An Algorithm for Low Memory Wavelet Image Compression

Christos Chrysafis
Hewlett-Packard Laboratories
1501 Page Mill Road, Bldg.3U-3
Palo Alto, CA 94304-1126
chrysafi@hpl.hp.com
Tel: 650-857-3382, Fax: 650-857-4691

Antonio Ortega*
Integrated Media Systems Center
University of Southern California
Los Angeles, CA 90089-2564
ortega@sipi.usc.edu
Tel:213-740-2320, Fax:213-740-4651

Abstract

As wavelet-based image coding is set to become more widely used (e.g. with the completion of the JPEG2000 standard), memory efficiency for wavelet-based coding is becoming an increasingly important issue. In this paper we present a complete system to perform low memory wavelet image coding. Our approach is "line-based" in that the images are read line by line and only the minimum required number of lines is kept in memory. The line-based transform is combined with a low memory entropy coder that does not require any global image information. Our system achieves a large (two orders of magnitude) reduction in memory requirements compared to other available coders, with limited performance loss (e.g., less than 0.5dB).

1 Introduction

Memory efficient compression algorithms are needed for some cost-sensitive applications (e.g. printers or digital cameras) and are also desirable in general purpose platforms (e.g., lower memory requirements can speed up computation by allowing more effective caching.) Discrete Cosine Transform (DCT) based compression (e.g., JPEG [1]) is memory efficient, since processing is based on small blocks. In this paper we investigate memory efficiency for wavelet-based image coding algorithms, such as those that will make the core of the upcoming JPEG2000 standard [2].

Algorithms such as those in [3, 4, 5], are representative of the state of the art in wavelet coders. These algorithms assume that the wavelet transform (WT) of the whole image is computed before coding, so that some useful "global" information (e.g., maximum and minimum coefficient values in one subband, energy per subband, etc) can be derived and used for coding (e.g. for classification, initialization of the probability models used by a quantizer, etc.) Clearly, the corresponding memory utilization will be of the order of the image size. Instead, in this paper we describe a wavelet-based coder with a memory requirement well below the size of the input image. An early version of this work was presented in [6] and a complete description of the system can be found in [7]. A complete encoder and decoder system that operates with low memory requires (i) a wavelet transform implementation that uses as little memory as possible and (ii) an algorithm that can compress wavelet coefficients as they are generated and does not rely on global information.

Low memory implementations of the WT were first addressed in [8], where only one-dimensional (1D) transforms were studied. This work did not consider the synchronization issues that arise when both forward and inverse transform memory requirements are considered, and it focused only on the WT (i.e., the impact of low memory operation on a potential compression algorithm was not studied.) One straightforward approach to address overall (i.e., transform and coding) memory is to tile a large image into smaller subimages, which are then independently coded. This approach has well known drawbacks in terms of coding performance. The more efficient approach of [9] uses a standard algorithm (e.g. [3]), which buffers the whole image at the encoder, but then allows the bit-stream to be re-ordered before transmission so as to reduce the memory utilization at the decoder.

Unlike prior work, we address both encoder and decoder memory utilization and consider a complete coding system. To address the low memory wavelet transform requirement we propose a line-based implementation, *which yields the same results as a "normal" row column filtering implementation*, and where, we address memory issues arising from the need to synchronize encoder and decoder. Our proposed line-based transform, which has been incorporated in the JPEG 2000 verification model, is discussed

*This work was supported in part by the National Science Foundation under grant MIP-9502227 (CAREER), the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, the Annenberg Center for Communication at the University of Southern California, the California Trade and Commerce Agency, and Texas Instruments.

briefly in Section 2; we refer to [7] for more details. Our focus in this paper is the novel context-based encoder described in Section 3. This encoder achieves excellent compression performance while not requiring any global information to be available. Wavelet coefficients are generated line-by-line in an interleaved fashion and are encoded and transmitted as soon as they are generated. Thus, overall, only a few lines of wavelet coefficients (rather than entire subbands) will have to be buffered before encoding. The experimental results of Section 4 show that, with respect to existing coders, the degradation in performance is modest (e.g., less than $0.5dB$ in average with respect to [4]) although we do not support progressive transmission as [3]. In terms of memory utilization our results show reductions of almost *two orders of magnitude* with respect to widely available implementations of wavelet image coders.

