From lossy to lossless wavelet image coding in a tree-based encoder with resolution scalability

Jose Oliver[1], Manuel P. Malumbres[2]

[1] Department of Computer Engineering (DISCA), Technical University of Valencia,
Camino de Vera 17, 46017 Valencia, Spain
joliver@disca.upv.es
[2] Departamento de física y ATC, Miguel Hernández University,
Avda. Universidad s/n, 03203 Elche, Spain
mels@umh.es

Abstract. For a lossy encoder, it is important to be able to provide also lossless compression with little or no modification of the usual algorithm, so that an implementation of that algorithm can work in lossy or lossless mode, depending on the specific application, simply by varying the input parameters. In this paper, we evaluate the capability of the Lower Tree Wavelet (LTW) image encoder to work in lossless mode. LTW is a fast and multiresolution wavelet image encoder, which uses trees as a fast mode to group coefficients. In addition, general details on how to implement efficiently (i.e., with only shift and addition/subtraction operations) a reversible integer-to-integer wavelet transform are also given, as a requirement to implement a wavelet-based lossless encoder. Numerical results show that despite being general purpose (i.e., both lossy and lossless) and lacking of complex techniques (such as high-order context and predictive coding), the LTW performs as well as JPEG 2000 in lossless mode, and only 5% below LOCO-I, a specific lossless algorithm.

1 Introduction

Most specific lossless image coders are based on entropy coding with various contexts and predictive techniques. Predictive coding schemes try to predict each sample from the samples that have been previously encoded, which are available to both encoder and decoder. In image compression, prediction is usually performed from nearby pixels. Once a prediction has been calculated, the residual pixel is encoded as the error committed by this prediction. This way, the better a prediction is, the lower it will be the entropy of the residual pixels. The CALIC scheme [1] follows this approach, becoming one of the most efficient lossless image coders in terms of compression performance. A simplification of CALIC was adopted as the JPEG-LS standard, which replaced the lossless mode of the original JPEG standard. This simplified version of CALIC is called LOCO-I [2], and its performance is close to CALIC with lower complexity. Other lossless image encoders are PNG (proposed as a royalty-free alternative to GIF) and JBIG (intended to bi-level image coding and used in fax transmission).

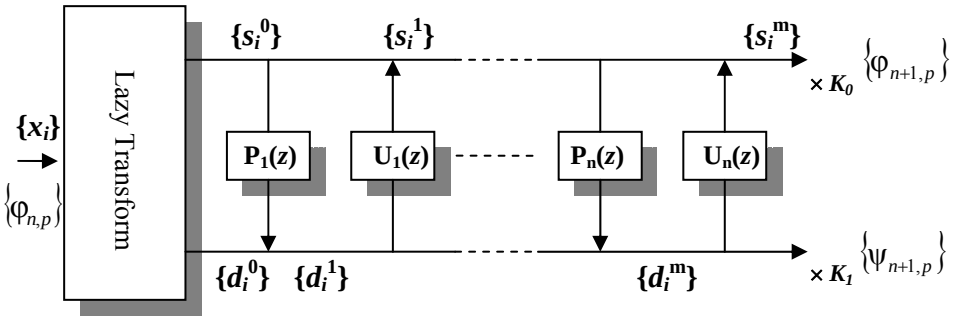

Fig. 1. General diagram for a wavelet decomposition using the lifting scheme.

On the other hand, an interesting feature of general lossy image encoders is the ability to losslessly encode an image if no quantization is applied. This way, the emerging JPEG 2000 standard [3] was designed to be able to work in both lossy and lossless mode. SPIHT [4] and EZW [5] are tree-based lossy wavelet image encoders that also can store an image in lossless mode with SNR scalability.

LTW [6] was proposed as a low-complexity multiresolution alternative to the previous encoders. Multiresolution is a very interesting feature in heterogeneous frameworks (such as today's Internet) in which multiple devices with different display capabilities (e.g., image size) are potential clients. For instant, if an image is encoded with spatial scalability, the same bitstream can be employed by a mobile phone (reading only the base layer), a PDA (reading an additional layer to provide a slightly higher resolution) and a desktop computer (maybe needing all the image layers for full resolution).

