Acta Cybernetica **14** (1999) 1-.

Limiting Distortion of a Wavelet Image Codec

Joonas Lehtinen *

Abstract

A new image compression algorithm, *Distortion Limited Wavelet Image Codec* (DLWIC), *is introduced*. The codec is designed to be *simple to implement*, *fast* and have *modest requirements for the working storage*. It is shown, *how the distortion of the result can be calculated while progressively coding a transformed image* and thus how the *mean square error of the result can be limited* to a predefined value. The DLWIC uses zerotrees for efficient coding of the wavelet coefficients. Correlations between different orientation components are also taken into account by binding together the coefficients on the three different orientation components in the same spatial location. The maximum numbers of significant bits in the coefficients of all subtrees are stored in two-dimensional heap structure that allows the coder to test the zerotree property of a subtree with only one comparison. The compression performance of the DLWIC is compared to the industry standard JPEG compression and to an advanced wavelet image compression algorithm, vqSPIHT. An estimation of execution speed and memory requirements for the algorithm is given. The compression performance of the algorithm seems to exceed the performance of the JPEG and to be comparable with the vqSPIHT.

1 Introduction

In some digital image archiving and transferring applications, especially in medical imaging, the quality of images must meet predefined constrains. The quality must be often guaranteed by using a lossless image compression technique. This is somewhat problematic, because the compression performance of the best known lossless image compression algorithms is fairly modest; the compression ratio ranges typically from 1:2 to 1:4 for medical images [5].

Lossy compression techniques generally offer much higher compression ratios than lossless ones, but this is achieved by losing details and thus decreasing the quality of the reconstructed image. Compression performance and also the amount of distortion are usually controlled with some parameters which are not directly connected to image quality, defined by *mean square error* [1] (MSE). If a lossy technique is used, the quality constrains can be often met by overestimating the control parameters, which results worse compression performance.

*Turku Centre for Computer Science, University of Turku, Lemminkäisenkatu 14 A, 20520 Turku, Finland, email: jole@jole.fi, WWW: http://jole.fi/

In this paper a new lossy image compression technique called *Distortion Limited Wavelet Image Codec* (*DLWIC*) is presented. The DLWIC is related to *embedded zerotree wavelet coding* (EZW) [9] technique introduced by J.M. Shapiro in 1993. Also some ideas from SPIHT [8] and vqSPIHT [3] have been used. DLWIC solves the problem of *distortion limiting* (DL) by allowing the user of the algorithm to specify the *MSE of the decompressed image as controlling parameter for the compression algorithm.*

The algorithm is designed to be *as simple as possible*, which is achieved by *binding together the orientation bands of the octave band composition* and *coding the zerotree structures and wavelet coefficient bits in the same pass.* A special auxiliary data structure called *two dimensional heap is introduced* to make the zerotree coding simple and fast. The DLWIC uses only little extra memory in the compression and is thus suitable for compression of very large images. The technique also seems to provide competitive compression performance in comparison with the vqSPIHT.

In the DLWIC, the image to be compressed is first converted to the wavelet domain with the orthonormal *Daubechies wavelet transform* [10]. The transformed data is then coded by bit-levels using a scanning algorithm presented in this paper. The output of the scanning algorithm is coded using *QM-coder* [7], an advanced binary arithmetic coder.

The scanning algorithm processes the bits of the wavelet transformed image data in decreasing order of their significance in terms of MSE, as in the EZW. This produces *progressive* output stream: the algorithm can be stopped at any phase of the coding and the already coded output can be used to construct an approximation of the original image. This feature can be used when a user browses images using slow connection to the image archive: The image can be viewed immediately after only few bits have been received; the subsequent bits then make it more accurate. The DLWIC uses the progressivity by stopping the coding when the quality of the reconstruction exceeds threshold given as a parameter to the algorithm. The coding can also be stopped when the size of the coded output exceeds a given threshold. This way both the MSE and *bits per pixel* (BPP) value of the output can be accurately controlled.

After the introduction, the structure of the DLWIC is explained. A quick overview of the octave band composition is given and it is shown with an example how the wavelet coefficients are connected to each other in different parts of the coefficient matrix.

Some general ideas of the bit-level coding are then explained (2.3) and it is shown how the unknown bits should be approximated in the decoder. The meaning of zerotrees in DLWIC is then discussed (2.4). After that an auxiliary data structure called two dimensional heap is introduced (2.5). The scanning algorithm is given as pseudo code (2.6).

The distortion limiting feature is introduced and the stopping of the algorithm on certain stopping conditions is discussed (2.7). Finally we show how separate probability distributions are allocated for coding the bits with the QM-coder in different contexts (2.8).

