Chapter 5

Design of the Software Library WAILI

WAILI[1](Wavelets with Integer Lifting) is a software library — written in C++ —providing wavelet transforms and wavelet-based operations on two-dimensional images of various kinds [57]. We developed WAILI in the scope of the project "Wavelet-Based Interactive Video Communication and Image Database Consulting", with applications situated in image processing.

In this chapter we will discuss the goals that had to be met in WAILI. We will report about the design decisions that were taken to meet these goals as close as possible. Furthermore the internal structure of the library is explained.

WAILI is available in source form for research purposes [56]. Currently it is in use at ca. 250 research institutes.

5.1 Design Criteria

The project we designed WAILI for required the development of a software library for wavelet transforms and wavelet-based image processing operations. The library had to possess the following properties:

- support arbitrary image sizes,
- support images containing multiple channels, possibly each with a different size,

[1]It is written *WAILI*, but pronounced *WILY*, except in Dutch, where it's pronounced *PIEFPAK*.

- use an integer lossless wavelet transform, based on the lifting scheme,
- allow the transform of both the columns and rows of an image, or either the columns or the rows.
- allow to cut a rectangular subimage out of a larger transformed image, and to replace a rectangular area in a large wavelet-transformed image,
- allow noise reduction,
- allow simple compression, using thresholding of subbands,
- provide edge enhancement of images, by scaling the wavelet coefficients in a particular subband,
- provides some simple image operations like addition and subtraction.
- be object oriented (written in C++, because C++ offers the combination of flexibility, fast execution speeds and portability),

5.2 WAILI: Wavelets with Integer Lifting

5.2.1 The Lifting Scheme

We chose to implement two-dimensional tensor product wavelet transforms using the integer version of the lifting scheme.

Although the lifting scheme allows to transform signals with a finite length without extending the signal, we did not choose to take this approach. Instead we use the classical symmetric extension [4] because it is easier to implement and suffices for the applications we have in mind.

All arithmetic operations are done with 16 bit input data and 16 bit results. This is sufficient for applications where the input data is 8 bit wide. Of course this can easily be changed if necessary.

5.2.2 Cohen-Daubechies-Feauveau Biorthogonal Wavelets

The wavelets we use are a subclass of the Cohen-Daubechies-Feauveau family of biorthogonal wavelets [9]. We have chosen these because of their key benefits, which are:

- They have finite support. This preserves the locality of image features.
- The scaling function $\varphi(x)$ is always symmetric, and the wavelet function $\psi(x)$ is always symmetric or antisymmetric. This is important for image processing operations since the detection of features in an image does not depend on their left/right or up/down orientation.

- Its filter coefficients are dyadic, i.e. they are of the form $z/2^n$, with $z \in \mathbf{Z}$ and $n \in \mathbf{N}$. This simplifies the implementation. But unfortunately this feature is not always preserved by the decomposition in lifting steps.

We chose not to use wavelets with more than 6 vanishing moments to restrict the filter lengths. Longer filters have less locality and thus perform worse in image processing applications, in spite of their increase in smoothness.

WAILI provides the following wavelet transforms of this family. The notation $(n,\tilde{n})$ means that the primal wavelet has n vanishing moments, while the dual wavelet has $\tilde{n}$ vanishing moments:

(1, x): $(1,1)$, $(1,3)$, $(1,5)$

(2, x): $(2,2)$, $(2,4)$, $(2,6)$

(4, x): $(4,2)$, $(4,4)$, $(4,6)$

We deliberately did not implement any of the $(3,x)$ or $(5,x)$ wavelet transforms because their lifting steps require divisions by 3 or 5 — which are difficult to implement in hardware —, or because the normalization factor K_1 requires a division. $(6,x)$ are not implemented either because they require more than 16 bits (for 8 bit input data).

