A SIMPLE PICTURE CODING ALGORITHM WITH FAST RUN-LENGTH MODE [1]

J. Oliver, M. P. Malumbres

Department of Computer Engineering (DISCA)
Technical University of Valencia
Camino de Vera 17, 46017 Valencia, Spain
E-mail: {joliver, mperez}@disca.upv.es

ABSTRACT

Complex image coding algorithms have been previously proposed in order to achieve high rate/distortion performance, while features like SNR/resolution salability and error resilience are also usually considered. The complexity of these algorithms lies not only in their design but also in their computation, resulting in slow execution times.

In this paper, we present a new wavelet image coder that is extremely simple in its definition and implementation, performing faster than previous proposals. Despite its simplicity, real implementations have shown that its rate/distortion performance is within the state-of-the-art. Thus, our algorithm presents the same PSNR distortion as SPIHT and JASPER (an official JPEG2000 implementation) while both of them are slower in real executions.

Moreover, our proposal is resolution scalable and presents more robustness than SPIHT, due to its lack of inter-band dependency. In addition, a run-length mode, which improves its execution time, is described.

1. INTRODUCTION

Wavelet image encoders have been proved to be the best compression schemes in term of rate/distortion (R/D) optimization. However, one of the main drawbacks in previous wavelet image encoders [1][2][3] is their high complexity. That is mainly due to the bit plane processing, that is performed along different iterations, using a threshold that focuses on a different bit plane in each iteration. This way, it is easy to achieve an embedded bit-stream with progressive coding, since more bit planes add more SNR resolution to the image.

[1] This work was supported by the *Spanish Ministry of Science and Technology* under Grant TIC 2000-1151-c07-01 and by the *Generalitat Valenciana* research Grant CTIDIB/2002/19

Although embedded bit-stream is a nice feature in an image coder, it is not always needed and other alternatives, like resolution scalability, may be more valuable according to the final purpose. In this paper, we propose a very fast and simple algorithm that is able to encode the wavelet coefficients without performing one loop scan per bit plane. Instead of it, only one scan of the transform coefficients is needed.

In section 2, the simple wavelet image coder is presented, while it is tuned and evaluated in section 3. Section 4 introduces a run-length mode, and numerical results from real executions are attained in section 5. Finally, some conclusions are drawn in section 6.

2. A SIMPLE WAVELET IMAGE CODER

In the proposed algorithm, the quantization process is performed by two strategies: one coarser and another finer. The finer one consists in applying a scalar uniform quantization to the coefficients, and it is performed just after the DWT is applied. The coarser one is based on removing bit planes from the least significant part of the coefficients, and it is performed while the algorithm is applied. Now we can define *rplanes* as the number of less significant bits to be removed.

At encoder initialization, the maximum number of bits needed to represent the highest coefficient (*maxplane*) is calculated. This value and the *rplanes* parameter are output to the decoder. Afterwards, we initialize an adaptive arithmetic encoder that is used to encode the number of bits required by the coefficients. We encode all the coefficients. For those coefficients that require more than *rplanes* bits to be coded ($c_{i,j}<2^{rplanes}$), we use an arithmetic symbol indicating how many bits are necessary in order to encode that symbol. Only *maxplane-rplanes* symbols of this type are needed to represent this information. However, an extra symbol, called *LOWER* symbol, is required to encode those coefficients that are lower than the established threshold ($2^{rplanes}$). Notice that we say that $c_{i,j}$ is a significant coefficient when it is

