

Stack-Run Coding for Low Bit Rate Image Communication

Min-Jen Tsai, John D. Villasenor, and Feng Chen

Electrical Engineering Department
University of California, Los Angeles
405 Hilgard Avenue, Los Angeles, CA 90095, U.S.A.
E-mail: mjtsai@icsl.ucla.edu, villa@icsl.ucla.edu, fchen@icsl.ucla.edu

Abstract - We describe a new wavelet image coding approach in which a 4-ary symbol set with two contexts is used to represent significant coefficient values and the lengths of zero runs between coefficients. This algorithm works by raster scanning within subbands, and therefore involves much lower addressing complexity than other algorithms such as zerotree coding that require the creation and maintenance of intersubband dependencies across different decomposition levels. Despite its simplicity, and the fact that these dependencies are not explicitly utilized, the algorithm presented here is competitive with the recent refinements of zerotree coding. Stack-run coding can also be applied in the absence of entropy coding, resulting in an even lower complexity coder which in many cases performs within 1 dB PSNR of the entropy coded version.

1. INTRODUCTION

Many image coding algorithms include the basic steps of transformation, scalar quantization, run-length conversion and entropy coding. Recent coding advances have targeted all of these steps, either singly or in combination. For example, for the image transformation step, the advantages of wavelet transforms [1] over the block-based discrete cosine transform are now very well established. More recent wavelet coding advances have applied entropy considerations, rate-distortion optimizations, and joint space/frequency domain tradeoffs to the choice of wavelet basis and subband splitting [2, 3, 4]. Lossless coding of the integers that result from scalar quantization of transformed data is also benefitting from ongoing efforts. The simplest approach, and the one which is embodied in most of the current image and video coding standards is to perform run-length coding of the quantized data followed by Huffman coding of the run/level pairs. A more sophisticated approach which is often used in wavelet coding is to construct zerotrees [5] in which groups of zero-valued coefficients within and across subbands can be efficiently represented. Refinements of zerotree that apply rate-distortion tradeoffs in tree construction and more efficient coding of the zerotree data structures have further improved performance [4, 6].

In: *Proceedings of the International Conference on Image Processing, Vol. I*
October 1996

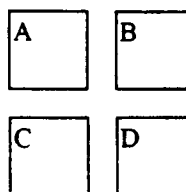
The principal result of the present paper is a new data structure for encoding quantized transformed image data that has the simultaneous advantages of being 1) very simple conceptually and computationally, and 2) giving coding performance that is better than the basic zerotree algorithm and competitive with its most recent refinements. This algorithm, which we refer to as "stack-run coding" [7], involves a wavelet transform, scalar quantization, run-length coding, and arithmetic coding, all of which are well-established techniques. The main innovation lies in the construction of a symbol set for the arithmetic coding of run/level pairs in which context information is used to enable multiple uses of a single symbol. When coupled with a simple procedure for alternating between two contexts, this yields a low complexity coding scheme which often performs within a few tenths of a dB of the most advanced enhancements of zerotree algorithms for coding of grayscale images at .25 bpp.

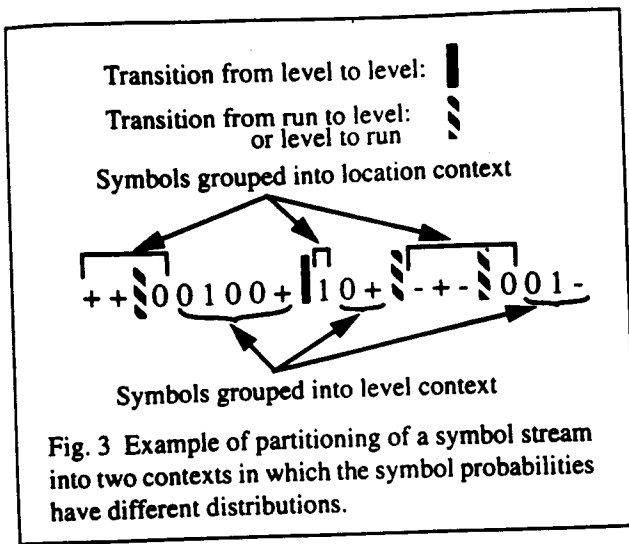
2. DESCRIPTION OF THE ALGORITHM

Consider a wavelet transformed image to which a scalar quantizer has been applied, yielding a set of integers representing transform coefficient values. Within each subband the quantized coefficients can be partitioned into two groups containing zero-valued and non-zero valued (referred to as significant) integers respectively. Let each significant coefficient be represented in binary notation as a stack, or column of bits with the MSB at the top and the LSB at the bottom. This binary representation is illustrated in Figure 1, which shows a 4 by 4 array with three significant coefficients having values 35, 4, and -11. The most significant bit in each stack is replaced by "+" if the associated coefficient is positive and "-" if it is negative. For reasons that will be explained below, the binary representation is incremented by one with respect to the coefficient value; e.g. 4 is shown as binary +01 as opposed to binary +00. A complete description of a subband can be provided by starting in one corner of the subband and performing a raster scan described in terms of pairs of the form (a,b) where a is the number of zero-valued coefficients encountered before the next significant coefficient, and b is the value of the significant coefficient.