5.2.3 Image Size Constraints

Ideal signals have a length of 2^n samples. Only in that case we can apply the wavelet transform for n consecutive levels and guarantee that the low-pass and high-pass bands have the same size at each resolution level. Unfortunately we do not live in an ideal world, and real-world signals have arbitrary length. If the signal length is odd, or becomes odd at some resolution level, either the low-pass or high-pass band will have one more sample than the other. Which band will contain the most samples must be defined by some convention. A first solution is to standardize on the number of "even" samples (in lifting terminology) not being smaller than the number of "odd" samples. However, we did not take this approach. The reasons for this will become clear in Section 5.3.1, which discusses the implementation of crop and merge operations in WAILI.

Extended to two dimensions, ideal images are square with a width of 2^n pixels. In that case one can perform the same number of transform levels in both the horizontal and vertical directions. But real images are usually rectangular, and can have aspect ratios (width/height) that differ a lot from 1. This means the number of transform levels can be different for the columns and rows of the image. This adds even more complexity.

5.2.4 Images and Wavelet Transforms

Performing wavelet transforms on images involves several components (C++ classes).

Image Objects

An *Image* consists of one or more independent channels, thus allowing for different sizes and wavelet transform types per channel. No interpretation or format is imposed on the channels and its data. The actual meaning of the image data can be freely chosen by the user. Examples are grey scale, RGB, YUV or Lab color, etc.

Channel Objects

The basic building block of the library is the *Channel*. A channel is a rectangular matrix containing one-valued pixels. A channel can be non-transformed (an *NTChannel*), or wavelet-transformed (an *LChannel* — lifted channel).

Since a wavelet transform is a recursive transform, a LChannel contains some subchannels (subbands), which can be either non-transformed or wavelet-transformed. The number of subchannels in an LChannel depends on the type of wavelet transform. One can have the following combinations (all of these are derived from the Channel abstract base class):

LChannelCR Obtained by transforming both the columns and rows of an NTChannel. As a result, one has 4 subbands:

- **LL** Low-pass band in both the horizontal and the vertical direction,
- **LH** Low-pass band in the vertical direction, high-pass in the horizontal direction,
- **HL** High-pass band in the vertical direction, low-pass in the horizontal direction,
- **HH** High-pass band in both the horizontal and the vertical direction.

LChannelC Obtained by transforming only the columns of an NTChannel. As a result, one has 2 subbands:

- **L** Low-pass band in the vertical direction,
- **H** High-pass band in the vertical direction.

LChannelR Obtained by transforming only the rows of an NTChannel. As a result, one has 2 subbands:

- **L** Low-pass band in the horizontal direction,
- **H** High-pass band in the horizontal direction.

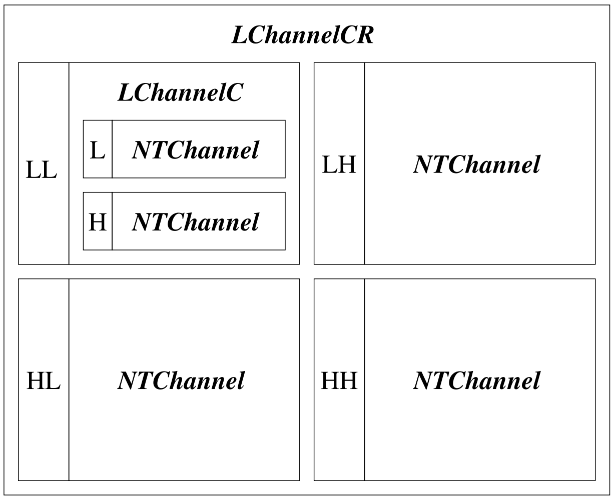

Figure 5.1: Example of a channel after two transform levels. In the first step both the columns and the rows are transformed, in the second step only the columns are transformed.

Figure 5.1 shows an example of a channel after two transform levels. And of course one can use a different wavelet at each transform level. A LChannel remembers the wavelet transform that was used to be able to perform the inverse transform later.

Wavelet Objects

A *Wavelet* represents the filters and lifting steps associated with a specific wavelet transform. Different wavelet transforms are implemented through inheritance. Some wavelet transforms of the Cohen-Daubechies-Feauveau family are implemented. One can add his own favorite wavelet transform if a decomposition in integer lifting steps for it is available, simply by deriving a new class from the Wavelet class that calls the lifting operations with the appropriate lifting coefficients.