The algorithm is tested with a set of images and the compression performance

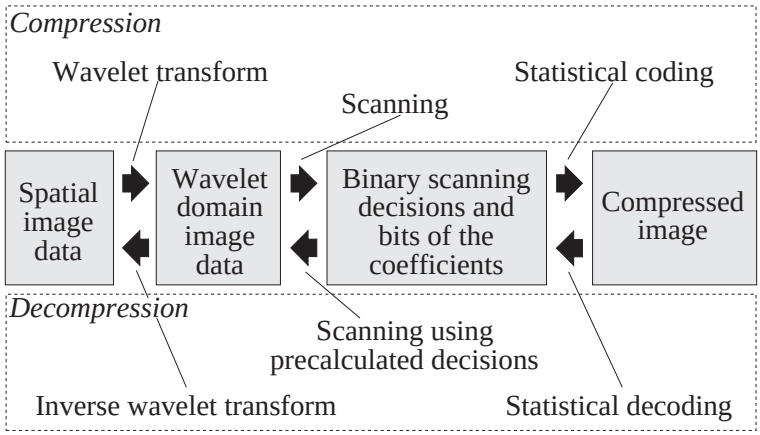

Figure 1: The structure of the DLWIC compression algorithm

is compared to the JPEG and the vqSPIHT compression algorithms (3). Variations in the quality achieved by the constant quantization in the JPEG is demonstrated with an example. Also an estimation of the speed and memory usage is given (3.2).

2 DLWIC algorithm

2.1 Structure of the DLWIC and the wavelet transform

The DLWIC algorithm consists of three steps (Figure 1): 1) the wavelet transform, 2) scanning the wavelet coefficients by bit-levels and 3) coding the binary decisions made by the scanning algorithm and the bits of the coefficients with the statistical coder. The decoding algorithm is almost identical: 1) binary decisions and coefficient bits are decoded, 2) the coefficient data is generated using the same scanning algorithm as in the coding phase, but using the previously coded decision information, 3) the coefficient matrix is converted to a spatial image with the inverse wavelet transform.

The original spatial domain picture is transformed to the wavelet domain using Daubechies wavelet transform [10]. The transform is applied recursively to the rows and columns of the matrix representing the original spatial domain image. This operation gives us an octave band composition (Figure 2). The left side (B) of the resulting coefficient matrix contains horizontal components of the spatial domain image, the vertical components of the image are on the top (A) and the diagonal components are along the diagonal axis (C). Each *orientation pyramid* is divided to levels, for example the horizontal orientation pyramid (B) consists of three levels (B0, B1 and B2). Each level contains details of different size; the lowest level (B0), for example, contains the smallest horizontal details of the spatial image. The three orientation pyramids have one shared top level (S), which contains

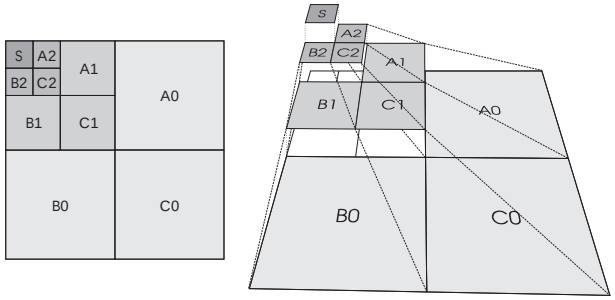

Figure 2: Octave band composition produced by recursive wavelet transform is illustrated on the left and the pyramid structure inside the coefficient matrix is shown on the right.

scaling coefficients of the image, representing essentially the average intensity of the corresponding region in the image. Usually the coefficients in the wavelet transform of a natural image are small on the lower levels and bigger on the upper levels (Figure 3). This property is very important for the compression: the coefficients of this highly skewed distribution can be coded using fewer bits.

2.2 Connection between orientation pyramids

Each level in the coefficient matrix represents certain property of the spatial domain image in its different locations. Structures in the natural image contain almost always both big and small details. In the coefficient matrix this means that if some coefficient is small, it is most likely that also the coefficients, representing smaller features of the same spatial location, are small. This can be seen in Figure 3: different levels of the same coefficient pyramid look similar, but are in different scales. The EZW takes advantage of this by scanning the image in depth first order, i.e. it scans all the coefficients related to one spatial location in one orientation pyramid before moving to another location. This way it can code a group of small coefficients together, and thus achieves better compression performance.

In natural image, most of the features are not strictly horizontal or vertical, but contain both components. The *DLWIC takes advantage of this by binding also all three orientation pyramids together*: The scanning is done only for the horizontal orientation pyramid (B), but bits of all three coefficients, representing the three orientations of the same location and scale, are coded together. Surprisingly this only slightly enhances the compression performance. The feature is however included in the DLWIC because of its advantages: it simplifies the scanning, makes the implementation faster and reduces the size of auxiliary data structures.