Figure 3 A. Original image
 B. Plain VQ (0.3125 bpp)
 C. Adaptive Interpolative Coding (0.3125 bpp)
 D. VSB Interpolative Coding (0.3125 bpp)





corresponding to binary 1010 ordered from LSB to MSB. In this case, however, the final "+" in this sequence is redundant, and this run length of 10 is represented as "-+-". The final coefficient has value "-11" and is decremented to "-12" and represented as "001-". The complete symbol stream for the coefficients is "+ + 0 0 1 0 0 + 1 0 + - - - 0 0 1 -".

Given this method of mapping data into symbols, there are several further steps that can be taken to enhance performance prior to arithmetic coding of the symbol set. First, the probability tables used in the arithmetic coder can be adaptive. This will take advantage of local variations in the run length and level statistics. Second, the symbols for the run lengths and the LSB of the levels can be considered separately from the remaining symbols for the levels, resulting in two arithmetic coding contexts that can be independently adapted. Figure 3 illustrates this partitioning. The first run is of length three, which is represented as mentioned above using "+ +". Both of these symbols, as well as the "0" representing the LSB of the level are associated with the run context. Once the LSB of the level has been identified, both the encoder and the decoder are aware that the next symbol must be associated with a level. This next symbol, and all subsequent symbols up to and including the "+" or "-" that terminates the level are encoded using the level context. After the encoder encounters the last symbol in the level, the context is switched again to the run context until the appearance of another "0" or "1". Both contexts will include all four symbols, but the symbols will be represented with different frequencies in the different contexts.

3. DISCUSSION

A more careful analysis of the stack-run symbol mapping reveals that there are several factors that lead to effi-

cient coding [8]. The first is the use of a mapping based on multiple subalphabets which is optimal for a highly peaked, wide tailed pdf. Codes optimized for such pdfs degrade relatively gracefully under the pdf mismatches that inevitably occur in coding of real wavelet subbands, especially when context switching as in Figure 3 is applied. The mapping used here is one of many possible ways to losslessly represent the run/level pairs. While all lossless mappings will preserve the entropy of the underlying information source, a well designed mapping results in a symbol stream that can be more efficiently compressed using a simple (zeroth order) arithmetic coder.

The second factor is the use of a small symbol set. As Shapiro has noted in [5], the probability table in an adaptive arithmetic code will track the true symbol probabilities much more effectively when the number of occurrences of each symbol provides a meaningful estimate of the pdf.

We have applied stack-run coding to a large set of images across a wide range of bit rates. A low complexity adaptive arithmetic coder based on [9] was used to encode the symbol stream created using the approach described in the previous section. Coding results for the 512x512 grayscale "Barbara" and "Lena" images are summarized in Table 1. The wavelet filter used for the first row of the table was the 10-tap/18-tap bi-orthogonal filter given in Table 2. For the second row the 9-tap/7-tap filter [10] was employed. Table 1 also contains results for zerotree using the algorithm of Shapiro [5], for adaptive frequency-space splitting based on the work of Xiong et al [4], and for the enhanced zerotree algorithm of Said and Pearlman [6]. The filter and subband splitting used to obtain each PSNR result are also presented to allow a more meaningful comparison between the algorithms in the table.

The performance of stack-run coding is consistently 1 to 2 dB above the original zerotree algorithm of [5]. It is slightly inferior to the algorithms of [4] and [6] when coding is performed using the same DWT and filter. In general, we have found that stack-run coding is most advantageous at relatively low bit rates, where the percentage of zero-value coefficients, and therefore of zero runs of significant length, is relatively high.