Lifting Objects

A *Lifting* object performs lifting steps. The provided lifting operations are generic in the sense that one has to pass the lifting coefficients to be used. But due to the object oriented nature of C++ one can easily plug in highly optimized versions with hardcoded lifting coefficients for specific wavelet transforms to achieve maximum

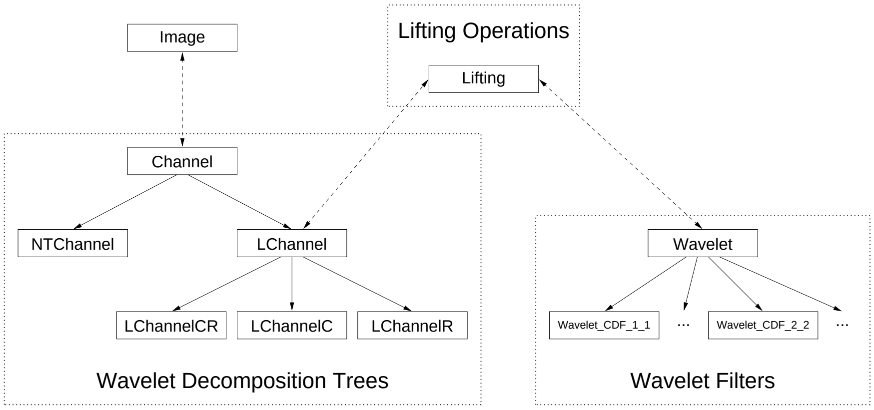

Figure 5.2: Relations among the major components of WAILI. Inheritance relations are indicated by solid arrows, computational relations are indicated by dashed arrows.

performance.

The relations among the major components of WAILI are shown in Figure 5.2.

5.3 Wavelet-Based Operations

Within WAILI, we implemented some image processing operations on wavelet-transformed images.

5.3.1 Crop and Merge on Wavelet-Transformed Images

The wavelet and scaling functions are local in the spatial and the frequency domain. If one wants to cut a small rectangle B out of a large image A, one can exploit the spatial locality of the basis functions.

At each transform level l, we only need the coefficients that influence the pixel values of B. This is a rectangle of coefficients that is only slightly larger than B (scaled to the transform level l). The extra coefficients are needed due to the overlap of the basis functions at the borders of B.

Using these technique we efficiently implemented the following operations on wavelet transformed images:

Crop Get a rectangular part of an image, at a specified position and with a specified size.

Merge Paste a rectangular image into a larger image, at a specified position.

The major point of difficulty here is that a wavelet transform is not translation invariant, due to the different treatment of even and odd samples in the lifting scheme (see Section 2.4). We came up with the following scheme. If each transform level is considered independently, one step of a wavelet transform is translation invariant if the translation is limited to an even number of pixels. Thus we associate with every image global *coordinates* (a horizontal and vertical offset for the upper left pixel), depending on the transform level. Hence we no longer always put the upper left pixel of the image in the even/even data set during the lazy wavelet transform step, but instead we partition all pixels in the image based on the parities of their global coordinates. This is equivalent to considering two (actually four because of the two-dimensional aspect) versions of the wavelet transform: an "even" and an "odd' version. This way we can merge two images without retransforming one of them if the parities of the offsets match at each level. If they do not match, we have to retransform one of them. The same is done for the crop operation.

The main idea behind this scheme is that in many cases the coordinates of the subimage that will be pasted into another image are *known in advance*, so it can be transformed correctly. An example of this is the creation of one large image by gluing together several separately created subimages.

5.3.2 Scaling of Images

In Section 4.3 we described how images can be scaled in the wavelet domain. If a digital image is transformed, the low-pass band will be a low-resolution version of the original image. In this way we can downscale images by a power of 2. Alternatively, we can use the inverse transform — with all high-pass coefficients being zero — to upscale an image by a power of 2.