In this paper, we describe the details to implement the LTW encoder working in a lossless mode, implemented with integer data type. We tackle this problem within the two stages of a wavelet-based image coder, i.e., in the wavelet transform and in the coding stage.

Many applications need to be able to work in lossless mode. Medical imaging is an example of this type of application in which lossless compression is required, since all the image details must be preserved so that medical analysis is not hindered. Another application of lossless coding is image editing. In this type of application, if lossy compression is employed, accumulative errors from successive editions may seriously damage the final image quality.

Lossless compression requires reversibility, which is not guaranteed with regular floating-point operations due to the finite-precision of the operands. In this case, a reversible integer-to-integer implementation is needed. In addition, an integer implementation is not only interesting for lossless image coding to achieve a reversible transform, but also in hardware architectures that only support integer arithmetic, such as some DSPs and many FPGAs. In fact, doing floating-point on FPGAs is difficult due to large amount of hardware required.

The rest of this paper is structured as follows. In Section 2, there is a detailed description of the wavelet transform implemented with the lifting scheme, focusing on a reversible implementation with integer data types. Section 3 describes the LTW algo-

rithm and the required details to work in lossless mode. In Section 4, some experimental results are given, comparing the LTW encoder with JPEG 2000 working in lossless mode, and the specific lossless encoder LOCO-I. Finally, in Section 5 some conclusions are given.

2 Reversible wavelet transform

The wavelet transform was earlier defined and implemented using a regular filtering operation following a multiresolution analysis [7], but a more efficient algorithm to compute it was introduced by Sweldens in [8]. This algorithm is called the lifting scheme. The main advantage of this approach is the reduction in the number of operations needed to perform the wavelet transform. An additional advantage is that it allows in-place computation, and hence no extra memory is required to store the resulting coefficients as it happens with any regular filtering method. The third benefit that the lifting scheme introduces is the feasibility of a reversible integer-to-integer wavelet transform with only a slight modification of the usual floating-point implementation. In this section, we will deal with this type of integer wavelet transform.

We have mentioned that the lifting scheme implements an in-place DWT decomposition as an alternative algorithm to the classical filtering algorithm. In the filtering algorithm, in-place processing is not possible because each input sample is required as incoming data for the computation of its neighbor coefficients. Therefore, an extra array is needed to store the resulting coefficients, doubling the memory requirements. In addition, the lifting-scheme reduces the number of operations needed to compute the DWT.

In Figure 1, we present a diagram to illustrate the general lifting process. The whole process consists of a first lazy transform, one or several prediction and update steps, and coefficient normalization. In the lazy transform, the input samples are split into two data sets, one with the even samples and the other one with the odd ones. Thus, if we consider $\{x_i\}$ the input samples, we define both coefficient sets as:

$$\left\{s_i^{\,0}\right\} = \{x_{2i}\} \qquad \left\{d_i^{\,0}\right\} = \{x_{2i+1}\}$$

Then, in a prediction step (sometimes called dual lifting), each sample in $\left\{d_i^{\,0}\right\}$ is replaced by the error committed in the prediction of that sample from the samples in $\left\{s_i^{\,0}\right\}$:

$$d_i^{\,1} = d_i^{\,0} - P\left(\left\{s_i^{\,0}\right\}\right)$$

while in an update step (also known as primal lifting), each sample in the set $\left\{s_i^{\,0}\right\}$ is updated by $\left\{d_i^{\,1}\right\}$ as:

$$s_i^{\,1} = s_i^{\,0} + U\left(\left\{d_i^{\,1}\right\}\right)$$

After *m* successive prediction and update steps, the final low frequency coefficients (scaling $\{\varphi_i\}$) and high frequency coefficients (wavelet $\{\psi_i\}$) are achieved normalization:

$$\{\varphi_i\} = K_0 \times \{s_i^m\} \qquad \{\psi_i\} = K_1 \times \{d_i^m\}$$

