figure 7.7: Barbara order 4 (local)

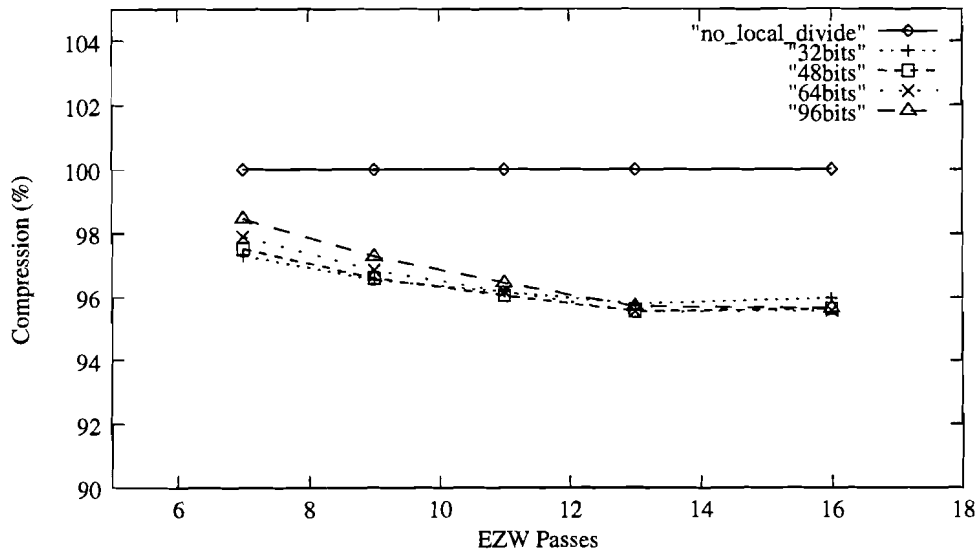

Figure 7.8: Barbara order 5 (local)

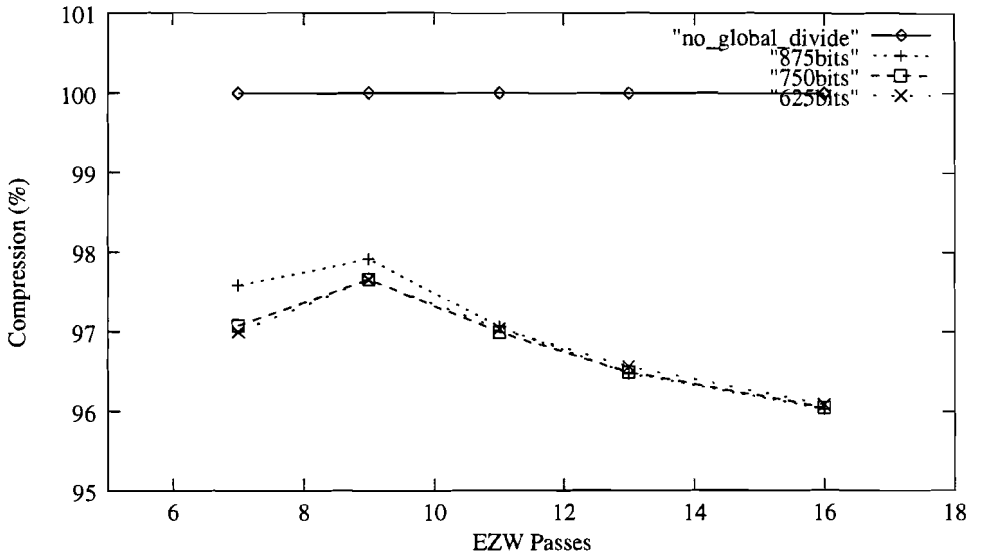

Figure 7.9: Lena order 3 (global)

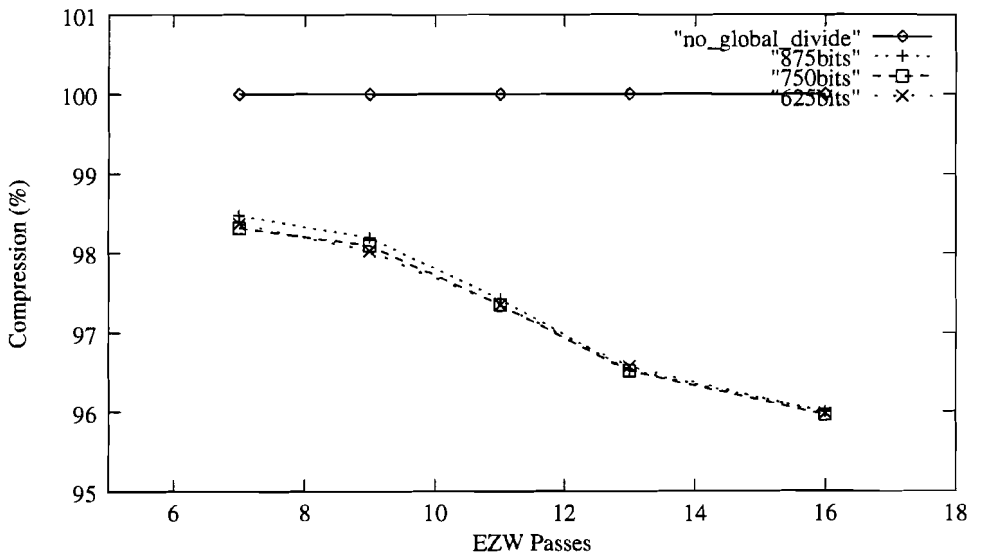

Figure 7.10: Lena order 4 (global)

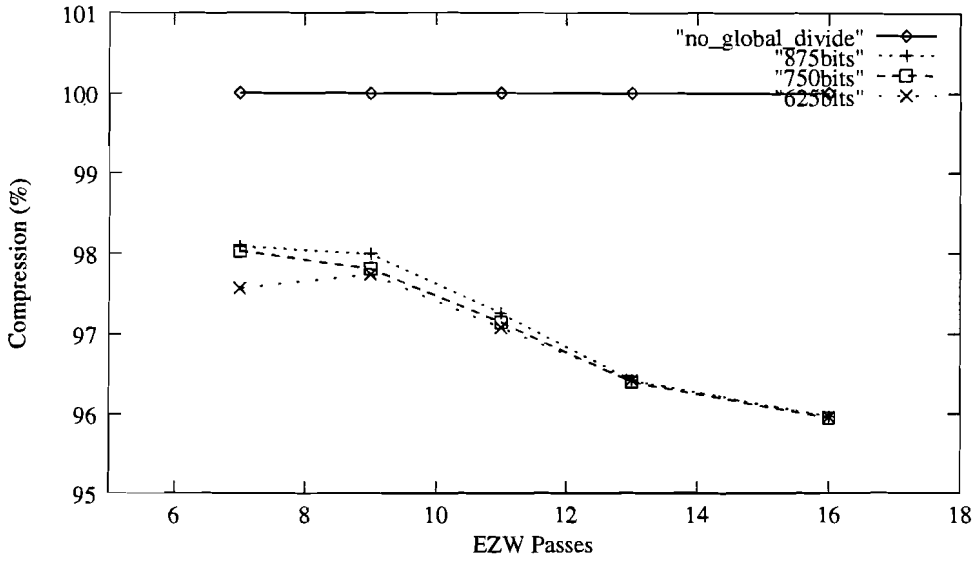

Figure 7.11: Lena order 5 (global)