If we want to scale an image with a factor that is not a power of two, we scale the image using a power of 2 factor such that the result is larger than the wanted result. After that a classical interpolation algorithm is used to downscale the image to the wanted size.

5.3.3 Noise Reduction

Wavelet soft-thresholding is used to reduce additive, stationary noise. We select a threshold for each transform level and for each subband separately. To select these thresholds we use generalized cross validation (See Section 4.2.2). Since this is an asymptotical method, it only performs well for sufficiently large subbands. Experiments pointed out that 1000 coefficients in a subband is about the minimum for a good threshold estimation. Therefore, WAILI does not perform the denoising operation on subbands with less than 1000 coefficients. These are the subbands at coarse levels. In most cases, most of the noise appears at finer levels, and so, leaving the coarse levels untouched generally gives good results.

5.4 Implementation Issues

5.4.1 In-Place Transform

A wavelet transform implementation using the lifting scheme consists of two parts:

- A splitting step (lazy wavelet transform)
- One or more lifting steps

The splitting step is just a partitioning of the data and does not require a physical data transport, and the lifting steps operate on the two partitions only. Thus the wavelet transform can be implemented "in-place" by using the lifting scheme. No additional memory is required. The same is true for the inverse transform. Traditional filter bank algorithms do require additional memory, to store both the original and the transformed signal during the computations.

Because after each transform level one iterates on the "even" partition, one will get an "interleaved" representation in the wavelet domain. Indeed, after two transform levels the low-pass subband is formed by every fourth coefficient, after three transform levels by every eight coefficient, and so on. This interleaving causes a non-optimal memory access pattern because the wavelet transform operates on data that is not contiguous in memory. However, the locality in space is preserved: neighboring coefficients correspond to neighboring features of the original signal.

Alternatively the wavelet coefficients can be reordered to store the coefficients belonging to the same subband together.

5.4.2 The Two-Dimensional Wavelet Transform: Mallat Ordering

The two-dimensional wavelet transform of a matrix is given by transforming the columns and rows of the matrix independently using the one-dimensional transform. This results in a tensor product transform at each transform level. Instead of with 2 subbands (low-pass and high-pass), one will end up with 4 subbands: `LL` — low-pass in both the horizontal and vertical directions, `LH` — low-pass in the vertical, high-pass in the horizontal direction, `HL` and `HH`. When iterated on the LL subband, the result is a multi-resolution decomposition as shown in Figure 5.3. This ordering of the subbands at the different resolution levels is called the "Mallat" ordering [35].

Since the data belonging to the same subband are stored together, the Mallat ordering is advantageous if one is interested in multi-resolution analysis, or to provide progressive transmission. One can easily start with a low-resolution representation of the original image and gradually increase the resolution by transmitting additional high-pass subbands.

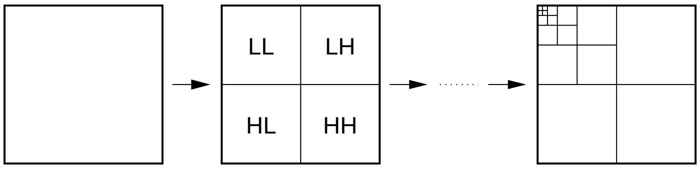

Figure 5.3: The two-dimensional wavelet transform: iteration on the `LL` subband, showing the wavelet coefficients in "Mallat" ordering.

In our applications we are interested in multi-resolution analysis and in progressive transmission, and we wanted to use the Mallat ordering. Hence we chose not to implement the lifting scheme "in-place", but to physically split the input data into two partitions (actually four, due to the two dimensions of images) in the splitting step. This does require additional memory, but has the additional advantage of operating on contiguous data, thus avoiding trashing of the data cache.

5.4.3 Transform Order

Normally the order of transform (first columns, then rows, or vice versa) at each transform level does not matter, but because of the rounding in the lifting steps, transform order does become important. Thus we always transform the columns first, followed by the rows. For the inverse transform, the order is inverted.