A nice feature of the lifting scheme is that it is formed by very simple steps, and each of these steps is easily invertible, which leads to an almost trivial inverse transform. For the inverse transform, we only have to perform the inverse operations in the reverse order. Hence, from the subsets $\{\varphi_i\}$ and $\{\psi_i\}$, we can get $\{s_i^m\}$ and $\{d_i^m\}$ simply by dividing these coefficients by the scaling factors:

$$\{s_i^m\} = \{\varphi_i\}/K_0 \qquad \{d_i^m\} = \{\psi_i\}/K_1$$

Then, an inverse update operation can be done from these data sets as follows:

$$s_i^{m-1} = s_i^m - U(\{d_i^m\})$$

and at this moment, we can apply the inverse prediction step:

$$d_i^{m-1} = d_i^m + P(\{s_i^{m-1}\})$$

After *m* successive inverse update and prediction steps, we get the initial sets of even and odd samples, we can interleave these data sets to obtain the original set of samples $\{x_i\}$.

2.1 The integer-to-integer lifting scheme

With the above scheme, floating-point arithmetic is needed despite having integer input samples (e.g., image pixels), if the weighting factors employed for the prediction/update operations are floating-point and not integer or rational. Actually, even if rational filters are employed, the precision required to perform lossless operation with fixed-point arithmetic grows with each mathematical operation if we do not change the scheme described above.

Fortunately, the lifting scheme can be slightly modified to achieve reversible integer-to-integer wavelet transform [9]. Since the lifting scheme is formed by several simple steps, the whole process can be reversible if we perform each single step in a reversible way.

For the forward transform, we have seen that each prediction step has the form:

$$d_i^m = d_i^{m-1} - P(\{s_i^{m-1}\})$$

In a wavelet transform for integer implementation, the prediction operation $P\left(\left\{s_i^{m-1}\right\}\right)$ involves rational weighting factors (e.g., division by two), and hence the resulting data are not integer. If a rounding operation is added after the prediction operation, an integer variable can be used to store the result of that operation, and hence each d_i^m can be computed from d_i^{m-1} and the $\left\{s_i^{m-1}\right\}$ set using integer values as follows:

$$d_i^m = d_i^{m-1} - \left\lfloor P\left(\left\{s_i^{m-1}\right\}\right)\right\rfloor$$

In the inverse transform, the exact value of each d_i^{m-1} can be recovered from d_i^m and the $\left\{s_i^{m-1}\right\}$ set as follows:

$$d_i^{m-1} = d_i^m + \left\lfloor P\left(\left\{s_i^{m-1}\right\}\right)\right\rfloor$$

Thereby, perfect reconstruction is guaranteed despite the rounding operation. The same analysis can be performed for an update operation with integer data type.

Although we have used the floor operator for rounding in the above equations, any other rounding operation, such as ceil or rounding to the nearest integer, can be used as long as the same operator is employed in both the forward and inverse transforms.

Finally, a reversible integer-to-integer transform can only be obtained if the normalization factors K_0 and K_1 are integer values.

A drawback of the use of rounding is that the new wavelet transform is no longer linear. Hence, for a 2D wavelet transform, the reverse column-row order of the forward transform has to be used in the inverse transform to achieve perfect reconstruction.

2.2 An implementation using the bi-orthogonal 5/3 transform

The 5/3 wavelet transform is a typical wavelet for integer-to-integer transform, being part of the JPEG2000 standard for lossless compression. In order to compute it in terms of the lifting scheme, after the lazy transform, the dual lifting is calculated as:

$$d_i^1 = d_i^0 - \left\lfloor \frac{1}{2}\left(s_i^0 + s_{i+1}^0\right)\right\rfloor$$

while the primal lifting is (notice the different rounding):

$$s_i^1 = s_i^0 + \left\lfloor \frac{1}{4}\left(d_i^1 + d_{i-1}^1\right) + \frac{1}{2}\right\rfloor$$

These operations can be easily performed with integer data types and integer arithmetic. For example, in C language, the two above equations can be efficiently computed as:

```
d1[i]=d0[i]-((s0[i]+s0[i+1])>>1);
s1[i]=s0[i]+((d1[i]+d1[i-1]+2)>>2);
```

Where d0, d1, s0 and s1 are arrays of integers, and >> is the right shift operator in C ($a >> b$ is equivalent to the division of a by 2^b with floor rounding).