Figure 3: An example of the Daubechies wavelet transform. The original 512×512 sized picture is on the left and its transform is presented with absolute values of the coefficients in logarithmic scale on the right.

2.3 Bit-level coding

The coefficient matrix of size $W \times H$ is scanned by bit-levels beginning from the highest bit-level n_{max} required for coding the biggest coefficient in the matrix (i.e. the number of the significant bits in the biggest coefficient):

$$n_{max} = \lfloor \log_2(\max\{(|c_{i,j}|)|0 \leq i < W \wedge 0 \leq j < H\}) + 1 \rfloor, \tag{1}$$

where the coefficient in (i, j) is marked with $c_{i,j}$. The coefficients are represented using positive integers and the sign bits that are stored separately. The coder first codes all the bits on the bit-level n_{max} of all the coefficients, then all the bits on bit-level $n_{max} - 1$ and so on until the least significant bit-level 1 is reached or the scanning algorithm is stopped (Section 2.7). The sign is coded together with the most significant bit (the first 1-bit) of a coefficient. For example three coefficients $c_{0,0} = -19_{10} = -10011_2, c_{1,0} = 9_{10} = 01001_2, c_{2,0} = -2_{10} = -00010_2$ would be coded as

$$\underbrace{1}_{5}\underbrace{1}_{4}\underbrace{00}_{3}\underbrace{0}_{4}\underbrace{1}_{2}\underbrace{00}_{3}\underbrace{1}_{2}\underbrace{011}_{1}\underbrace{1}_{2}\underbrace{10}_{1}, \tag{2}$$

where the corresponding bit-level numbers are marked under the bits coded on that level (without signs it would be $\underbrace{100}_{5}\underbrace{010}_{4}\underbrace{000}_{3}\underbrace{101}_{2}\underbrace{110}_{1}$).

Because of the progressivity, the code stream can be truncated at any position and the decoder can approximate the coefficient matrix using received information. The easiest way of approximating the unknown bits in the coefficient matrix would be to fill them with zeroes. In the DLWIC algorithm a more accurate estimation is

used, the first unknown bit of each coefficient, for which the sign is known, is filled with one and the rest bits are filled with zeroes. For example, if the first seven bits of the bit-stream (2) have been received, the coefficients would be approximated: $c_{0,0} = -20_{10} = -10100_2, c_{1,0} = 12_{10} = 01100_2, c_{2,0} = 0_{10} = 00000_2$.

2.4 Zerotrees in DLWIC

A bit-level is scanned by first coding a bit of a scaling coefficient (on the level S in the Figure 2). Then recursively the three bits of the coefficients in the same spatial location on the next level of the orientation pyramids (A2,B2,C2) are coded. The scanning continues to the next scaling coefficient, after all the coefficients in the previous spatial location in all the pyramid levels has been scanned.

We will define that a coefficient c is *insignificant* on a bit-level n, if and only if $|c| < 2^{n-1}$. Because the coefficients on the lower pyramid levels tend to be smaller than on the higher levels and different sized details are often spatially clustered, probability for a coefficient for being insignificant is high, if the coefficient on the higher level in the same spatial location is insignificant.

If an insignificant coefficient is found in the scanning, the compression algorithm will check if any of the coefficients below the insignificant one is significant. If no significant coefficients are found, all the bits in those coefficients on current bit-level are zeroes and thus can be coded with only one bit. This structure is called *zerotree*.

One difference to the EZW algorithm is that the DLWIC scans all the orientations simultaneously and thus constructs only one shared zerotree for the all the orientation pyramids. Also the significance information is coded at the same pass as the significant bits in the coefficients, whereas the EZW and SPIHT algorithms use separate passes for the significance information.

2.5 Two dimensional significance heap

It is a slow operation to perform a significance check for all the coefficients on a specific spatial location on all the pyramid levels. The DLWIC algorithm uses a new auxiliary data-structure, which we call *two dimensional significance heap*, to eliminate the slow significance checks.

The heap is a two dimensional data-structure of the same size (number of elements) and shape as the horizontal orientation pyramid in the coefficient matrix. *Each element in the heap defines the number of bits needed to represent the largest coefficient in any orientation pyramid in the same location on the same level or below it.* Thus the scanning algorithm can find out, whether there is a zerotree starting from a particular coefficient on a certain bit-level by comparing the number of the bit-level to the corresponding value in the heap.