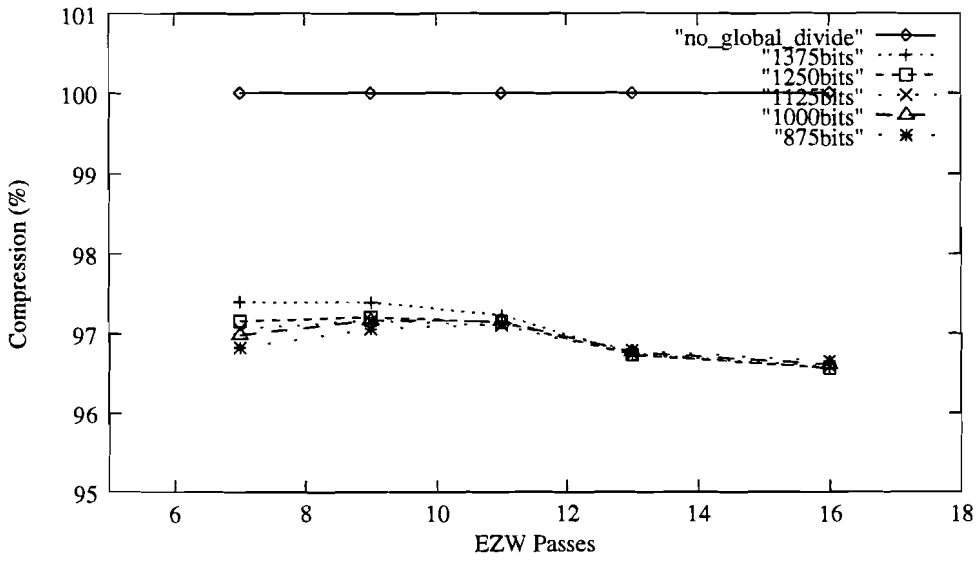

Figure 7.12: Barbara order 3 (global)

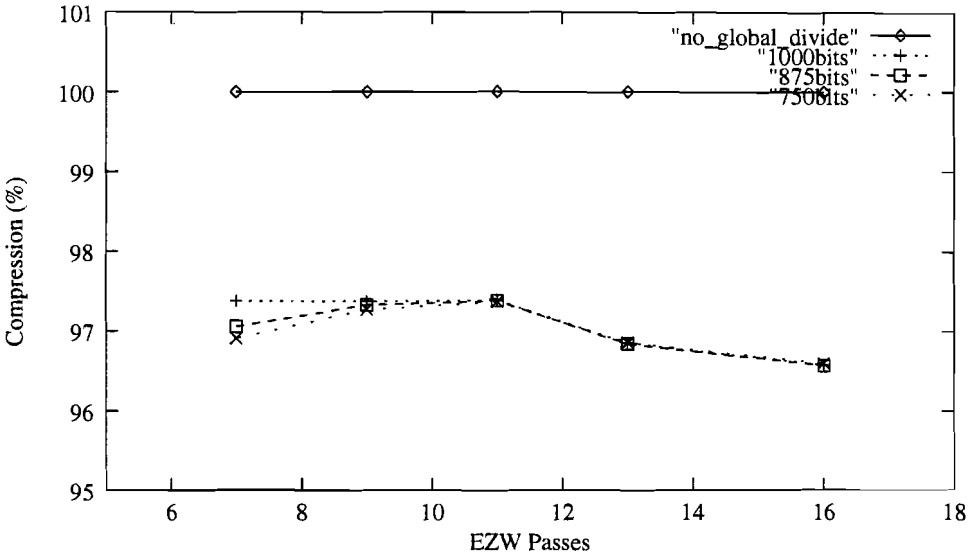

Figure 7.13: Barbara order 4 (global)

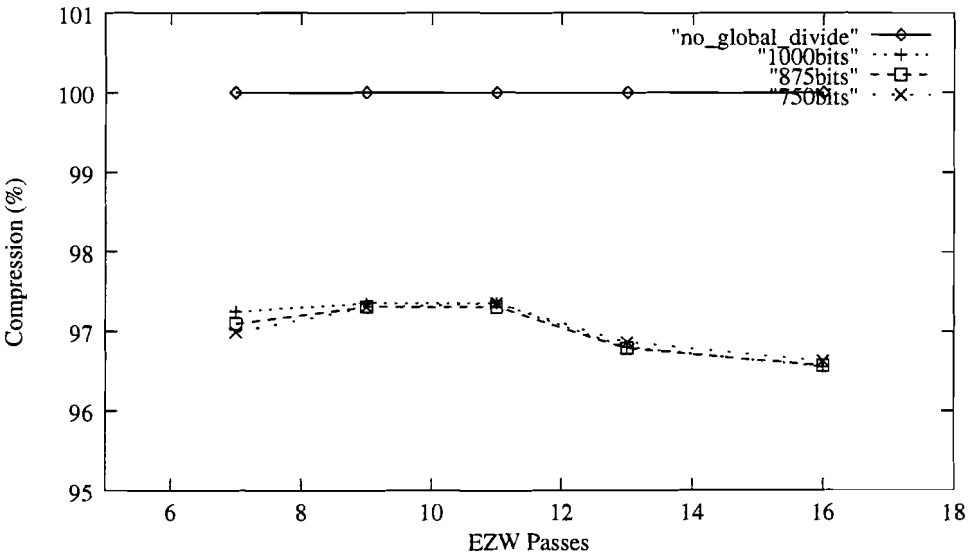

Figure 7.14: Barbara order 5 (global)