For a lossless transform, the normalization factors K_0 and K_1 are equal to 1, achieving (1,2) normalization in this case. Thus, the set $\{d_i^{\,1}\}$ is directly the final wavelet coefficient set, and the set $\{s_i^{\,1}\}$ is the scaling one.

The inverse transform to recover losslessly the original samples is given by:

$$s_i^{\,0} = s_i^{\,1} - \left\lfloor \frac{1}{4}\left(d_i^{\,1} + d_{i-1}^{\,1}\right) + \frac{1}{2} \right\rfloor \qquad d_i^{\,0} = d_i^{\,1} - \left\lfloor \frac{1}{2}\left(s_i^{\,0} + s_{i+1}^{\,0}\right) \right\rfloor$$

Other reversible integer-to-integer wavelet transforms are given in [10], including an integer version of the popular bi-orthogonal 9/7 transform [11].

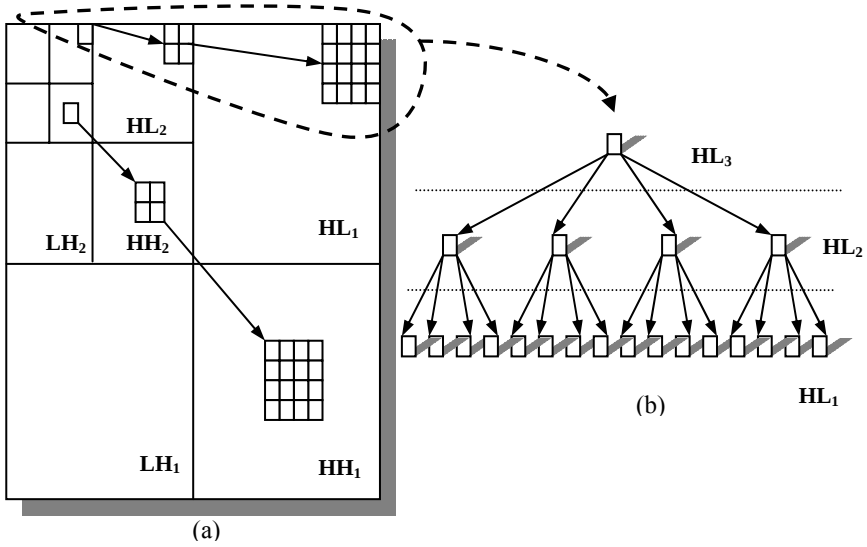

Fig. 2. Definition of wavelet coefficient trees. In (a), it is shown that coefficients of the same type of subband (HL, LH or HH) representing the same image area through different levels can be logically arranged as a quadtree, in which each node is a wavelet coefficient. The parent/children relation between each a pair of nodes in the quadtree is presented in (b).

3 Tree-based coding of wavelet coefficients with multiresolution

3.1 Multiresolution in wavelet image coding

One of the features that have turned the wavelet transform so popular is the ability to perform a multiresolution analysis. In order to achieve this type of scalability, the order in which the coefficients are received by the decoder has to follow a decreasing order of the subband level. This way, the first subband that the decoder attains is the LL_N, which is a low-resolution scaled version of the original image. Then, the decoder progressively receives the remaining subbands, from lower frequency subbands to higher ones, which are used as a complement to the low-resolution image to recursively double its size, which is know as Mallat decomposition [7].

In tree-based wavelet image coding, neither EZW [5] nor SPIHT [4] possess multiresolution scalability due to the successive scans that they perform, focusing on a different bit plane in each scan. The Lower Tree Wavelet (LTW) image encoder [6] was one of the first tree-based wavelet encoders to introduce multiresolution, at the expense of losing SNR scalability. In the next subsection, we describe the LTW encoder.

3.2 Lower-tree wavelet coding

In the LTW encoder [6], the quantization process is performed by two strategies: one coarser and another finer. The finer one consists in applying a scalar uniform quantization, *Q*, to wavelet coefficients. On the other hand, the coarser one is based on removing least significant bit planes. We define *rplanes* as the number of least significant bit planes that have been removed from the wavelet coefficients.