Lifting Steps on the Columns of a Matrix

Since matrices are stored in memory row by row, lifting the columns one after each other is not a good idea. By taking advantage of locality in memory and caching, it is faster to vectorize the calculations so they operate on rows. We use the following algorithm:

for all rows r
  for all columns c
    $x(c,r) \leftarrow \{a_0 \cdot y(c,r-1) + a_1 \cdot \mathrm{y}(c,r)\}$

This assumes we want to lift the coefficients in x using the coefficients in y, with lifting coefficients a_0 and a_1. Of course we have to take special care at the borders of the matrix.

Lifting Steps on the Rows of a Matrix

This can be done in the straightforward manner:

for all rows r
  for all columns c
    $x(c,r) \leftarrow \{a_0 \cdot y(c-1,r) + a_1 \cdot y(c,r)\}$

5.4.4 Normalization of the Wavelet Coefficients

Ideally we would like the normalization factor K_1 (see Equation 3.1) for the low-pass data to be $\sqrt{2}$ in all cases. Indeed, then the dynamic range of the low-pass data remains the same if we forget about the normalization factor. If the input samples are 8-bit values, the low-pass data after applying a wavelet transform step (the "mean" values) will be approx. 8-bit values too. However, there may be a slight bit growth in the low-pass band because the low-pass filter operation calculates a non-convex linear combination of the input data.

Of course we will have to remember the accumulated normalization factors for every subband if we want to process the wavelet-transformed data later. One way to do this is to consider each wavelet coefficient (i.e. the value after filtering) to be the mantissa m of a floating point number in base $\sqrt{2}$. The exponent e of this floating point number is $\log_{\sqrt{2}}$ of the accumulated normalization factor, which depends on the subband. Thus the real normalized value of each coefficient becomes:

$$x = m \cdot \left(\sqrt{2}\right)^e .$$

A special and simpler case of this is the two-dimensional wavelet transform where we always apply the transform to both the rows and the columns of a matrix. Since $\sqrt{2} \cdot \sqrt{2} = 2$, the normalization factor becomes 2, and the floating point numbers in base $\sqrt{2}$ can easily be converted to base 2.

Example

For CDF $(2,2)$, the lifting steps used are:

$$\begin{aligned} d_i &\leftarrow d_i - \left\{ \frac{s_i + s_{i+1}}{2} \right\}, \\ s_i &\leftarrow s_i + \left\{ \frac{d_{i-1} + d_i}{4} \right\}, \end{aligned}$$

Implying normalization factors

$$\begin{aligned} K_1 &= \sqrt{2} = \left(\sqrt{2}\right)^{+1}, \\ K_2 &= \frac{\sqrt{2}}{2} = \left(\sqrt{2}\right)^{-1}. \end{aligned}$$

In two dimensions, we have 4 subbands (LL, HL, LH, HH) after one transform step, with normalizations factors:

$$\begin{aligned}
K_{LL} &= K_1 \cdot K_1 = \left(\sqrt{2}\right)^{+2} = 2 \\
K_{LH} &= K_1 \cdot K_2 = \left(\sqrt{2}\right)^{+0} = 1 \\
K_{HL} &= K_2 \cdot K_1 = \left(\sqrt{2}\right)^{+0} = 1 \\
K_{HH} &= K_2 \cdot K_2 = \left(\sqrt{2}\right)^{-2} = \frac{1}{2}
\end{aligned}$$

For further transform steps, we have to take into account that the normalization factors accumulate due to the iteration on the low-pass subbands. Applying two wavelet transform steps, as shown in fig. 5.4, gives the following results. $x_S^{(l)}$ and $m_S^{(l)}$ represent a coefficient value respectively a mantissa in subband S at decomposition level l.