Here and in the rest of this paper we denote the height of the coefficient matrix with H, the width with W and the number of levels in the pyramid excluding the scaling coefficient level (S) with L. Thus the dimensions of the scaling coefficient level are: $H_s = H/2^L$ and $W_s = W/2^L$. Furthermore the dimensions of the level

in the two-dimensional heap, where (x, y) resides are

$$\begin{aligned} W_{x,y} &= W_s 2^{\lfloor \log_2(\max\{1, \frac{y}{H_s}\}) \rfloor} \\ H_{x,y} &= H_s 2^{\lfloor \log_2(\max\{1, \frac{y}{H_s}\}) \rfloor} \end{aligned} \tag{3}$$

Now the heap elements $h_{x,y}$ can be defined with the functions $h_t(x, y)$, $h_c(x, y)$ and $h_s(x, y)$:

$$\begin{aligned} h_t(x,y) &= \max\{h_{x,y+H_s}, \lfloor \log_2(|c_{x,y}|) \rfloor + 1\} \\ h_c(x,y) &= \max\{h_{2x,2y}, h_{2x+1,2y}, h_{2x,2y+1}, h_{2x+1,2y+1}\} \\ h_s(x,y) &= \lfloor \log_2(\max\{|c_{x,y}|, |c_{x+W_{x,y},y}|, |c_{x+W_{x,y},y-H_{x,y}}|\}) \rfloor + 1 \\ h_{x,y} &= \begin{cases} h_t(x,y), & \text{if } x < W_s \wedge y < H_s \\ h_s(x,y), & \text{if } x \geq W/2 \wedge y \geq H/2 \\ \max\{h_s(x,y), h_c(x,y)\}, & \text{otherwise,} \end{cases} . \end{aligned} \tag{4}$$

Note that the definitions (3) and (4) are only valid for the elements in the heap, where $0 \leq y < H$ and $0 \leq x < W_s 2^{\lfloor \log_2(\max\{1, \frac{y}{H_s}\}) \rfloor}$. While the definition of the heap looks complex, we can construct the heap with a very simple and fast algorithm (Alg. 1).

2.6 Coding algorithm

The skeleton of the compression algorithm (Alg. 2) is straightforward: 1) the spatial domain image is transformed to wavelet domain by constructing the octave band composition, 2) the two dimensional heap is constructed (Alg. 1), 3) the QM-coder is initialized, 4) the coefficient matrix is scanned in bit-levels by executing scanning algorithm (Alg. 3) for each top level coefficient on each bit-level.

The decoding algorithm is similar. First an empty two dimensional heap is created by filling it with zeroes. Then the QM-decoder is initialized and the same scanning algorithm is executed in such way that instead of calculating the decisions, it extracts the decision information from the coded data.

The scanning algorithm (Alg. 3) is the core of the compression scheme. It tries to minimize correlations between the saved bits by coding as many bits as possible with zerotrees. In the pseudo-code, Bit(x,n) returns n:th bit of the absolute value of x and $s_{i,j}$ denotes the sign of the coefficient in the matrix element (i, j). Bits are coded with function QMCode(b,CONTEXT), where b is the bit to be coded and CONTEXT is the context used as explained in Section 2.8. Context can be either a constant or some function of variables known to both the coder and decoder. In both cases, the value of the context is not important, but it should be unique for each combination of parameters. Stopping of the algorithm is queried with function ContinueCoding(), which returns true, if the coding should be continued. In order to calculate the stopping condition, the quality of the approximated resulting image must be calculated while coding. This is achieved by calling function DLUpdate(n,x) every time after coding n:th bit of the coefficient x. Both calculations are explained in the Section 2.7. The dimensions of the matrix and its levels are noted in the same way as in the Section 2.5.

The scanning algorithm first checks the stopping condition. Then we check from the two dimensional heap, whether there is a zerotree starting from this location, and code the result. If the coefficient had become significant earlier, the decoder knows also that and thus we can omit the coding of the result. If we are coding a scaling coefficient ($l = 0$), we only process that coefficient and then recursively scan the coefficient in the same location on the next level below this one. If we are coding a coefficient below the top-level, we must process all three coefficients in three orientation pyramids in this spatial location and then recursively scan all four coefficients on the next level, if that level exists.

When a coefficient is processed using ScanCoeff algorithm (Alg. 4), we first check, whether it had become significant earlier. If that is the case, we just code the bit on the current bit-level and then do the distortion calculation. If the coefficient is smaller than 2^n, we code the bit on the current bit-level, and also check whether that was the first 1-bit of the coefficient. If that is true, we also code the sign of the coefficient and do the distortion calculation.