2 Line-based Wavelet Transform

Throughout this paper we will assume that the encoder can scan or read an image one line at a time. Obviously, performing a 1D WT on a single line can be done without significant memory. However, in order to implement the separable 2D transform the next step is to perform column filtering and here memory utilization can become a concern. For example, a completely separable implementation would require that all the lines be filtered before column filtering starts and thus memory sizes of the order of the image size will be required. The obvious alternative is to start column filtering as soon as a sufficient number of lines, as determined by the filter length, has been horizontally filtered. For example, for a one level decomposition, if we use 9-7 tap filters we only need 9 image lines in order to start the column filtering and generate the first line of output wavelet coefficients.

This online computation approach forms the basis of our wavelet filtering and was considered in [8] for the 1D case. Assume X is the width of an image and L is the maximum length of a filter in a filter bank. Because dyadic decompositions are employed (and thus after each level, only haff the output coefficients are filtered again) it can be shown that the memory needed for filtering is $2L$ lines, asymptotically as the number of levels of decomposition increases.

However, as illustrated in Fig. 1, additional memory is needed because unbalanced decomposition trees (e.g., dyadic decompositions) are used. As can be seen in Fig. 1, some of the highpass samples ($H_1(z)$ filter) have to be buffered at either encoder or decoder, since they can only be used for reconstruction with the output of the lowpass branch, and this output is delayed. Thus, it will be necessary to introduce some additional "line management" functionality to *synchronize* encoder and decoder.

For 1D signals and N levels of decomposition, the memory needs for the n^{th} level of decomposition will consist of (i) a filtering buffer of size L, and (ii) a synchronization buffer of size $D_{N-n} = (2^{N-n} - 1)S$. Therefore the total memory size needed for N levels of decomposition for both analysis and synthesis filter banks is: $T_{total,N} = \sum_{n=0}^{N-1}(D_{N-n} + L) = (2^N - N - 1)S + NL$. Note that as the number of levels increases synchronization buffers become a major concern. For 2D signals the increase in the synchronization buffer sizes is even more dramatic. For an N-level dyadic decomposition, we will need filtering buffers for up to $2LX$ pixels, and synchronization buffers for $T_N^{(2d)} = (2 \cdot 2^N + 2^{-N} - 3)XS$ pixels. Thus, as the number of decomposition levels grows, the filtering requirements remain relatively modest, while the size of the synchronization buffers tends to grow fast. However, memory-efficient implementations are still possible because the filtering buffers hold data that is accessed multiple times, while the synchronization buffers are only delay lines (FIFO queues). This is a key distinction because the data in the synchronization buffers will only be used by the decoder and therefore it can be *quantized and entropy coded* so that the actual memory requirements are much lower (see [7] for details.)

3 Line based entropy coder

We now describe our proposed low-memory coding strategy. Processing is based on the lines of wavelet coefficients produced by the WT, which are compressed as they are generated; no global information is required. Each band is encoded *independently*, i.e., we do not use *any* kind of information from one band in order to encode another. All wavelet coefficients are quantized with a *dead-zone quantizer*, with a step size δ that is chosen to be the same for all subbands. The quantizer maps a wavelet coefficient α to the index $v = \lfloor \frac{\alpha}{\delta} \rfloor$. The reconstructed value $\hat{\alpha}$ is obtained as

$$\hat{\alpha} = \begin{cases} (v + 1/2) \cdot \delta & \text{if} \quad v > 0 \\ (v - 1/2) \cdot \delta & \text{if} \quad v < 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

3.1 Context Modeling and low memory

Context modeling is a key ingredient in many compression algorithms, and is particularly useful for lossless compression [10, 11]. These algorithms exploit the fact that large coefficients tend to be found in clusters in image subbands. Thus, in a neighborhood with several large coefficients, another large coefficient will tend to occur with higher probability. Therefore context information will be used to classify the neighborhood so that different probability models are used depending on the magnitude of the surrounding coefficients. Here we will use context modeling for lossless compression of the quantized coefficients.

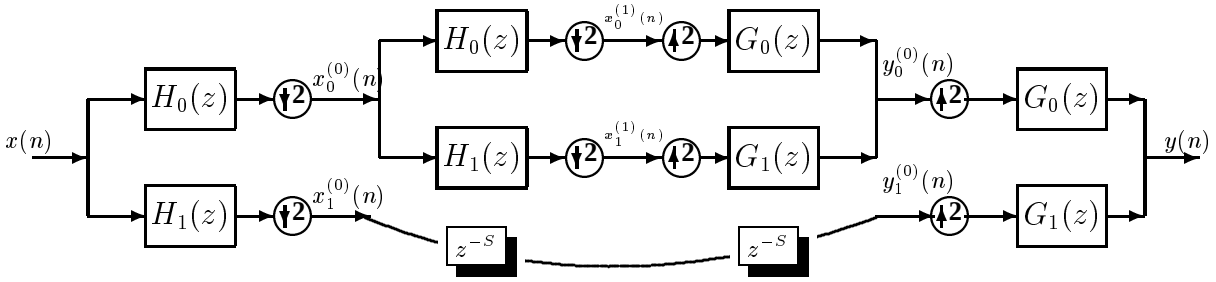

Figure 1: Synchronization buffers are needed to match the delays between the $x^{(0)}(n)$ and the $x^{(1)}$ samples.