$$\begin{array}{lllllll}
\boxed{\textbf{Level 0}} & LL & : & x_{LL}^{(0)} & = & m_{LL}^{(0)} & = & m_{LL} \cdot 2^{+0} \\
\\
\boxed{\textbf{Level 1}} & LL & : & x_{LL}^{(1)} & = & m_{LL}^{(1)} \cdot \left(\sqrt{2}\right)^{+2} & = & m_{LL}^{(1)} \cdot 2^{+1} \\
 & LH & : & x_{LH}^{(1)} & = & m_{LH}^{(1)} \cdot \left(\sqrt{2}\right)^{+0} & = & m_{LH}^{(1)} \cdot 2^{+0} \\
 & HL & : & x_{HL}^{(1)} & = & m_{HL}^{(1)} \cdot \left(\sqrt{2}\right)^{+0} & = & m_{HL}^{(1)} \cdot 2^{+0} \\
 & HH & : & x_{HH}^{(1)} & = & m_{HH}^{(1)} \cdot \left(\sqrt{2}\right)^{-2} & = & m_{HH}^{(1)} \cdot 2^{-1} \\
\\
\boxed{\textbf{Level 2}} & LL & : & x_{LL}^{(2)} & = & m_{LL}^{(2)} \cdot \left(\sqrt{2}\right)^{+2} \cdot \left(\sqrt{2}\right)^{+2} & = & m_{LL}^{(2)} \cdot 2^{+2} \\
 & LH & : & x_{LH}^{(2)} & = & m_{LH}^{(2)} \cdot \left(\sqrt{2}\right)^{+2} \cdot \left(\sqrt{2}\right)^{+0} & = & m_{LH}^{(2)} \cdot 2^{+1} \\
 & HL & : & x_{HL}^{(2)} & = & m_{HL}^{(2)} \cdot \left(\sqrt{2}\right)^{+2} \cdot \left(\sqrt{2}\right)^{+0} & = & m_{HL}^{(2)} \cdot 2^{+1} \\
 & HH & : & x_{HH}^{(2)} & = & m_{HH}^{(2)} \cdot \left(\sqrt{2}\right)^{+2} \cdot \left(\sqrt{2}\right)^{-2} & = & m_{HH}^{(2)} \cdot 2^{+0}
\end{array}$$

Within WAILI, the exponent e is called a “shift” because it indicates how many places the mantissa m has to be shifted to the left (in base $\sqrt{2}$!) to obtain its real value x. With each subband, a shift is associated, depending on the type of wavelet and the transform depth. In the special two-dimensional case where always both rows and columns are transformed, the shifts can be interpreted as binary shifts after dividing them be 2.

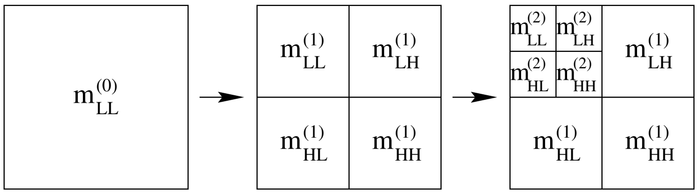

Figure 5.4: Two steps of the two-dimensional wavelet transform.

5.5 Summary of the Functionality of WAILI

Summarized, the software library WAILI provides the following functionality:

Wavelet transforms Wavelet transforms of images based on the integer version of the lifting scheme. One can transform both the columns and rows of an image, or only the columns or rows.

Image processing operations Crop and merge, noise reduction based on generalized cross validation, scaling, soft and hard thresholding, and edge enhancement.

Basic operations Addition and subtraction of images.

Utility operations Calculation of peak signal to noise ratio (PSNR), first order Shannon entropy, correlation matrices, minima and maxima in an image, histograms, and GCV values.

Image import/export Import of images in PNM and TIFF formats, and export of images in PNM format.

WAILI includes an interactive demo program that shows the usage of some of these operations, like e.g. wavelet transforms, thresholding and noise reduction. C++ source code for a simple image compression example using WAILI is present in Appendix B.

Including the demo program and the examples, WAILI counts ca. 12500 lines of C++ code.

5.6 Evaluation of the Compression Potential

In this section we evaluate the decorrelation achieved by some different kinds of wavelet transforms, as implemented in WAILI. This decorrelation is a measure for the compression potential.