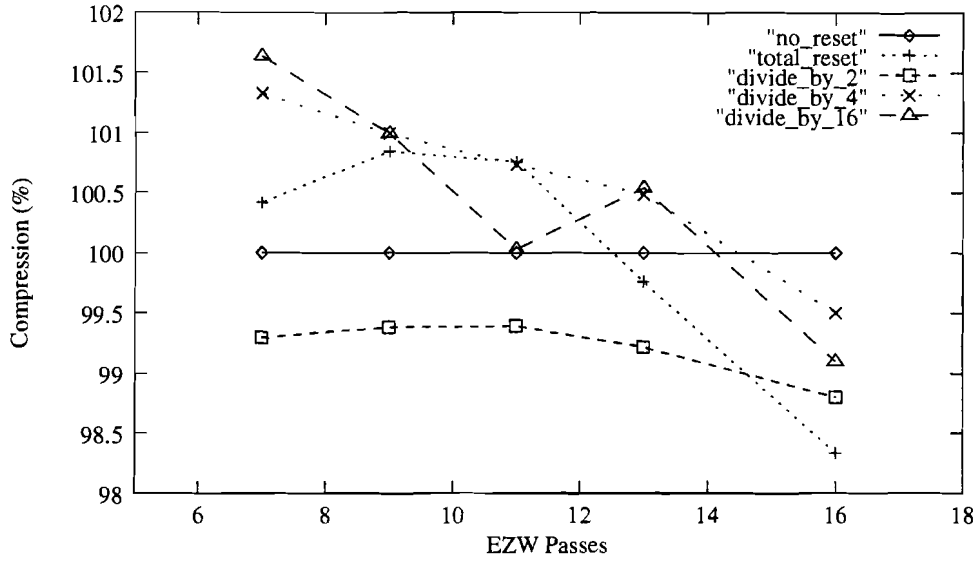

Figure 7.15: Lena order 3, 1 reset/pass

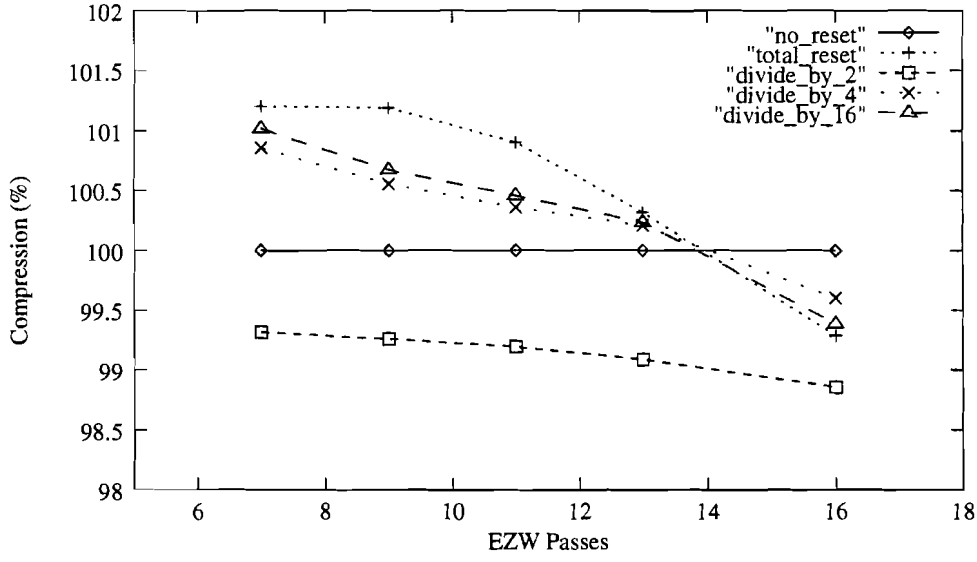

Figure 7.16: Barbara order 3, 1 reset/pass

rough)

What probably the best compression with dividing/resetting is a combination of local and global divide.

7.3.3 Local and Global divide

A problem with *local divide* only, is that a few weighted probabilities extracted at the beginning of the data-stream, are present in the whole rest of the compression, while these probabilities may not be applicable anymore. So, a combination of local and global divide can work better.
Expected is that for the best compression the global divide will be carried out at a higher maximum. The local dividing is expected to be at about the same maximum as the best for local dividing only.
Various measurements were taken. Ranging from *local divide* at 32 bits to 128 and a *global divide* from 5000 to 30000 bits. The best compression for the test image Lena at third order was reached at a *local divide* of 64 and a *global divide* at 20000 bits. Together with the best compressions from *local divide* and *global divide* this compression is plotted in figure 7.17.

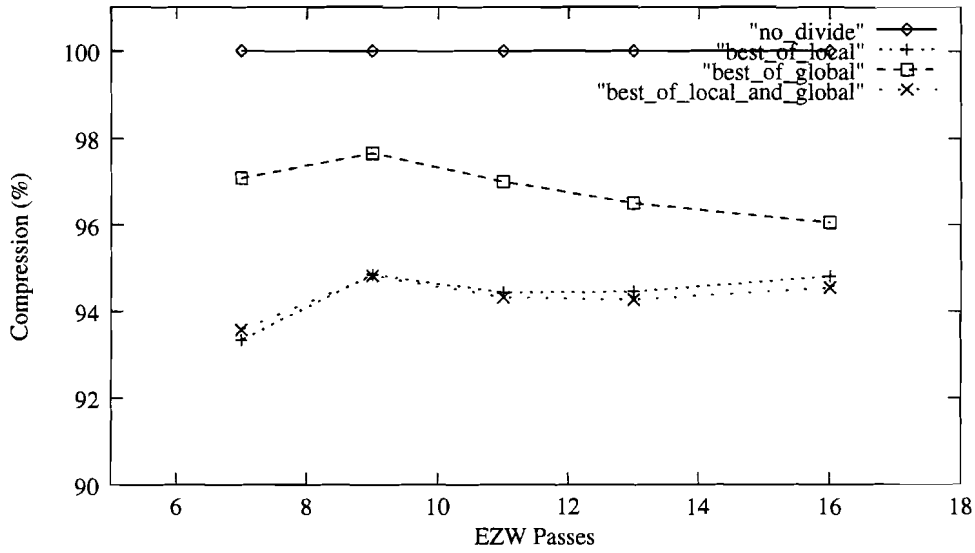

Figure 7.17: Local and Global comparison for Lena

The compression is just a little better than the *local divide*-only approach.
There remains the possibility that a global division (by 2) at each pass, combined with a certain local division is better. This would get rid of the fact that global division for a high maximum does not work for few EZW passes (there is not enough data). The measurements proved that this is a better method to compress. When using a local division at 64 bits, the global halving at each pass will result in a slightly better (0.1%-0.2%) compression (it is also a better than the compression in the previous image).