Note that separate context information is kept for each subband. Also note that we will be concerned with both coding performance and memory needs.

The context information consists of a single line of magnitude and sign information (per subband). Sign and magnitude of each coefficient are coded separately, and different context-adaptive probability models are used for each. Refer to Figure 2 and note that only causal information is used, so that no side information is required. Let $\alpha_i, i = 0, \ldots K$ be the current *quantized* wavelet coefficient, where K is the width of a line. Let $c_i, i = 0, \ldots, K$ be the context information for magnitude in this subband. Subbands are scanned from left to right, top to bottom, and the c_i's are initially set to zero. The update of c_i after we have encoded a pixel of value α_i is as follows:

$$c_i = \begin{cases} |\alpha_i| & \alpha_i \neq 0 \\ c_i/2 & \text{otherwise} \end{cases} \quad (2)$$

Thus, if a coefficient is found to be non-zero (at the given quantization level), we keep its absolute value as the context information in the current position in the line. However if the current coefficient is zero we divide the previous context information by 2, so as to lower (but not set to zero) the neighborhood magnitude. The factor of 2 is chosen for simplicity of implementation, and there was no particular optimization involved.

Thus, as a context we are using a *weighted average* of all previous lines of wavelet coefficients. This approach is a good compromise that allow us to maintain low memory (only one line) while also taking into account previous lines in the subband. Context information c_i is kept in fixed point format, so we expect to have $c_i = 0$ in areas having many zeros, i.e., a non zero coefficient does not "propagate" more than few lines or samples in the context information.

We separately also keep context information for the sign of the coefficients from the previous and current line. Coefficients can either be positive "+", negative "-", or zero "0", and thus two bits per coefficient are needed for storage. Note that the sign we store here is that of the actual coefficient in the line in that position.

The total memory needed for storing the context information for all subbands is three equivalent image lines. We need 3 lines of length $X/2$ for the first level of decomposition, 3 lines of length $X/8$ for the second level, and in general 3 lines of length 2^{-n-1} for the n^{th} level. The total buffer size is $T_C = \sum_{i=0}^{N-1} 3 \cdot 2^{-n-1} X = 3(1 - 2^{-N})X$, which tends to $3X$ as N grows.

3.2 Classification and Encoding

Given the context information as just described we now discuss our proposed approach to assign a probability model to each possible context. Given that the c_i can take many different values, in order to avoid suffering from "context dilution", we will define a small set of context classes, and these classes, rather than each individual context, will be assigned a probability model. Our context classification is based on the sum of the magnitude of the neighboring coefficients.

In what follows we use $\texttt{encode}\{v \mid \zeta\}$, to represent encoding a number v given a discrete probability model ζ, using arithmetic coding. $\texttt{encode-raw}\{v^{[\xi]}\}$ will represent the function of sending the ξ least significant bits of v, without any form of entropy coding. We will use the function $\texttt{encode-bit}\{\mu \mid \zeta\}$, to denote encoding a single bit using the binary probability model ζ. Based on the context information c_i (see Figure 2) we classify each new coefficient α_i into a class γ_i, among 15 possible classes, as follows:

$$\beta_i = c_i + 2c_{i-1} + c_{i+1} + (c_{i-3}||c_{i-2}||c_{i+2}||c_{i+3}) \quad (3)$$

$$\gamma_i = \begin{cases} 14 & \text{if} \quad \beta_i > 2^{14} - 1 \\ 0 & \text{if} \quad \beta_i = 0 \\ 1 + \lfloor \log_2 \beta_i \rfloor & \text{otherwise} \end{cases} \quad (4)$$

Where $||$ represents the logical "or",

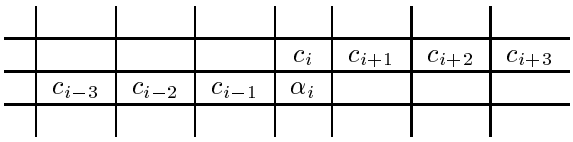

Figure 2: The context used to encode α_i, the magnitude of the current coefficient, includes the c_i's as defined in (2), which provide information about the current and previous lines.

$$\alpha||\beta = \begin{cases} 0 & \text{if} \quad \alpha = 0 \text{ and } \beta = 0 \\ 1 & \text{otherwise} \end{cases}$$

By using the $||$ operation we limit the contribution of $c_{i-3}, c_{i-2}, c_{i+2}, c_{i+3}$ to the context formation. The selection of the $1 + \lfloor \log_2 . \rfloor$ operator, is mostly for simplicity since it represents the number of significant bits. Moreover using a logarithmic rule for quantization into classes also accounts for the fact that in typical images neighborhoods with small context magnitude (β_i small) are more likely than those with high magnitude. Thus a logarithmic rule allows us to have more classes at low magnitude than at high magnitude.