5.6.1 Description of the Compression Method

We used a very simple lossy compression method, namely hard-thresholding of the wavelet coefficients. Thus all wavelet coefficients with a magnitude smaller than the threshold value are replaced by zeros.

5.6.2 Measure of the Compression Rate

The achieved compression rate is estimated based on the first order entropy of the thresholded wavelet coefficients.

The first order Shannon entropy H for a string of symbols i from a set $\mathcal{S}$ is defined by:

$$H = -\sum_{i \in \mathcal{S}} p_i \log_2 p_i,$$

with p_i the probability of the appearance of the symbol i. The first order entropy H is the theoretical minimum number of bits per symbol needed to encode the string by considering individual symbols [41].

Note that encoding techniques that combine multiple symbols (e.g. zerotree encoding) can increase the encoding efficiency beyond the limit as indicated by the first order entropy.

5.6.3 Measure of the Image Quality

The image quality after performing the inverse wavelet transform is measured by the Peak Signal to Noise Ratio (PSNR). The PSNR of a reconstructed signal x_i^*, compared to the original signal x_i, is defined by

$$\text{PSNR} = 20 \log_{10} \left(\frac{x_{\text{peak}}}{\text{RMSE}} \right) dB,$$

with $x_{\text{peak}} = \max_i |x_i|$ and RMSE the Root Mean Square Error:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{n} (x_i - x_i^*)^2}{n}},$$

with n the signal length.

Other quality measures are also possible. Visual quality is important in many application fields, but it is subjective and difficult to measure. The 1-norm $\sum_i |x_i - x_i^*|$ is another option. We chose the PSNR because it is simple to calculate and commonly used. Furthermore the PSNR is related to the properties of (bi)orthogonal wavelets: the identity of Parseval (see Equation 2.4) and the similar formula for a Riesz basis (see Equation 2.5) suggest that neglecting the wavelet coefficients with the smallest magnitudes is a good compression approach if one wants to keep a high PSNR.

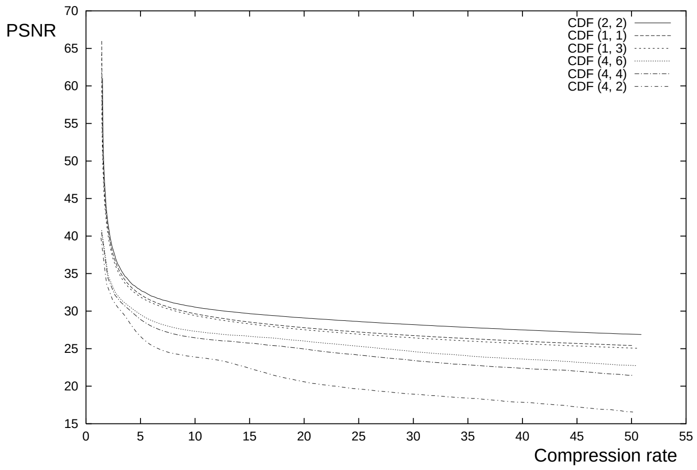

Figure 5.5: Comparison of compression rate vs. quality of reconstruction (PSNR, in dB) for different types of wavelets. The graphs for CDF $(2,4)$ and CDF $(2,6)$, and CDF $(1,5)$ are not shown because they are very similar to the graphs for CDF $(2,2)$ respectively CDF $(1,3)$.

5.6.4 Results

As test image, we used an 8-bit grey scale aerial picture of our campus, counting 768×768 pixels. The image was wavelet transformed using the integer wavelet transform routines of WAILI, up to 6 transform levels.

Figure 5.5 compares the compression rate vs. the quality of reconstruction (PSNR) for different types of wavelet transforms of the Cohen-Daubechies-Feauveau family of biorthogonal wavelets.

However, comparing the reconstructed image after decompression with the input image is not completely fair because the input image contains noise. Due to the methodological similarity between the noise reduction and compression operations, it is quite logical that the visual image quality at low compression rates is better than the visual quality of the noisy input image. To take this into account, Figure 5.6 shows the performance of compressing the original input image and two denoised versions:

Global denoising The first image was denoised by soft-thresholding with a global threshold (per subband), calculated using Generalized Cross Validation (see Section 4.2.2).