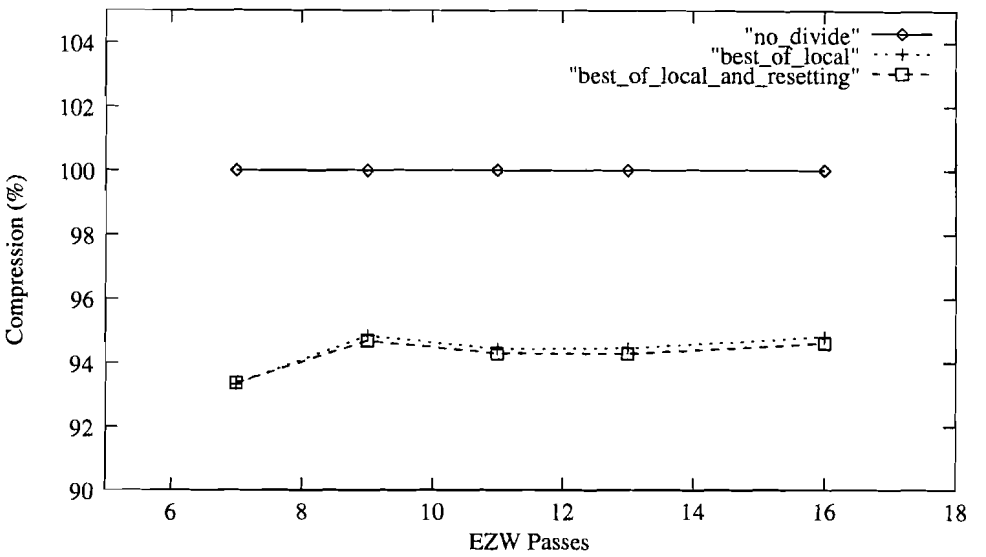

Figure 7.18: Lena 1 reset/pass

7.3.4 Local Dividing by more than two

In the subsection 7.3.1, the statistics were divided by two. This does not have to be the optimum. If a higher divider is used, the statistics will lose more of its specific characteristics. In order to compress as good as the case in which the division is two, the number of times the division is done will probably have to decrease. Otherwise the model can't converge to the optimum probabilities. So it is expected that the maximum at which the division is carried out will be higher than in the case of a division by two.
First the divider was changed to four. See figure 7.19 and 7.20.
There is little difference between the division by two and the division by four. Looking closely at the data reveals that the division by two is slightly better. Also, the division by four is better at a higher maximum (64 to 96 bits instead of 48 to 64), as expected.

With a division of 16 the 3rd order Lena measurement is depicted in figure 7.21. This division does not compress better than the division by two.

Concluding we can say that for the best compression, the CTW algorithm has to divide by two locally at every 64-96 bits, and a global division at each new EZW-pass.

7.4 Different Estimators

In the previous sections, the KT-estimator was used (as also explained in chapter 5). But this does not have to be the best choice. Any memoryless estimator can be used. Consider the following generalised estimator. Suppose the sequence so far has a zeros and b ones, then the probability that the next symbol is a 0 is defined by:

$$P(a+1,b) = P(a,b) \cdot \frac{a+\alpha}{a+b+2\alpha}, \ \alpha > 0. \tag{7.1}$$

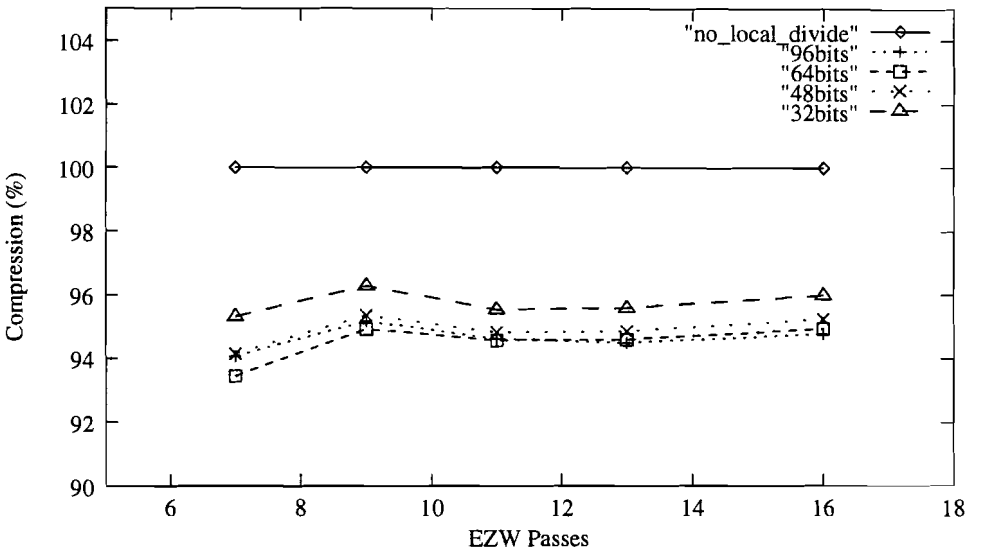

Figure 7.19: Dividing Lena 3rd order with 4 (local)

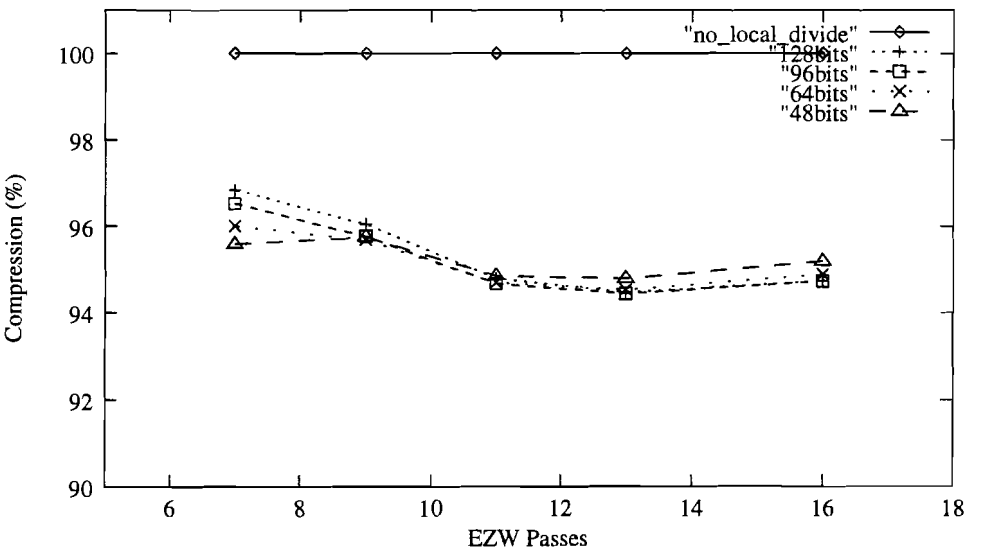

Figure 7.20: Dividing Lena 4th order with 4 (local)