2.7 Stopping condition and distortion limiting

The DLWIC continues coding until some of the following conditions occur: 1) All the bits of the coefficient matrix have been coded, 2) The number of bits produced by the QM-coder reach a user specified threshold, or 3) the distortion of the output image, that can be constructed from sent data, decreases below the user specified threshold. The binary stopping decisions made before coding each bit of a coefficient are coded, as the decoder must exactly know when to stop decoding.

The first condition is trivial, as the main loop (Alg. 2) ends when all the bits have been coded. The second condition is also easy to implement: output routine of the QM-coder can easily count the number of bits or bytes produced. To check the third condition, the algorithm must know the MSE of the decompressed image. The MSE of the decompressed image could be calculated by doing inverse wavelet transform for the whole coefficient matrix and then calculating the MSE from the result. Unfortunately this would be extremely slow, because the algorithm must check the stopping condition very often.

The reason for using Daubechies wavelet transform is its orthonormality. For orthonormal transforms, the square sums of the pixel values of the image before and after the transform are equal:

$$\sum_{i,j}(x_{i,j})^2 = \sum_{i,j}(c_{i,j})^2 \tag{5}$$

where $x_{i,j}$ stands for the spatial domain image intensity and $c_{i,j}$ is the wavelet coefficient. Furthermore, the mean square error between the original image and some approximation of it can be calculated equally in the wavelet and spatial domains. Thus we do not have to do the inverse wavelet transform to calculate the MSE.

Instead of tracking the MSE, we track the current square error, *cse*, of the approximated image because it is computationally easier. The initial approximation

of the image is zero coefficient matrix, as we have to approximate the coefficients to be zero, when we do not know their signs. Thus the initial *cse* equals to the energy of the coefficient matrix.

$$cse \leftarrow \sum_{i,j} (c_{i,j})^2 \tag{6}$$

After sending each bit of a coefficient c, we must update *cse* by subtracting the error produced by the previous approximation of c and adding the error of its new approximation. The error of an approximation of c depends only on the level of the last known bit and the coefficient c itself. If we code the n:th bit of c, then the *cse* should be updated:

$$cse \leftarrow cse - \begin{cases} \left[(|c|\mathrm{AND}_2(2^n - 1)) - 2^{n-1}\right]^2 - & \\ \left[(|c|\mathrm{AND}_2(2^{n-1} - 1)) - 2^{n-2}\right]^2 & \text{if} \lfloor \log_2 c \rfloor > n-1 \\ c^2 - (2^{(n-1)} + 2^{(n-2)} - c)^2 & \text{if} \lfloor \log_2 c \rfloor = n-1 \\ 0 & \text{if} \lfloor \log_2 c \rfloor < n-1, \end{cases} \tag{7}$$

where AND_2 is bitwise and-operation. The first case defines the error reduced by finding out one bit of a coefficient, when the sign is already known. The second case defines the error reduced by finding out the sign of a coefficient and the last case states that *cse* does not change if only zero bit before the coefficients first one bit is found. The equation 7 holds only, when $n > 1$.

2.8 The use of contexts in QM-coder

The QM-coder is a binary arithmetic coding algorithm that tries to code binary data following some probability distribution as efficiently as possible. Theoretically an arithmetic coder compresses data according to its entropy [4], but the QM-coder uses a dynamical probability estimation technique [6, 7, 2] based on state automata, and its compression performance can even exceed the entropy, if the local probability distribution differs from the global distribution used in the entropy calculation.

The DLWIC codes different types of information with differing probability distributions. For example the signs of coefficients are highly random, that is the probability of plus sign is approximately 0.5, but the probability of finding the stopping condition is only $1/N$, where N is the number of stopping condition evaluations. If bits following the both distributions would be coded using the same probability distribution, the compression performance obviously would not be acceptable.

To achieve better compression performance the DLWIC uses separate *contexts* for binary data following different probability distributions. The contexts for coding the following type of data are defined: 1) signs, 2) stopping conditions, 3) the bits of the coefficients after the first one bit, 4) the bits of the scaling coefficients, 5) zerotrees on the different levels of the pyramid, 6) the significance check of the

Figure 4: Test images from top left: 1) barb (512×512), 2) bird (256×256), 3) boat (512×512), 4) bridge (256×256), 5) camera (256×256), 6) circles (256×256), 7) crosses (256×256), 8) france (672×496) and 9) frog (621×498).

insignificant coefficients on different pyramid levels on different orientation pyramids. The number of separate contexts is $4 * (l + 1)$, where l defines the number of levels in the pyramids. It would also be possible to define different contexts for each bit-level, but dynamical probability estimation in the QM-coder seems to be so efficient that this is not necessary.