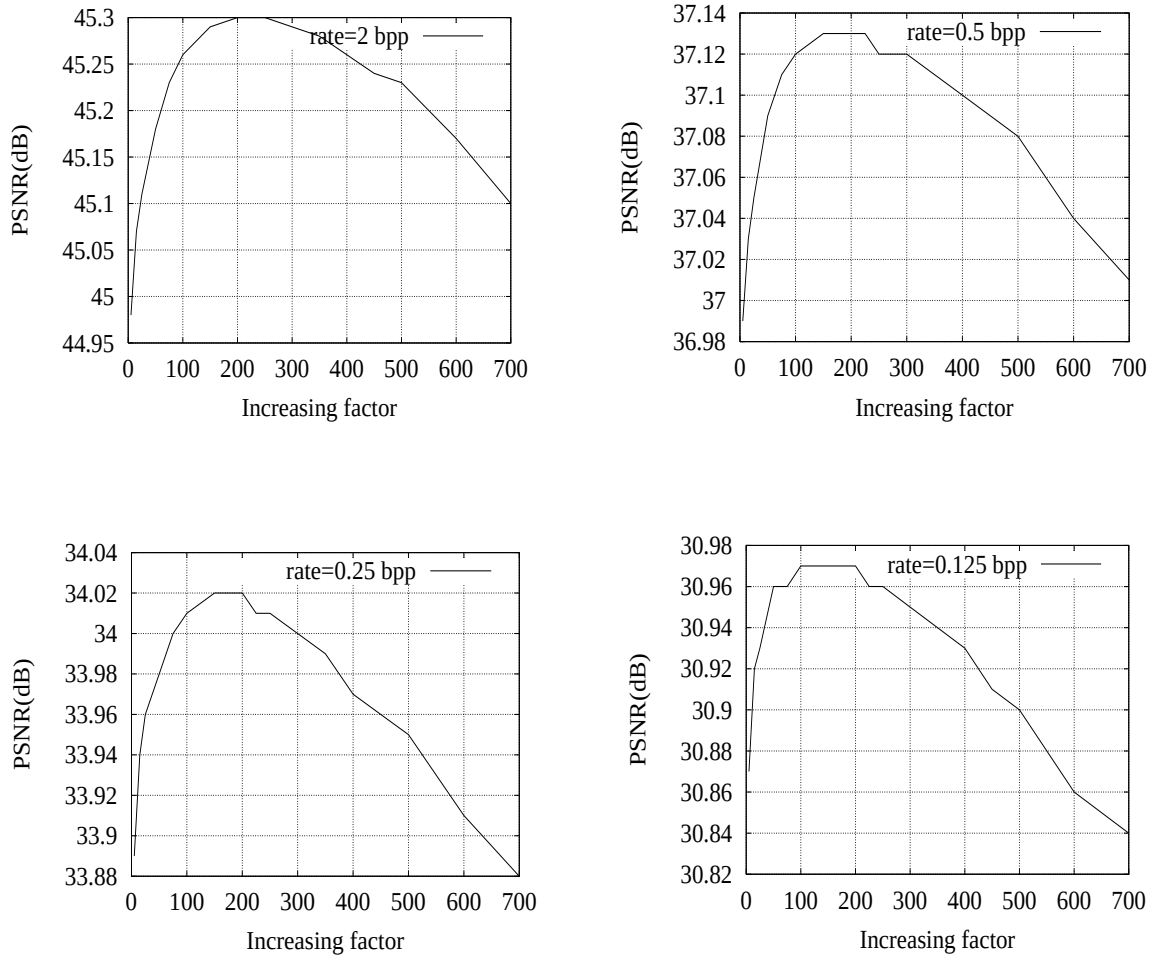

Figure 1: PSNR performance according to the increasing factor in the adaptive model, with high, medium and low bit rates

different to zero after discarding the least significant *rplanes* bits, in other words, if $c_{i,j} \geq 2^{rplanes}$.

In the next stage, the wavelet coefficients are encoded as follows. For each subband, all its coefficients are scanned. In order to preserve the locality of the information, the proposed scan order is not line-by-line but medium-sized blocks. A good block size for the scanning is the same size as the LL_N subband (the smallest and lowest-frequency subband in a dyadic decomposition). For each coefficient in a subband, if it is significant, a symbol indicating the number of bits required to represent that coefficient is arithmetically encoded. As coefficients in the same subband have similar magnitude, and due to the order we have established to scan the coefficients, the adaptive arithmetic encoder is able to represent very efficiently this information. However, we do not have enough information to reconstruct correctly the coefficient; we still need to encode its significant bits and sign.

On the other hand, if a coefficient is not significant, we should code a *LOWER* symbol, so that the decoder can determine that it has been absolutely quantized, and thus it does not have associated information (neither coefficient bits nor sign).