Class $\gamma_i = 0$ corresponds to a coefficient where all its neighbors are zero, so that we expect $|\alpha_i|$ to be very close to zero. For values of γ_i further away from zero the distribution of $|\alpha_i|$ is much less peaked around zero. Up to 15 classes can occur, but in practice the number of classes that are used depends on the bit rate. If we happen to have large enough wavelet coefficients, or high bit rate, all 15 classes might be used, but if we are encoding at low bit rates only a portion of the 15 classes might occur. After classification we encode a bit denoting if our current coefficient α_i is significant or not, the encoding of this bit is conditioned upon the class γ_i, sending information denoting if our current coefficient α_i is significant or not corresponds to $\texttt{encode-bit}\{|\alpha_i| > 0 \big| \gamma_i\}$. If a given coefficient is found to be significant, we also encode a parameter l_i:

$$l_i = \lfloor \log_2 |\alpha_i| \rfloor \tag{5}$$

This parameter denotes the extra number of LSBs[1] needed to represent the magnitude of $|\alpha_i|$, given that $|\alpha_i| > 0$. We follow by a raw encoding of the l_i LSBs of $|\alpha_i|$. As an example if[2] $|\alpha_i| = 0000\overline{1}\underline{1101}$, $l_i = 4$. If the decoder knows the value of l_i it can then find the highest order nonzero bit of α_i, and with the transmission of l_i more bits, $|\alpha_i|$ will have been completely transmitted.

[1] Least significant bits

[2] The over-lined bit is the highest order nonzero bit, while the under-lined bits are the additional l_i LSBs that will be sent to the decoder.

The sign of α_i is subsequently encoded using a separate context modeling, based on the sign of the two nearest neighbors. Each neighbor can be either zero, positive or negative so that we have a total of nine different combinations. By using a technique known as *sign flipping* [10] we can reduce the number of classes from nine to five. In sign flipping we take advantage of certain symmetries present in the context formation. As an example, we characterize the probability of one coefficient having the same sign as its two neighbors, rather than considering separate probabilities depending on whether the neighbors are both positive or both negative.

We use an arithmetic coder that operates with different probability models, where the probability model depends on the class γ_i. For each class, we keep track of symbol occurrences so as to update the probability model. The models needed for each band are: 1) Five binary models for sign encoding. 2) 15 binary models for encoding significance information, where each model corresponds to a class γ. 3) One model $\mathcal{B}$ of up to 14 symbols for encoding the number l_i from (5). This model will be used as an M-ary model with $0 < M \leq 14$ by considering the first M symbols. As will be seen next, l_i will always be bounded by a known M for each line.

As can be deduced from the foregoing discussion, context modeling does not represent a significant memory overhead to our system as compared to the memory needed for filtering.

3.3 Algorithm

After wavelet transform and quantization, with the same dead-zone quantizer for all the coefficients, the algorithm for encoding a line corresponding to one subband can be described as follows:

1. For each new line $\texttt{encode-raw}\{L^{[15]}\}$, where $L = \lfloor \log_2(1 + \max_i |\alpha_i|) \rfloor$ is the number of bits needed to represent the largest coefficient in a row line. If $L > 0$ go to step 2 else return to step 1. This allow us to determine the maximum value for l_i and to skip lines that are all zero.

2. Classify each new wavelet coefficient α_i into a class γ_i according to equations (3) and (4).

3. $\texttt{encode-bit}\{|\alpha_i| > 0 \big| \gamma_i\}$ (Encode whether α_i is zero or not, by using the corresponding model γ_i) and update the statistics for model γ_i.

4. if $|\alpha_i| > 0$ $\Big\{$

   - $\texttt{encode}\{l_i \big| \mathcal{B}^{(M)}\}$ where $l_i = \lfloor \log_2 |\alpha_i| \rfloor$, $M = \lceil \log_2 L \rceil$, $\mathcal{B}^{(M)}$ means that we are using model $\mathcal{B}$ as an $M-$ary model, since we know that $l_i < M$.