3 Test results

The performance of the DLWIC algorithm is compared to the JPEG and the vqSPIHT [3] algorithms with a set (Fig. 4) of 8 bit grayscale test images. The vqSPIHT is an efficient implementation of the SPIHT [8] compression algorithm. The vqSPIHT algorithm uses the biorthogonal B97 wavelet transform [10], the QM-coder and a more complicated image scanning algorithm than the DLWIC. Image quality is measured in terms of *peak signal to noise ratio* [1] (PSNR), which is an inverse logarithmic measure calculated from MSE.

3.1 Compression efficiency

To compare the compression performance of the algorithms, the test image set is compressed with different BPP-rates from 0.1 to 3.0 and the PSNR is calculated as the mean for all the images. Because it is not possible to specify BPP as a parameter for JPEG compression algorithm, various quantization parameters are used and the BPP value is calculated as a mean value of the image set for each quantization value.

As can be seen in the Figure 5, the performance of the vqSPIHT and the DLWIC algorithms is highly similar. This is somewhat surprising, because of the greater complexity and better wavelet transform used in the vqSPIHT. The quality of the images compressed with the EZW variants seem to exceed the quality produced by

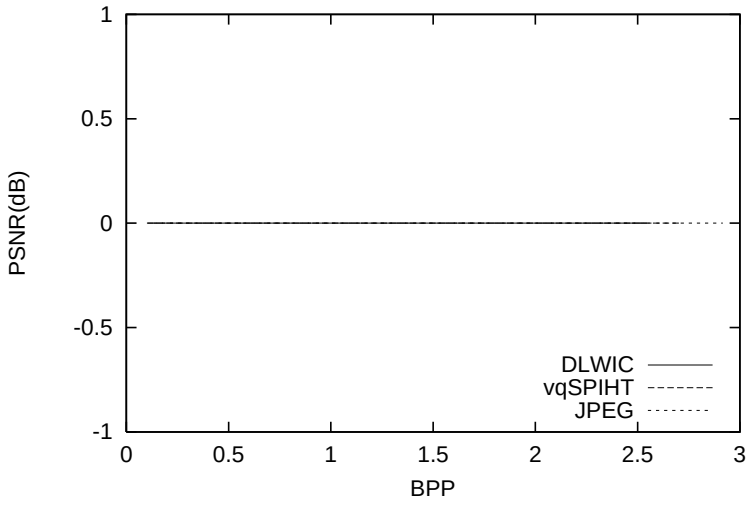

Figure 5: Compression performance comparison of DLWIC, vqSPIHT and JPEG. The PSNR-values correspond to mean value obtained from test image set (Fig. 4).

the JPEG. This is especially true when low bit-rates are used. Poor scalability of the JPEG to low BPP values is probably implied by the fixed block size used in the DCT transform of the JPEG as opposed to multi-resolution approach of the wavelet based methods.

One might expect that a conventional image compression algorithm such as the JPEG would give similar PSNR and BPP values for similar images when fixed quantization parameter is used. This is not the case as demonstrated in the Figure 6, where all the test images are compressed using the same quantization parameter (20) with the standard JPEG.

3.2 Speed and memory usage

The speed of the implementation is not compared to other techniques, because the implementation of the algorithm is not highly optimized. Instead an example of the time consumption of the different components of the compression process is examined using the GNU profiler. The frog test image is compressed using a 400MHz Intel Pentium II workstation running Linux and the algorithm is implemented in C language and compiled with GNU C 2.7.2.1 using "-O4 -p" options. The cumulative CPU time used in the different parts of the algorithm is shown in the Figure 7.

When the image is compressed with a low BPP-rate, most of the time is consumed by the wavelet transform. When the BPP-rate increases, the time used by the QM-coder, the scanning algorithm and the distortion calculations increases in

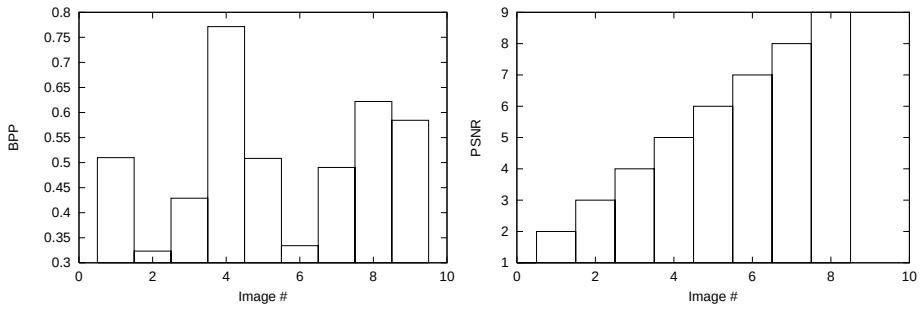

Figure 6: All the images of the test image set are compressed by JPEG with the same quantization value (20), and the BPP (left) and the PSNR (right) of the resulting images are shown.

somewhat linear manner. Construction of the two dimensional heap seems to be quite fast operation and distortion limiting is not very time consuming.