Notice that when encoding the bits of a significant coefficient, the first *rplanes* bits and the most significant bit are not coded, the decoder can deduce the most significant bit through the arithmetic symbol that indicates the number of bits required to encode this coefficient. Moreover, in order to speed up the execution time of the algorithm, the bits and sign of significant coefficients are raw encoded, which results in very small lost in R/D performance.

This is the encoding algorithm, *algorithm I*:

(E1) INITIALIZATION

output *rplanes*

output $maxplane = \max_{\forall c_{i,j} \in LL_N} \left\{ \left\lceil \log_2 \left(\left| c_{i,j} \right| \right) \right\rceil \right\}$

(E2) OUTPUT THE COEFFICIENTS. Scan the subbands in the established order.

For each $c_{i,j}$ in a subband

$nbits_{i,j} = \left\lceil \log_2 \left(\left| c_{i,j} \right| \right) \right\rceil$

if $nbits_{i,j} > rplanes$

arithmetic_output $nbits_{i,j}$

output $\text{bit}_{nbits_{(i,j)}-1}\left(\left| c_{i,j} \right| \right) \ldots \text{bit}_{rplane+1}\left(\left| c_{i,j} \right| \right)$

output $\text{sign}(c_{i,j})$

else

arithmetic_output *LOWER*

Note: $\text{bit}_n(c)$ is a function that returns the n^{th} bit of c.

3. TUNING THE PROPOSED ALGORITHM

Despite of the proposed algorithm simplicity, its rate/distortion performance is competitive with the state-of-the-art image coders, provided the suitable tuning of parameters is applied. In this section, we present some details on the algorithm that make it more efficient. In order to perform the desired tests, we have selected the standard Lena image as basis pattern.

An adaptive arithmetic encoder is used to efficiently encode the symbols that are output in the coding process. A regular adaptive arithmetic encoder uses a dynamic histogram to estimate the current probability of a symbol. In order to update this probability, a frequency count associated to a symbol is increased every time that it is encoded. Thus, we can consider a new parameter regarding how much the histogram is increased with every symbol; we call this parameter *increasing factor*. In the original adaptive arithmetic encoder, a value of one was used for this parameter. We have observed that if this value is greater than one, the adaptive arithmetic encoder may converge faster to local image features, leading to higher compression ratios. However, increasing it too high may turn the model (pdf estimation) inappropriate, leading to poorer performance. Besides, this parameter can be evaluated along with the *maximum frequency count*. When this value is exceeded by the sum of all the counts in the arithmetic encoder histogram (this count is called *cumulative frequency count*), all these counts are halved and thus overflow is prevented (see more details in [4]). The original proposal for this parameter is 16384 (when using 16 bits for coding), but experimental tests have led us to use a slightly lower value, 12500, so the model is halved more often.

In figure 1, we have evaluated the PSNR performance for several increasing factors, using low, medium and high compression rates (2bpp, 0.5bpp, 0.25 and 0.125bpp). This figure shows that, for the proposed maximum frequency count (12500), an optimal increasing factor is located around 200 for all the bit rates, achieving a profit of about 0.1-0.3 dB when it is compared to the original proposal (i.e., increasing only one).

As we mentioned previously, coefficients in the same subband have similar magnitude. In order to take better profit from this fact, different histograms may be handled according to the magnitude of previously coded coefficients, i.e., according to the coefficient context. In particular, we propose the use of two different contexts according to the significance of the left and upper coefficients (already encoded if a typical scan order is performed). So, if both coefficients are insignificant, the coefficient being encoded is likely to be also insignificant, and thus a specific probability model is used.

coder/ rate(bpp)	Proposed algorithm	Proposed with ctxt.	Jasper/ JPEG2000	SPIHT
2	45.30	45.30	44.62	45.07
1	40.24	40.32	40.31	40.41
0.5	36.95	37.13	37.22	37.21
0.25	33.79	34.02	34.04	34.11
0.125	30.79	30.97	30.84	31.10

Table 1: PSNR(dB)with different bit rates and coders using Lena

The benefit of using contexts is shown in table 1, where the PSNR/bit rate performance is presented for both cases (first and second columns), attaining a profit of up to 0.2 dB. On the other hand, in this table we can see that our coder is within the state-of-the-art in terms of rate/distortion performance, displaying similar PSNR results to SPIHT and Jasper [5], an official implementation of JPEG 2000 included in the ISO/IEC 15444-5.