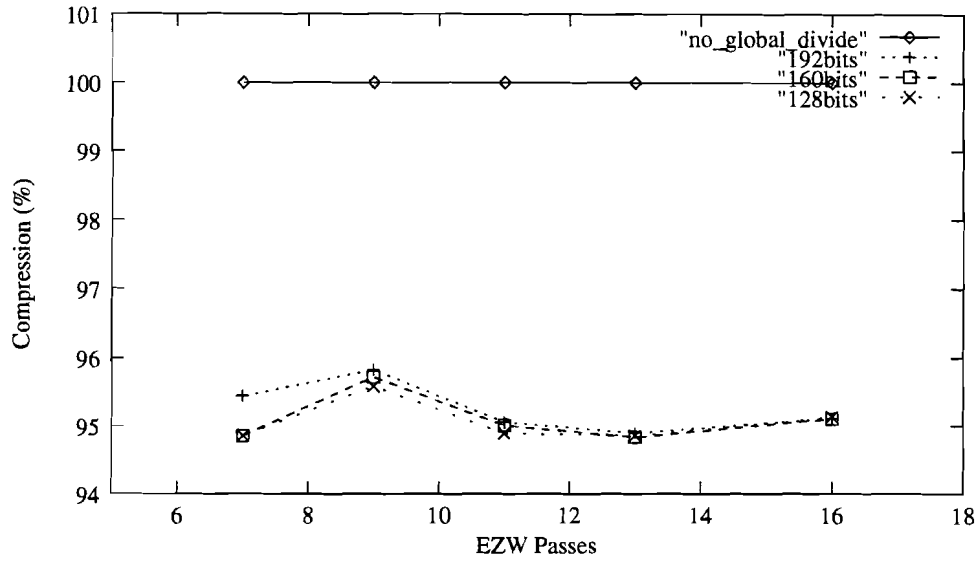

Figure 7.21: Dividing Lena 3rd order with 16 (local)

This estimator correctly assigns probability a $\frac{1}{2}$ to the first symbol. But the choice of α is important, especially for short sequences. A very large α will estimate the first few symbols with a probability around a $\frac{1}{2}$, and will converge slower to its end value (this is of course also affected by the local and global dividing, as we will see). It will have a low redundancy for sequences with symbol probabilities near a $\frac{1}{2}$, and it will have a higher redundancy for low entropy sequences. On the other hand, a very small value for α will have almost no effect after a few symbols. If the first symbol is a 0, then the probability that the next symbol is a 1, is very small. Thus a small α will give a low redundancy for sequences with a parameter near 0 or 1. With $\alpha = \frac{1}{2}$ equation 7.1 reduces to the Krichevski-Trofimov estimator, and for $\alpha = 1$ it is identical to the Laplace estimator [8, 9].
The redundancy of this estimator for $\alpha = 1, \frac{1}{2}, \frac{1}{8}, \frac{1}{16}$ is plotted in figure 7.22 in case there are 100 bits. The redundancy has been computed by comparing the probability of the estimator for a sequence of 100 bits, containing between 0 and 100 ones, to the "actual" probability. The actual probability is computed with (given that there were b ones and $a = 100 - b$ zeros):

$$P_a(a, b) = \left(\frac{a}{a+b}\right)^a \cdot \left(\frac{b}{a+b}\right)^b.$$

The Krichevski-Trofimov estimator has a remarkable constant redundancy, only at the edges it increases a little. It is likely that in the model which the context tree weighting algorithm approximates, most leaves will have a symbol probability near 0 or 1 [19]. From figure 7.22 it is clear that the choice $\alpha = \frac{1}{2}$ does not have to be the best. In order to determine the optimum α, some measurements were taken. Figure 7.23 shows us that the best choice would seem to be $\alpha = 1$. However it hasn't got a big influence on the compression. Here, the measurements were again compared to the Lena 3rd order CTW algorithm without division (vertical axis) for $\alpha = \frac{1}{16}$.
However, as stated earlier, the local divide will have some influence on the α and vice versa. So for the best local divide (at 48 and 64 bits), we re-measured with different α's.