If we want to optimize the implementation of the DLWIC, the biggest problem would probably be the extensive use of the QM-coder, that is already highly optimized. One way to alleviate the problem would be to store the stopping condition is some other way than compressing the binary decision after each bit received. Also the transform would have to be optimized to achieve faster compression, because it consumes nearly half of the processing time, when higher compression ratios are used.

Probably the biggest advantage of the DLWIC over the SPIHT and even the vqSPIHT is its low auxiliary memory usage. The only auxiliary data-structure used, the two dimensional heap, can be represented using 8-bit integers and thus only consumes approximately $8 * N/3$ bits of memory, where N is the number of coefficients. If the coefficients are stored with 32-bit integers, this implies 8% auxiliary memory overhead, which is very reasonable, when compared to 32% overhead in the vqSPIHT or even much higher overhead in the SPIHT algorithm, which depends on the target BPP-rate.

4 Summary and conclusion

In this paper a new general purpose wavelet image compression scheme, DLWIC, was introduced. Also it was shown how the distortion of the resulting decompressed image can be calculated while compressing the image and thus how the distortion of the compressed image can be limited. The scanning algorithm in the DLWIC is very simple and it was shown, how it can be efficiently implemented using a two dimensional heap structure.

Compression performance of the DLWIC was tested with a set of images and the compression performance seems to be promising, when compared to a more complex

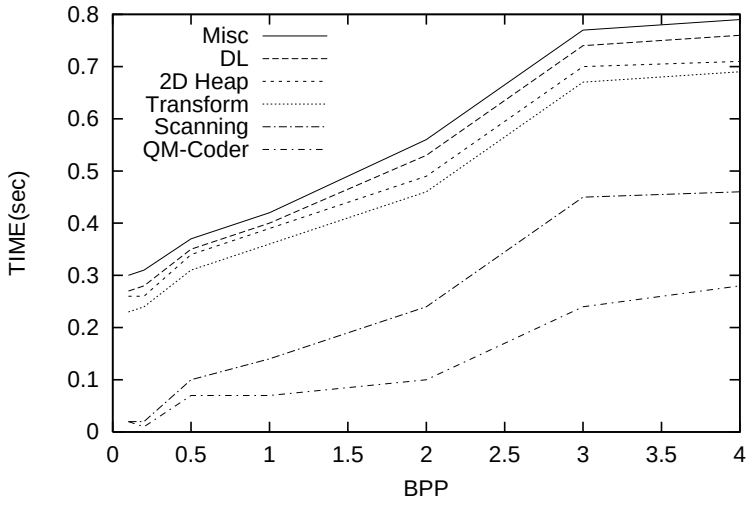

Figure 7: Running time of the different components in the DLWIC compression/decompression algorithm, when compressing the frog test image (Fig. 4). Graph shows the cumulative CPU time consumption when different BPP-rates are used.

compression algorithm, the vqSPIHT. Furthermore, the compression performance easily exceeds the performance of the JPEG, especially when high compression ratios are used.

Further research for extending the DLWIC algorithm to be used in lossless or nearly lossless multidimensional medical image compression is planned. Also the implementation of the DLWIC will be optimized and usage of some other wavelet transforms will be considered.

References

[1] R. Gonzalez and R. Woods. *Digital Image Processing.* Addison-Wesley Publishing Company, 1992.

[2] ITU-T. Progressive bi-level image compression, recommendation t.82. Technical report, International telecommunication union, 1993.

[3] A. Järvi, J. Lehtinen, and O. Nevalainen. Variable quality image compression system based on SPIHT. *to appear in Signal Processing: Image Communications*, 1998.

[4] Sayhood K. *Introduction to Data Compression.* Morgan Kaufmann, 1996.

[5] Juha Kivijrvi, Tiina Ojala, Timo Kaukoranta, Attila Kuba, László Nyúl, and Olli Nevalainen. The comparison of lossless compression methods in the case of a medical image database. Technical Report 171, Turku Centre for Computer Science, April 1998.

[6] W.B. Pennebaker and J.L. Mitchell. Probability estimation for the q-coder. *IBM Journal of Research and Development*, 32(6):737–752, 1988.

[7] William Pennebaker and Joan Mitchell. *Jpeg : Still Image Data Compression Standard*. Van Nostrand Reinhold, 1992.