Of all of the algorithms included in the table, the technique of Said and Pearlman is closest to stack-run coding in complexity. Experiments run on an HP735/125 workstation showed that for the Lena image at .25 bpp using a 6-level DWT based on the 9/7 filters, the algorithm of Said and Pearlman requires 2.08 seconds compared with 2.36 seconds for stack-run coding when the same decomposition is used [11]. These times are for a round trip encode/

| Algorithm and filter | Ref | Barbara (PSNR in dB) | | | Lena (PSNR in dB) | | |
|--|-----|----------------------|---------|---------|-------------------|---------|---------|
| | | DWT | .25 bpp | .50 bpp | DWT | .25 bpp | .50 bpp |
| Stack-run, 10/18 filter | | A | 28.90 | 32.66 | B | 33.80 | 36.89 |
| Stack-run, 9/7 filter | | B | 27.39 | 30.98 | B | 33.63 | 36.79 |
| Stack-run, 9/7 filter, no entropy coding | | B | 26.45 | 30.07 | B | 32.83 | 36.21 |
| EZW, 9-tap QMF filter | [5] | B | 26.77 | 30.53 | B | 33.17 | 36.28 |
| Space-Freq Tree, 9/7 filter | [4] | C | 27.96 | 32.25 | C | 34.35 | 37.42 |
| Said and Pearlman, 9/7 filter | [6] | A | 29.00* | 32.73* | B | 34.11 | 37.21 |

Table 1: Coding results. Decomposition A: 3-level uniform decomposition, followed by 3 more levels of octave-band splitting on the reference band (total of 73 subbands). B: 6-level octave-band DWT (total of 19 subbands). C: Adaptive subband splitting as described in ref. [4]. *Results obtained from anonymous reviewers of [7].

| | | |
|-------|---------|--|
| H_0 | 10 taps | .758907, .076790, -.157526, .000082, .028852 |
| G_0 | 18 taps | .623359, .163368, -.085661, -.013765, .030833, -.002528, -.009452, .0000272, .000954 |

Table 2: Biorthogonal wavelet filter coefficients. The right half (higher indexed coefficients) of each filter response is shown; the remaining coefficients follow by symmetry. H_0 is the low pass analysis filter; G_0 is the low pass synthesis filter.)

decode cycle and include I/O, and forward and inverse transformation, quantization, and lossless coding. While this suggests that the complexity of the two algorithms is comparable, the numbers should be interpreted with caution since neither of the implementations was optimized, and running time comparisons are vulnerable to implementation and machine-specific costs of different decision, data and memory structures. Finally, we note that a distinct disadvantage of stack-run coding relative to zerotree and related approaches is that stack-run coding does not generate an embedded bitstream. On the other hand, the fact that it operates independently on each subband is certainly advantageous for some applications.

4. CONCLUSIONS

A new image coding algorithm based on 4-ary arithmetic coding of significant coefficients and run lengths of a transformed, scalar-quantized image has been proposed. The symbol meanings are dependent on context, and are encoded using context-specific probability tables. Further savings are enabled because the MSBs of the binary run length and significant coefficient values are not explicitly encoded, but are instead implied by the structure of the symbol stream. Application of this coding scheme to commonly used test images demonstrates consistently high performance. In addition, the stack-run coding scheme described here is fast and simple to implement in hardware, and supports progressive transmission because more important subbands can be sent earlier.

REFERENCES

- [1] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," *IEEE Trans. Signal Proc.* vol. 40, pp. 2207-2232, 1992.
- [2] R.R. Coifman and M.V. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Trans. Information Theory*, vol. 38, No. 2, March 1992, pp. 713-718.
- [3] K. Ramchandran and M. Vetterli, "Best wavelet packet bases in a rate-distortion sense," *IEEE Trans. Image Processing*, vol.2, pp. 160-175, February, 1993.
- [4] Z. Xiong, C. Herley, K. Ramchandran, and M.T. Orchard, "Space-frequency quantization for a space-varying wavelet packet image coder," *Proceedings IEEE Int. Conf. on Image Processing*, vol.1, pp. 614-617, October 1995.
- [5] J.M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. Signal Processing*, vol.41, No. 12, pp. 3445-3462, Dec. 1993.
- [6] A. Said and W.A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circ. Sys. for Vid. Tech.*, June 1996.
- [7] M.J. Tsai, J. Villasenor, and F. Chen, "Stack-run image coding," to appear, *IEEE Trans. Circ. Sys. for Vid. Tech.*, 1996.
- [8] J. Villasenor, M.J. Tsai, and F. Chen, "Image compression using stack-run coding," submitted to *IEEE Transactions on Image Processing*, 1996.
- [9] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic Coding for Data Compression," *Comm. ACM*, vol. 30, pp. 520-540, June 1987.
- [10] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Proc.* vol. 1, pp. 205-220, 1992.
- [11] Weixing Zhang, Washington State University, personal correspondence.