In this encoder, a tree structure (like the one shown in Figure 2) is used, not only to reduce data redundancy among subbands, but also as a simple and fast way of grouping coefficients. As a consequence, the total number of symbols needed to encode the image is reduced, decreasing the overall execution time (because the arithmetic encoder stores less symbols). This structure is called lower tree, and it is a coefficient tree in which all its coefficients are lower than $2^{rplanes}$.

Our algorithm consists of two stages: (a) the construction of the significant map and (b) coefficient coding based on the symbols that have been computed in the first stage. In the first stage, the significance map is built after quantizing the wavelet coefficients (by means of using both *Q* and *rplanes* parameters). For the arithmetic encoder, the symbol set employed in our proposal is the following one:

(1) A *LOWER* symbol represents a coefficient that is the root of a lower-tree. The rest of coefficients in a lower-tree are labeled as *LOWER_COMPONENT*, but they are never encoded because they are already represented by the root coefficient.

(2) If a coefficient is insignificant (i.e., lower than $2^{rplanes}$), but it does not belong to a lower-tree because it has at least one significant descendant, it is labeled as an *ISOLATED_LOWER* symbol.

(3) For a significant coefficient (i.e., higher or equal to $2^{rplanes}$), we use a symbol indicating the number of bits needed to represent it. Finally, there is a special type of significant coefficient in which all its descendants are insignificant. This type of symbol is able to represent efficiently some special lower-trees, where only the root coefficient is significant, and the descendants are insignificant.

Let us describe now the whole coding algorithm.

In the first stage (symbol computation), all wavelet subbands are scanned in 2×2 blocks of coefficients, from the first decomposition level to the N^{th} (to be able to build the lower-trees from leaves to root). In the first level subband, if the four coefficients in each 2×2 block are insignificant (i.e., lower than $2^{rplanes}$), they are considered to be part of the same lower-tree, and thereby the are labeled as *LOWER_COMPONENT*. Then, when scanning upper level subbands, if a 2×2 block has four insignificant coefficients, and all their direct descendants are *LOWER_COMPONENT*, the coefficients in that block can be labeled as *LOWER_COMPONENT* as well, increasing the lower-tree size.

However, when at least one coefficient in the block is significant, the lower-tree cannot continue growing. In that case, a symbol for each coefficient is computed one by one. Each insignificant coefficient in the block is assigned a *LOWER* symbol if all its descendants are *LOWER_COMPONENT*, otherwise it is assigned an *ISOLATED_LOWER* symbol. On the other hand, for each significant coefficient, a symbol indicating the number of bits needed to represent that coefficient is employed, but this symbol is marked as a special symbol if its direct descendants are *LOWER_COMPONENT* to be able to identify this type of tree.

Finally, in the second stage, the subbands are encoded from the LL_N subband to the first-level wavelet subbands. Observe that this is the order in which the decoder needs to know the symbols, so that lower-tree roots are decoded before its leaves. In addition, this order provides resolution scalability.

In each subband, for each 2×2 block of coefficients, the symbols that were computed in the first stage are entropy coded by means of an arithmetic encoder with two simple contexts based on the significance of the upper coefficient and the coefficient previously encoded (on the left). Recall that no *LOWER_COMPONENT* is encoded. In addition, for the significant coefficients, the significant bits and its sign are also needed, and therefore they are binary encoded.

3.3 Lossless mode

As we mentioned in the introduction, it is important for an encoder to be able to provide lossless compression with little or no modification of the usual algorithm, so that an implementation of that algorithm can work in lossy or lossless mode, depending on the specific application, simply by varying the input parameters. The Lower Tree Wavelet encoder possesses this feature if no quantization is applied and an inte-

ger-to-integer wavelet transform, such as the one presented in the previous section, is used. In order to skip the quantization process, the quantization parameters presented in the description of the algorithm can be set as *rplanes*=0, *Q*=1, although it is faster if we simply omit all the operations related to the scalar quantization. For the wavelet transform, we will use the reversible bi-orthogonal 5/3 filter bank for integer implementation, which is fully described in Section 2.