Although our algorithm executes faster than SPIHT and Jasper (it will be shown in section 5) it presents a major drawback for low bit rate images. If we analyze the description in previous section, we can observe that all the symbols are explicitly encoded, i.e., $width \times height$ symbols are arithmetically encoded. As we know, the adaptive arithmetic encoder is one of the slower parts of an image coding system. In our algorithm, experimental results have shown that, for low bit rates, more than 3/4 part of time is expended in the arithmetic encoder system. Besides, most of those symbols being encoded have been absolutely quantized, and are always represented by the same symbol: *LOWER*.

In order to overcome this problem and to reduce the complexity in these cases, a way of grouping large streams

of *LOWER* symbols seems necessary. It will be introduced in next section.

4. FAST RUN-LENGTH MODE

In this section, a run mode is introduced in the algorithm proposed in section 2. This run mode serves to reduce complexity in the case of large number of consecutive *LOWER* symbols, which usually occurs in moderate to high compression ratios. Very minor improvements in compression performance are expected, due to the fact that we are replacing many likely symbols by a symbol or symbols that indicate the count of *LOWER*s, which will be less likely. Therefore, although less number of symbols are encoded, the probability dispersion affects adversely to the adaptive arithmetic encoder, since it works better with probability concentration.

We know that, in this new version, a run length count of *LOWER* symbols is performed, however this run mode is only applied when the *LOWER* count passes a threshold value (called *enter_run_mode* parameter). Otherwise, the compression performance of the algorithm would decrease due to the large number of run-length symbols introduced replacing short streams of the same type of symbol, the *LOWER* symbol.

When the run count is interrupted by a significant symbol, and the run value is high enough (greater than *enter_run_mode*), the value of the run length count must be output to the decoder and the run count resets.

At this point, a new symbol is introduced: the *RUN* symbol. This symbol is used to indicate that a run value is going to be encoded. After encoding a *RUN* symbol, the run count is stored in a similar way as the significant values. First, the number of bits needed to encode the run value is arithmetically output (using a different context) afterwards, the bits are raw output.

This is the new run-length version, *algorithm II*:

(E1) INITIALIZATION

output *rplanes*

output $maxplane = \max_{\forall c_{i,j} \in LL_N} \left\{ \left\lceil \log_2 \left(|c_{i,j}| \right) \right\rceil \right\}$

run_length=0

(E2) OUTPUT THE COEFFICIENTS. Scan the subbands in the established order.

For each $c_{i,j}$ in a subband

$nbits_{i,j} = \left\lceil \log_2 \left(|c_{i,j}| \right) \right\rceil$

if $nbits_{i,j} \le rplanes$

increase *run_length*

else

if $run\_length \neq 0$

if $run\_length < enter\_run\_mode$

repeat *run_length* **times**

arithmetic_output *LOWER*

else

arithmetic_output *RUN*

$rbits = \left\lceil \log_2 (run\_length) \right\rceil$

arithmetic_output *rbits*

output $\text{bit}_{rbits-1}(run\_legth) \ldots \text{bit}_1(run\_length)$

run_length=0

arithmetic_output $nbits_{i,j}$

output $\text{bit}_{nbits_{(i,j)}-1}\left(|c_{i,j}| \right) \ldots \text{bit}_{rplane+1}\left(|c_{i,j}| \right)$

output $\text{sign}(c_{i,j})$

Note: $\text{bit}_n(c)$ is a function that returns the n^{th} bit of c

Algorithms I and II are resolution scalable, due to the selected scanning order and the nature of the wavelet transform. 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 increasing the image resolution. The robustness of the proposed algorithms lies in the low dependency among the encoded information. This dependency is only present in consecutive arithmetic encoded symbols and run-length counts, and thus, the use of synchronism marks would increase the error resilience at the cost of slightly decreasing the R/D performance.