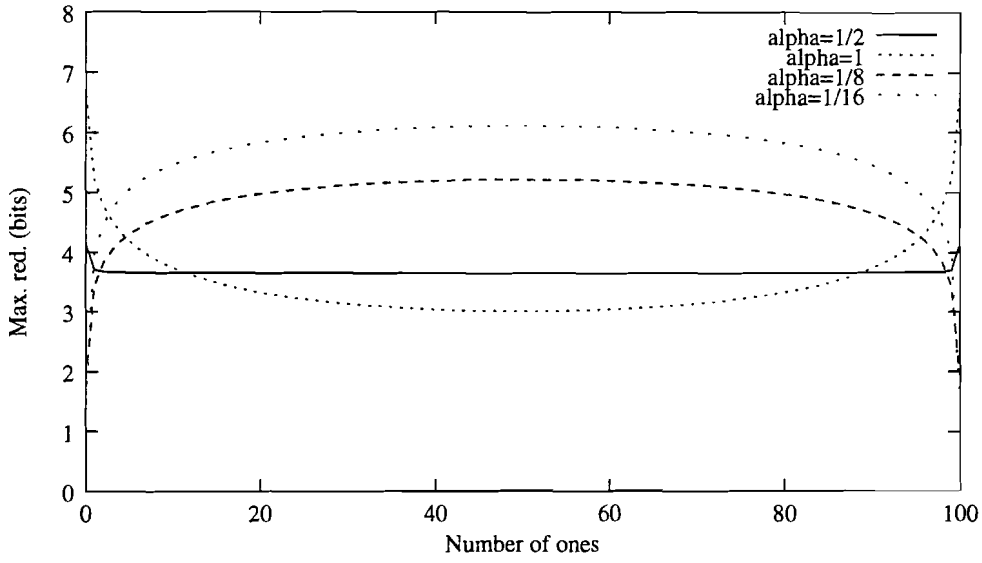

Figure 7.22: Estimators redundancy for different alpha's

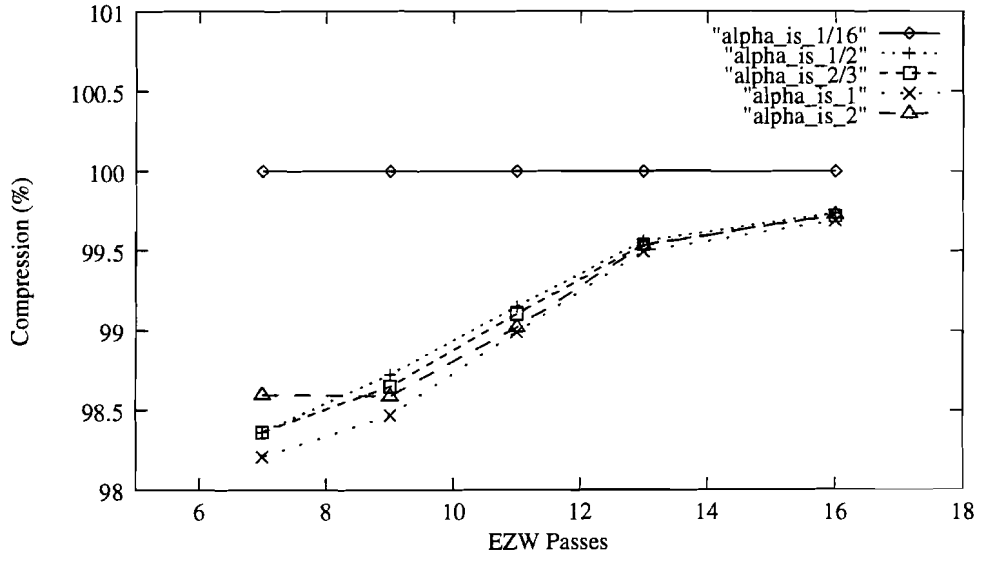

Figure 7.23: Different compression for different α's (Lena, 3rd order)

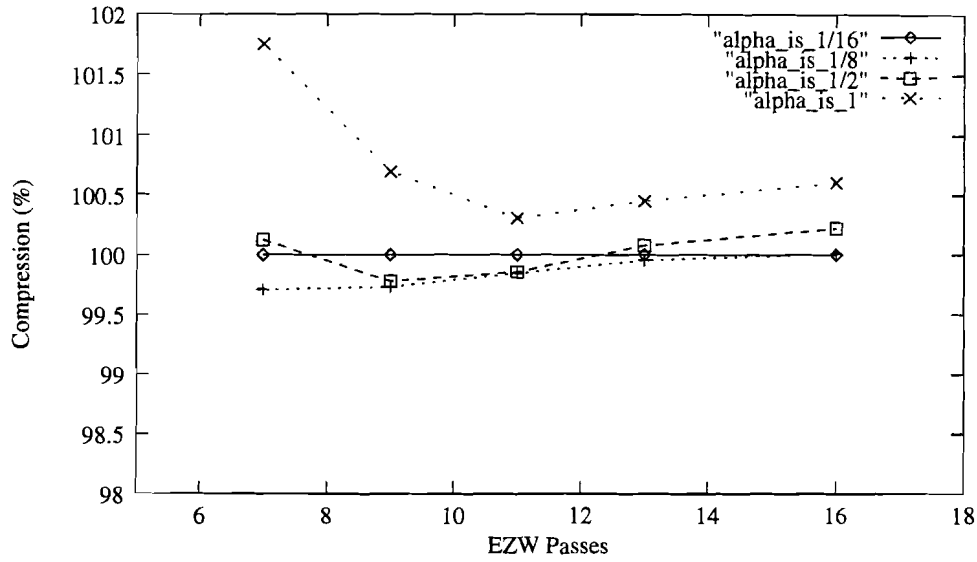

Figure 7.24: Different compression for different α's (Lena, 3rd order, 48 bits local)

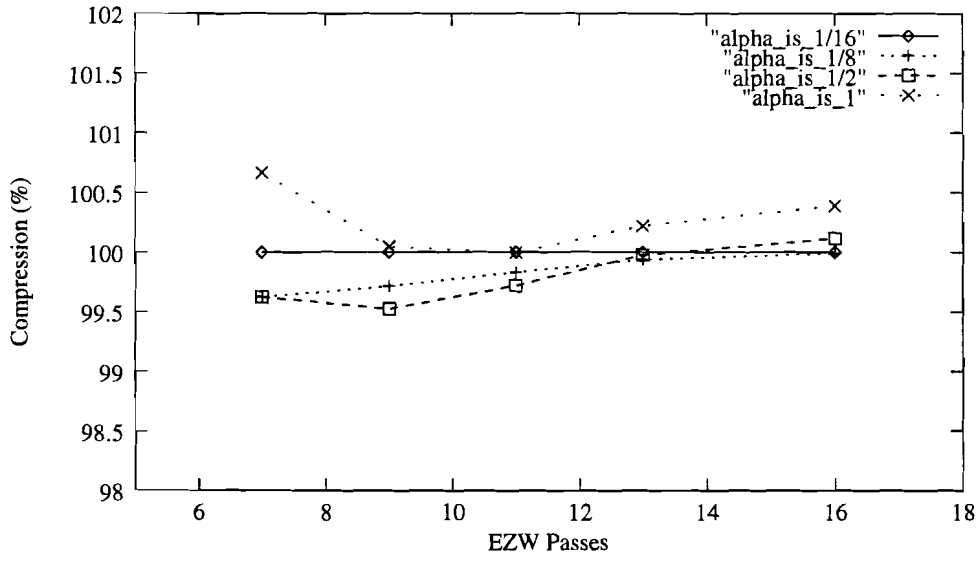

Figure 7.25: Different compression for different α's (Lena, 3rd order, 64 bits local)

Figure	Test image	Order	Local	Global	Division	Page
7.3	Lena	3	Variable	-	2	38
7.4	Lena	4	Variable	-	2	39
7.5	Lena	5	Variable	-	2	40
7.6	Barbara	3	Variable	-	2	40
7.7	Barbara	4	Variable	-	2	41
7.8	Barbara	5	Variable	-	2	41
7.9	Lena	3	-	Variable	2	42
7.10	Lena	4	-	Variable	2	42
7.11	Lena	5	-	Variable	2	43
7.12	Barbara	3	-	Variable	2	43
7.13	Barbara	4	-	Variable	2	44
7.14	Barbara	5	-	Variable	2	44
7.15	Lena	3	-	1 Reset/Pass	Variable	45
7.16	Barbara	3	-	1 Reset/Pass	Variable	45
7.17	Lena	3	Best	Best	2	46
7.18	Lena	3	Variable	1 Reset/Pass	2 (local)	47
7.19	Lena	3	Variable	-	4	48
7.20	Lena	4	Variable	-	4	48
7.21	Lena	3	Variable	-	16	49

Table 7.1: Table of measurement-figures in section 7.3

From figure 7.24 and 7.25 one can conclude that the influence of the α can lead to a better (all be it a small) improvement with $\alpha = \frac{1}{2}$ or $\frac{1}{8}$.