- form a class w_i for the encoding of the sign based on the sign of the two already encoded nearest neighbors
- `encode-bit`$\{\alpha_i > 0 \Big| w_i\}$ and update the statistics for model w_i.
- `encode-raw`$\{|\alpha_i|^{[l_i]}\}$, that is we encode the l_i LSBs of $|\alpha_i|$.

$\Big\}$ else go to step 5

5. update the context information according to equation (2)

6. if not end of line go to the next pixel (step 2)

7. if not end of image go to the next line (step 1)

4 Experimental Results

In Table 2 we present PSNR results for three different images along with comparisons with algorithms in [3, 4, 5] and also JPEG [1] with arithmetic coding. We use JPEG arithmetic coding in order to have a fair comparison, given that all the wavelet coders also use arithmetic coding. Our results are competitive at a fraction of the complexity and memory utilization.

In Table 1 we present the exact memory usage for all algorithms [4, 3, 5, 1], as measured in an HP-Kayak workstation running windows NT, the memory needs of our algorithm are much closer to JPEG than any of the above mentioned algorithms. The scaling problems that occur in traditional (i.e., not memory-optimized) wavelet coders can be seen clearly by observing the numbers in Table 1. As the image size increases the memory requirements can become excessive. By contrast our approach provides a relatively modest increase in memory.

Image Size	2560×2048	3312×5120	6624×5120
Compressed	650K	2.1M	4.2M
SPIHT[3]	27M	81M	*
C/B[4]	21M	67M	92M
JPEG[1]	688K	720K	720K
VM2.0[5]	51M	97M	*
This work	850K	1.3M	1.3M
This work	(1.5M)	(3.4M)	(5.5M)

Table 1: All images were compressed at 1b/p. The numbers in parenthesis for the line based algorithm correspond to the memory needed for the algorithm plus memory for buffering of the complete bit stream.

	Rate	SPIHT[3]	C/B[4]	JPEGAR	VM2.0[5]	This work
Lena	0.125	31.10	31.32	28.45	30.93	31.05
	0.25	34.13	34.45	31.96	34.03	34.20
	0.50	37.24	37.60	35.51	37.16	37.35
	1.00	40.45	40.86	38.78	40.36	40.47
Barbara	0.125	24.84	25.39	23.69	24.87	25.20
	0.25	27.57	28.32	26.42	28.17	28.18
	0.50	31.39	32.29	30.53	31.82	31.87
	1.00	36.41	37.40	35.60	36.96	36.68
Goldhill	0.125	28.47	28.61	27.25	28.48	28.49
	0.25	30.55	30.75	29.47	30.58	30.64
	0.50	33.12	33.45	32.12	33.27	33.27
	1.00	36.54	36.95	35.57	36.81	36.66

Table 2: PSNR results, we used five levels dyadic decomposition with 9-7 tap filters.

References

[1] W. Pennebaker and J. Mitchell, *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1994.

[2] D. Lee, "New work item proposal: JPEG 2000 image coding system." ISO/IEC JTC1/SC29/WG1 N390, 1996.

[3] A. Said and W. Pearlman, "A New Fast and Efficient Image Coder Based on Set Partitioning on Hierarchical Trees," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June 1996.

[4] C. Chrysafis and A. Ortega, "Efficient Context-based Entropy Coding for Lossy Wavelet Image Compression," in *Proc. DCC'97*, (Snowbird, Utah), pp. 241–250, 1997.

[5] C. Christopoulos (Editor), "JPEG 2000 Verification Model Version 2.1," *ISO/IEC JTC/SC29/WG1*, June 1998.

[6] C. Chrysafis and A. Ortega, "Line Based Reduced Memory Wavelet Image Compression," in *Proc. DCC'98*, (Snowbird, Utah), pp. 308–407, 1998.

[7] C. Chrysafis and A. Ortega, "Line based, reduced memory, wavelet image compression," *IEEE Transactions on Image Proc.*, 1999. To appear.

[8] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform ," *IEEE Trans. Signal Processing*, vol. 42, pp. 673–676, March 1994.

[9] P. Cosman and K. Zeger, "Memory Constrained Wavelet-Based Image Coding," *Signal Processing Letters*, vol. 5, pp. 221–223, September 1998.

[10] X. Wu, "Lossless Compression of Continuous-tone Images via Contex Selection, Quantization and Modeling," *IEEE Trans. Image Proc.*, vol. 6, pp. 656–664, May 1997.

[11] "ISO/IEC JTC1/SC29/WG1 (ITU-T SG8)", "Coding of Still Pictures, JBIG, JPEG. Lossless, nearly lossless Compression, WD14495," June 1996.