[8] Amir Said and William A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, June 1996.

[9] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Processing*, 31(12), December 1993.

[10] M. Vettereli and J. Kovačević. *Wavelets and Subband Coding*. Prentice Hall, Englewood Cliffs, NJ, 1995.

Algorithm 1 Construct2DHeap

for $l \leftarrow 1$ **to** $L+1$ **do**
  $H_t \leftarrow H/2^{\min\{l,L\}}$, $W_t \leftarrow W/2^{\min\{l,L\}}$
  for $j \leftarrow 0$ **to** $H_t - 1$ **do**
    for $i \leftarrow 0$ **to** $W_t - 1$ **do**
      if $l = 1$ **then**
        $t \leftarrow 0$
        $u \leftarrow \lfloor 1 + \log_2(\max\{|c_{i,j+H}|, |c_{i+W,j}|, |c_{i+W,j+H}|\})\rfloor$
      else if $l \leq L$ **then**
        $t \leftarrow \max\{h_{2x,2y}, h_{2x+1,2y}, h_{2x,2y+1}, h_{2x+1,2y+1}\}$
        $u \leftarrow \lfloor 1 + \log_2(\max\{|c_{i,j+H}|, |c_{i+W,j}|, |c_{i+W,j+H}|\})\rfloor$
      else
        $t \leftarrow \max\{h_{i,j+H_s}, h_{i+W_s,j}, h_{i+W_s,j+H_s}\}$
        $u \leftarrow \lfloor 1 + \log_2(\max\{(|c_{i,j}|)|0 \leq i < W \wedge 0 \leq j < H\})\rfloor$
      $h_{i,j} \leftarrow \max\{t, u\}$

Algorithm 2 CompressDLWIC

Transform the spatial image with Daubechies wavelet transform constructing the octave band composition where the coefficients $c_{i,j}$ are represented with positive integers and separate sign bit.
Construct the two dimensional heap (Alg. 1).
Initialize QM-coder
Calculate initial distortion of the image (Section 2.7).
$n_{max} \leftarrow \max\{h_{i,j}|0 \leq i < W_s \wedge 0 \leq j < H_s\}$
for $n \leftarrow n_{max}$ **to** 1 **do**
  for $j \leftarrow 0$ **to** $H_s - 1$ **do**
    for $i \leftarrow 0$ **to** $W_s - 1$ **do**
      Scan$(i, j, 0, n)$ (Alg. 3)

Algorithm 3 Scan(i, j, l, n)

if ContinueCoding() **then**
  if $h_{i,j} < n$ **then**
    QMCode(INSIGNIFICANT, SIGNIFICANCE-TEST(l))
  else
    if $h_{i,j} = n$ **then**
      QMCode(SIGNIFICANT, SIGNIFICANCE-TEST(l))
    if $l = 0$ **then**
      ScanCoeff$(i, j$,TOPLEVEL$, n)$
      Scan$(i, j + H_s, 1, n)$
    else
      ScanCoeff$(i, j$,HORIZONTAL$\left(\begin{array}{l} l, c_{(i+W_{i,j}),j} < 2^n, \\ c_{(i+W_{i,j}),(j-Hi,j)} < 2^n \end{array} \right), n)$
      ScanCoeff$(i + W_{i,j}, j$,DIAGONAL$\left(\begin{array}{l} l, c_{i,j} < 2^{n-1}, \\ c_{(i+W_{i,j}),(j-Hi,j)} < 2^n \end{array} \right), n)$
      ScanCoeff$(i + W_{i,j}, j - H_{i,j}$,VERTICAL$\left(\begin{array}{l} l, c_{i,j} < 2^{n-1}, \\ c_{(i+W_{i,j}),j} < 2^{n-1} \end{array} \right), n)$
      if $2 * y < H$ **then**
        Scan$(2 * i, 2 * j, l + 1, n)$
        Scan$(2 * i + 1, 2 * j, l + 1, n)$
        Scan$(2 * i, 2 * j + 1, l + 1, n)$
        Scan$(2 * i + 1, 2 * j + 1q, l + 1, n)$

Algorithm 4 ScanCoeff$(x, y$,CONTEXT$, n)$

if $c_{x,y} < 2^n$ **then**
  QMCode(Bit$(c_{x,y}, n)$,CONTEXT)
  if Bit$(c_{x,y}, n) = 1$ **then**
    QMCode($s_{x,y}$,SIGN)
    DLUpdate$(n, c_{x,y})$
else
  QMCode(Bit$(c_{x,y}, n)$,COEFFICIENTBIT)
  DLUpdate$(n, c_{x,y})$