Besides the tuning of the parameter α one can also use the so called Zero-redundancy estimator [17]. This estimator has a very small redundancy for probabilities near 0 and 1. However, concluding that for images the probabilities are not near 0 or 1 (then a smaller α would be better), the zero-redundancy estimator was not taken into account.

7.5 Subordinate Pass

As found in section 6.2, for the subordinate pass, the optimum decision tree consisted of a single root. This means that no single context symbol could influence the compression. Indeed, the information from the subordinate pass (the bits needed to refine the value of the coefficients), has little to do with the conditions if a neighbour is ZeroTreeRoot or not. Therefor, the context used to compress the subordinate data is the last 10 bits from the subordinate pass.
For the subordinate data, a local and global divide was experimented with. This gave the result that a local divide at 64-96 bits and a global divide at each pass is the optimum. The α in the estimator (equation 7.1) did not have a significant influence on the compression.

The CTW algorithm does compress the data stream better. See figure 7.26. As one can see, the CTW-algorithm is (much) better in the beginning of the stream. It converges much

faster.

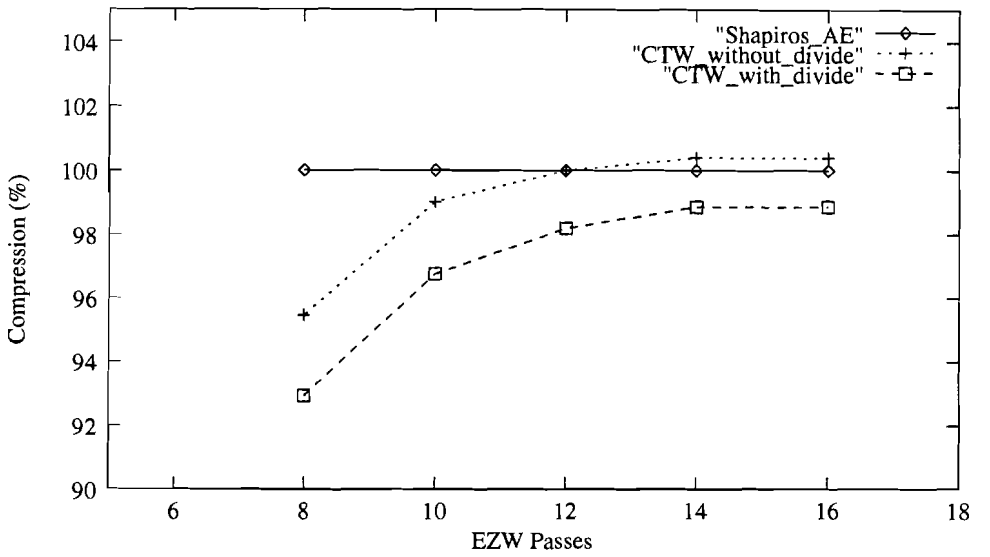

Figure 7.26: CTW compared to a general AE [25] for subordinate data

7.6 Which contexts to use?

According to the measurements in section 6.2 the context which should give the best compression is (from left to right with decreasing importance):

```
 P     W       N      NW    NE
1 2   3 4     5 6    7 8   9 10
```

Compared with the second context used in section 6.2.2 and the reversed order of above specified context, the best compression is reached for the context which was given by the MDL-algorithm. See figure 7.27.
Which context is used has its impact on the local and global dividing. Some quick measurements proved that in this case the difference for dividing with the best context compared to the context used in section 7.3 is neglectable.

Because the MDL takes a long time to calculate the best order, and for different orders there are different trees, it is difficult to draw a conclusion from the MDL-trees. Nevertheless, some more measurements were done with different contexts using the CTW algorithm, and it proved to be that the context above gives indeed the best compression.

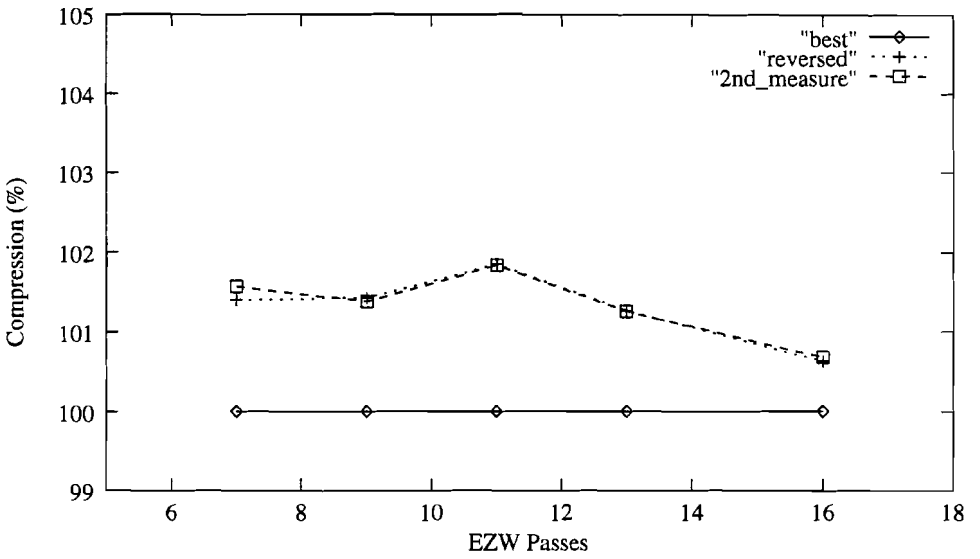

Figure 7.27: Different compressions for different contexts

Chapter 8

Conclusions

With all the parameters optimal as found in the previous chapter, the compression was measured for Lena and Barbara. These are plotted in figure 8.1.