Adaptive denoising The second image was denoised by soft-thresholding with a local threshold, calculated using Generalized Cross Validation on blocks of

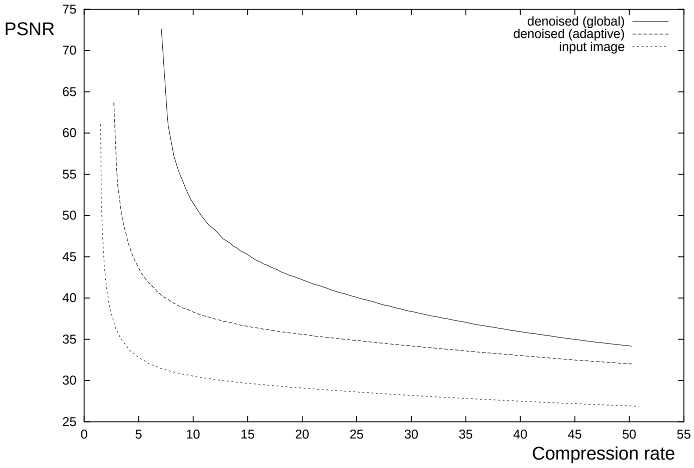

Figure 5.6: Comparison of compression rate vs. quality of reconstruction (PSNR, in dB) for CDF $(2,2)$ wavelets for the original input image and two denoised versions.

32×32 coefficients. This method adapts better to the local properties of the image and gives a better visual result, according to experts specialized in image processing (see Section 6.3.1).

In both cases the denoising was performed using the CDF $(2,2)$ wavelet transform.

Based on the experiments described above, we can formulate the following conclusions:

- CDF $(2,x)$ performs better than CDF $(1,x)$, and CDF $(1,x)$ performs better than CDF $(4,x)$.
- The 3 members of the CDF $(2,x)$ family ($(2,2)$, $(2,4)$ and $(2,6)$) perform about equally well. Since CDF $(2,2)$ is the cheapest to implement, it becomes our first choice.
- Within the CDF $(1,x)$ family, $(1,1)$ (Haar) performs best, followed by $(1,3)$ and $(1,5)$.
- Within the CDF $(4,x)$ family, there are very clear performance gaps between $(4,6)$ and $(4,4)$, and between $(4,4)$ and $(4,2)$.
- Compressing denoised images allows for higher compression rates, as expected.

5.7 Comparison with Other Wavelet Software

Several software packages and tools implementing wavelet transforms are available. The available wavelet software can be divided in three classes: software libraries to be linked with your own applications, extension packages for common integrated technical computing environments like Matlab and Mathematica, and stand-alone tools for data analysis.

Most or all of them implement classical first-generation wavelet transforms on floating point data. They are usually targeted at signal processing (one dimension: speech, sound, . . .) or image processing (two dimensions), providing simple operations like thresholding (for compression or noise reduction), or at numerical analysis applications. Some of them offer the wavelet packet transform as a straightforward extension to the fast wavelet transform. The Discrete Wavelet Transform (DWT) is most popular due to its simplicity and the discrete nature of digital signals, while some software also provides the Continuous Wavelet Transform (CWT) [13].

WAILI differs in that it is a software library providing *integer wavelet transforms* implemented using the *lifting scheme*. LIFTPACK [19] is a software library that also implements the lifting scheme, but it differs substantially from WAILI:

- WAILI uses integer wavelet transforms, while LIFTPACK uses floating point transforms,
- WAILI uses the "Mallat" ordering for the wavelet coefficients, while LIFTPACK implements the in-place transform with an interleaved ordering,
- WAILI uses symmetric extension at the image borders, while LIFTPACK modifies the wavelet bases near the borders.
- WAILI is targeted at image processing and includes some image processing operations, while LIFTPACK is meant for generic wavelet processing, including numerical analysis applications.