5. NUMERICAL RESULTS

We have implemented these algorithms in order to test their compression and complexity performance. The reader can easily perform new tests using the win32 version of this coder, available at *http://www.disca.upv.es/joliver/wavelet/RLW.zip*.

In order to compare our algorithm with other wavelet encoders, the standard Lena and Barbara images are used. The results for Lena using run-length mode are practically the same as those shown in table 1, version with contexts. With the correct election of the *enter_run_mode* parameter (128 in our tests), compression performance for high bit rates is certainly the same (+/- 0.01dB), and at low bit rates, very small improvement is achieved (+0.03 dB).

Table 2 shows similar compression performance comparison using a high frequency image, Barbara. In this case, our algorithm is clearly better than SPIHT but it is unable to reach the performance of JPEG 2000, due to the high number of contexts used in this standard.

We have seen that including a run-length mode has not significantly improved the compression performance. However, the main goal of these mode was reducing the complexity of the algorithm, most of all for low bit rates. We can see in tables 3 and 4, where the execution time for coding and decoding Lena is presented. This objective has

been carried out. In fact, the number of symbols arithmetically encoded at 0.125 bpp has passed from 512x512 (262144) to only 27485, and then, the execution time expended in the arithmetic encoder system has decreased from 3/4 to less than 1/3 part of the total.

In these tables, we also can observe that our final run-length proposal is up to ten times faster than Jasper/JPEG 2000 when encoding, and up to twice when decoding. Beside, compared with SPIHT, our algorithm is approx. twice faster in the coding process and up to 3.5 times faster in the decoding process.

coder/ rate(bpp)	Proposed run-length	Jasper/ JPEG2000	SPIHT
1	36.54	37.11	36.41
0.5	31.66	32.14	31.39
0.25	27.95	28.34	27.58
0.125	25.12	25.25	24.86

Table 2: PSNR (dB) with diff. bit rates and coders using Barbara

codec\ rate	SPIHT	Jasper / JPEG 2000	Proposed (with ctxt)	Proposed (run-length)
2	210.4	278.5	91.2	95.4
1	119.4	256.1	64.3	61.2
0.5	72.3	238.2	52.7	37.0
0.25	48.7	223.4	47.0	25.5
0.125	36.8	211.3	44.0	19.7

Table 3: Execution time for coding (Million of CPU cycles)

codec\ rate	SPIHT	Jasper / JPEG 2000	Proposed (with ctxt)	Proposed (run-length)
2	217.0	108.8	91.3	93.7
1	132.7	72.3	70.2	63.0
0.5	90.7	51.4	60.3	36.3
0.25	69.6	38.1	55.4	24.1
0.125	59.7	31.3	53.0	17.8

Table 4: Execution time for decoding (Million of CPU cycles)

6. CONCLUSIONS

In this paper, we have presented a new simple wavelet algorithm with run-length mode. This coder is simpler than previous proposals while its compression performance is within the state-of-the-art. Although the complexity of this algorithm is lower than others (like JPEG 2000 and SPIHT), a run-length mode is introduced in order to decrease it, especially at low bit rates. This way, we have seen that our proposal is up to 10 times faster than Jasper, and 3.5 faster than SPIHT.

Due to its lower complexity, the lack of memory overhead, possibility of robustness (no inter-band dependency) and high symmetry in coding and decoding execution times, we think that it is a good candidate for real-time interactive multimedia communications.

7. REFERENCES

[1] ISO/IEC 15444-1: "JPEG2000 image coding system," 2000.

[2] J.M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Tr.Signal Proc*, vol. 41, Dec. 1993.

[3] A. Said, A. Pearlman. "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on circuits and sys. for video tech*, vol. 6, nº 3, June 1996.

[4] I.H. Witten, R.M. Neal, J.G. Cleary, "Arithmetic coding for compression," *Commun. ACM*, vol 30. pp. 520-540, 1986.

[5] M. Adams. "Jasper Software Reference Manual (Version 1.600.0," *ISO/IEC JTC 1/SC 29/WG 1 N 2415*, Oct. 2002.