Table 1. Lossless coding comparison of various image encoders with six greyscale 8 bpp images. Results are given in bits per pixel (bpp) needed to losslessly encode the original image.

codec \ image	LOCO-I	JPEG 2000	LTW
Lena (512×512)	4.24	4.31	4.26
Barbara (512×512)	4.86	4.78	4.83
Goldhill (512×512)	4.71	4.84	4.78
Woman (2560×2048)	4.45	4.51	4.50
Café (2560×2048)	5.09	5.35	5.36
Bike (2560×2048)	4.36	4.53	4.56

4 Numerical results

In Table 1, we compare the results of losslessly encode six images (grayscale 8 bpp) with our encoder, JPEG 2000 and the LOCO-I algorithm (in which the JPEG-LS standard is based). In JPEG 2000, the same bi-orthogonal 5/3 transform is used. In this table, results are expressed as the number of bits per pixel needed for the compressed image, and in general it is reduced from 8 bpp (in the original image) to 4-5 bpp after lossless coding. LTW and JPEG 2000 are general purpose encoders and, if we compare them, they perform almost the same in all the images, with no more than 0.05 bpp difference between them (about 1% in performance). This is a good result for our encoder, if we take into account that lossless coding is mainly based on predictive techniques and context modeling (which are heavily developed in JPEG 2000). LTW, contrary to JPEG 2000, only handles two contexts. As we said in the introduction, LOCO-I [2] is a specific prediction-based lossless technique in which the lossless standard JPEG-LS is based. However, it is not much more efficient than the other two encoders under evaluation, requiring about 0.1-0.2 bpp less than JPEG 2000 and LTW. In particular, LOCO-I's coding efficiency is not higher than 5% compared with JPEG 2000 and LTW.

5 Conclusions

In this paper, we have presented the LTW encoder in a lossless framework, showing that it is also competitive with this type of application. In fact, the coding efficiency is only 5% under the specific lossless algorithm LOCO-I.

In addition, we have presented a detailed description on how to implement the wavelet transform using the lifting scheme. To this end, we have provided the exact instructions in C language to implement this transform with only shift and addition/subtraction operations. This implementation can be used for any wavelet-based encoder, and even to implement the wavelet transform with hardware architectures that only support integer arithmetic.

Acknowledgment

This work was supported by the Spanish Ministry of Education and Science under grant TIC2004-0052.

References

1. X. Wu, N.D. Memon, *CALIC- A Context Based Adaptive Lossless Image Coding Scheme*, IEEE Transactions on Communications, Vol. 45, 437-444, May 1996.
2. M. Weinberger, G. Seroussi, G. Sapiro, *The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS*, IEEE Transactions on Image Processing, Vol. 9, 1309-1324, August 2000.
3. ISO/IEC 15444-1: *JPEG2000 image coding system*, 2000.
4. A. Said, 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, Vol. 6, no3, June 1996.
5. J.M. Shapiro, *Embedded image coding using zerotrees of wavelet coefficients*, IEEE Transactions on Signal Processing, Vol. 41, n12, December 1993.
6. J. Oliver, M. P. Malumbres, *Fast and Efficient Spatial Scalable Image Compression Using Wavelet Lower Trees*, in Proc. IEEE Data Compression Conference, Snowbird, UT, March 2003
7. S. Mallat, *A Theory for Multiresolution Signal Decomposition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, pp. 674-693, July 1989.
8. W. Sweldens, *The lifting scheme: a custom-design construction of biorthogonal wavelets*, Journal of Applied Computational and Harmonic Analysis, vol. 3, pp. 186-200, 1996.
9. R. C. Calderbank, I. Daubechies, W. Sweldens, B. L. Yeo, Wavelet transforms that map integers to integer, Journal of Applied Computational and Harmonic Analysis, vol. 5, pp. 332-369, 1998.
10. M. Adams, F. Kossentini, *Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis*, IEEE Transactions on Image Processing, vol. 9, pp. 1010-1024, June 2000.
11. M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, *Imagen Coding Using Wavelet Transform*, IEEE Transactions on Image Processing, vol. 1, no2, April 1992.