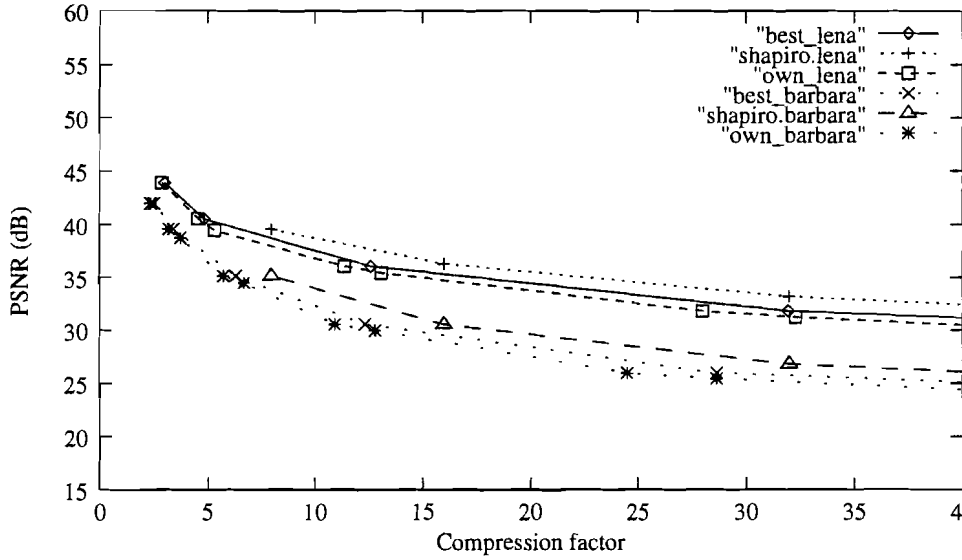

Figure 8.1: Maximum compression for Lena and Barbara

As one can see from figure 8.1, the compression gained with the CTW algorithm is just below the compression that Shapiro achieved.

Why doesn't perform the CTW algorithm any better? There are several reasons for this.
First, data coming from the subordinate pass, is almost random. Especially when the threshold is low. This is because these bits have a minor influence on the value of the coefficient. Hence these bits are more or less random. Of course the CTW cannot compress this any better than a general arithmetic encoder.
Second and this is probably the main reason, the EZW method does not have to be the best supplier of the data for the CTW weighting methods. It is difficult to give a better context with the information EZW supplies. The only information the decoder also has is whether a coefficient is a ZeroTreeRoot, a SIGnificant, an ISOlated zero or a ZeroTreeRoot. That

information does not seem to be enough.
In [6] the values of the coefficients were used as context. This gives a better result compared to a general arithmetic encoder. When using this method however, one loses the ability to stop the data stream and have a complete picture (embedded image coding). Also in [6] the wavelet energy compaction ability is not considered. This works in favour of the CTW-algorithm, because then, the CTW has more and more accurate data.
Third, because our implementation of the EZW does not reach the compression/quality of Shapiro's, it could be that the little compression increase the CTW algorithm gains, could be better with the real EZW implementation. After all, if our EZW implementation is different than Shapiro's, other data will be given to the CTW algorithm. However, it may also be the case that the wavelet transform is different, which will have little influence on the compression the CTW algorithm achieves, but it'll give a better quality for the same compression.
Finally, the CTW algorithm cannot improve the quality. It only can compact the information better. So in figure 8.1, the line can only move horizontally by the CTW-algorithm. Looking only at the difference in compression between our own EZW and the EZW with the CTW algorithms, we can conclude that the CTW indeed does compress significantly better. Hence, there is indeed more context information which one can use than the context used by the Zerotree.

Bibliography

[1] E.H. Adelson and E. Simoncelli. Orthogonal pyramid transforms for image coding. *SPIE, Visual Communications and Image Processing II*, 845:50–58, 1887.

[2] M. Antonini, M Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, April 1992.

[3] H.J. Barnard. *Image and Video Coding Using a Wavelet Decomposition.* PhD thesis, Delft University of Technology, Delft, Netherlands, 1994.

[4] Calderbank, Daubechies, Sweldens, and Yeo. Wavelet transforms that map integers to integers. http://www.ee.princeton.edu/yeo/, http://cm.bell-labs.com/who/wim/, august 1996.

[5] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45:485–560, 1988.

[6] N. Ekstrand. Some results on lossless compression of grayscale images. Technical report, Lund University, Department of Information Technology, 1998.

[7] R.E. Krichevski and V.K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207, 1981.

[8] P.S. Laplace. Memoire sur la probabilite de causes par les evenemens. *Memoires de l'Academie Royale del Sciences*, (6):612–656, 1774. Reprinted in *Laplace Complete Work*, vol 8.

[9] P.S. Laplace. Memoire sur les approximations des formulas qui sont functions de tres grands nombres et sur leur application aux probabilities. *Memoires de l'Academie des Sciences de Paris*, 1810. Reprinted in *Laplace Complete Work*, vol 12.

[10] S.G. Mallat. A theory for multiresolution signal decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[11] D. Marr. *Vision.* Freeman, San Francisco, 1982.

[12] W.B. Pennebaker and J.L. Mitchell. *JPEG, still image data compression standard.* Van Nostrand Reinhold, 1993.

[13] J. Provine and R. Rangayyan. Lossless compression of peanoscanned images. *Journal of Electric Imaging*, 3(2):176–181, April 1994.

[14] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[15] Smith and Barnwell III. Exact reconstruction of tree-structured sub band coders. *IEEE transactions on Acoustics, Speech and Signal Processing*, ASSP-34(3):434–441, June 1986.

[16] W. Sweldens and P. Schröder. Building your own wavelets at home. http://cm.bell-labs.com/who/wim/, 1996.

[17] Tj. J. Tjalkens, F.M.J. Willems, and Y. M. Shtarkov. Multi-alphabet universal coding using a binary decomposition context tree weighting algorithm. *15^{th} Symposium on Information Theory in the Benelux*, pages 259–265, May 1994.

[18] P.A.J. Volf. Deriving mdl-decision trees using the context maximising algorithm. Master's thesis, Eindhoven University of Technology, April 1994.

[19] P.A.J. Volf. Text compression methods based on context weighting. Technical report, Stan Ackermans Instituut, June 1996.

[20] M.J. Weinberger, G. Seroussi, and G. Sapiro. LOCO-I: A low complexity, context-based, lossless image compression algorithm. In *IEEE Data Compression Conference*, March-April 1996.

[21] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. Context tree weighting: A sequential universal source coding procedure for fsmx sources. *IEEE International Symposium on Information Theory*, page 59, January 1993.

[22] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. Context tree weighting: Redundancy bounds and optimality. *6^{th} Swedish-Russian Workshop on Information Theory*, August 1993.

[23] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. Context weighting: General finite context sources. *14^{th} Symposium on Information Theory in the Benelux*, pages 120–127, May 1993.

[24] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. Context tree weighting: Basic properties. *IEEE Transactions on Information Theory*, 41(3):653, 1995.

[25] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.