High performance, low-complexity image compression

William A. Pearlman

Electrical, Computer and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA
E-mail: pearlman@ecse.rpi.edu

ABSTRACT

Optimal performance in a data compression scheme is very hard to obtain without inordinate computational complexity. However, there are possibilities of obtaining high performance with low complexity by proper exploitation of the characteristics of the data. Here, we shall concentrate on two recent low complexity algorithms for image compression which exploit data characteristics very efficiently. We shall try to discover principles for realizing high performance with low complexity and explain how these recent algorithms utilize these principles. We shall also present image compression results with actual codecs which realize the promise of high compression with low complexity.

Keywords: image coding, data compression

1. INTRODUCTION

Complexity of encoding has always been the main impediment in realizing truly high performance in data compression. Recently, however, there has been noticeable progress toward the goal of high compression performance with low complexity. Two recent image codecs, SPIHT (for set partitioning in hierarchical trees)[1] and AGP (for amplitude and group partitioning)[2], seem to achieve this goal. We shall concentrate on presenting their underlying principles for obtaining high compression with low complexity and on revealing other attributes which make them attractive in a coding system.

The organization of this paper is as follows. In Section 2, we present the SPIHT algorithm and its features, attributes, and results for coding images. Then, in Section 3, we present the same for the AGP method. We then conclude with a brief summary of the attributes of these two algorithms.

2. SPIHT

The set partitioning in herarchical trees (SPIHT) algorithm by Said and Pearlman [1] simultaneously utilizes several principles for obtaining high performance with reduced complexity. It features a type of classification that has been thought to be too expensive in bit resources – classification by magnitude. However, instead of requiring classification by actual magnitude, which is indeed too expensive in resources, it backs off to classification by highest significant magnitude bit, which we shall see turns out to be not only effective, but extremely inexpensive in bit resources, due to the unique characteristics of the wavelet transform of an image.

Let us assume that we have sorted the coefficients of the wavelet transform of an image by the highest significant bit in a magnitude and sign representation of their values. Furthermore, let us assume that we have ordered these coefficients from highest order to lowest order significant bit, as shown by the example in Fig. 1. Note that this is a partial ordering, as no order is imposed among those coefficients with the same highest significant bit.

Let us ignore for the moment how this ordering was accomplished. Once it is known that the highest magnitude bit of a coefficient is n, then the most number of bits to represent it exactly is the n lower order bits 0 through $n-1$, plus the sign bit. Without knowledge of the highest order bit, the upper limit on exact representation is the smaller of the precision order of the computer or the number of bits in the highest magnitude coefficient.

BIT ROW																
	sign	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
msb	5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	4		→	1	1	0	0	0	0	0	0	0	0	0	0	0
	3				→	1	1	1	1	0	0	0	0	0	0	0
	2								→	1	1	1	1	1	1	1
	1															→
lsb	0															→

Figure 1. Binary representation of the magnitude-ordered coefficients.

Now if you can obtain the number ν_n of coefficients in the class with the highest significant magnitude bit of n for all n from the largest, say n_{max}, to the smallest of 0, then you have possibilities for even greater compression, if you are willing to accept less than exact representation. Assuming that you can identify the coordinates of each coefficient with highest significant magnitude bit of n, then you need only convey ν_n to specifiy the magnitude range of each such coefficient is between 2^n and 2^{n+1}. That information plus a sign bit allows an efficient quantization. It is especially efficient in a wavelet transform where there are many coefficients of low and even zero values. The range quantization on these coefficients represents high precision and even perfect precision for the zero-valued coefficients. It turns out that the number of coefficients ν_n with bit n as the most significant does not have to be conveyed explicitly, giving even greater potential for compression.

The encoder and decoder both visit pixels in an agreed order. The encoder tests a given pixel to see whether its magnitude equals or exceeds 2^n, for some n, and, if so, will add that pixel to a list of significant pixels (LSP), as depicted in Fig. 1. If not, that pixel will not be added. If the encoder sends to the decoder the binary result of the decision to add that pixel to the LSP or not, then the decoder is informed of the encoder's action and can mimic it. Therefore, the decoder can duplicate the execution path of the decoder from the reception of a binary stream representing the significance decisions. Thus, the number ν_n of significant pixels at threshold 2^n is extracted from the binary stream of significance decisions. If a single binary decision conveys insignificance of large groups of pixels, then those pixels do not have to be visited for significance tests at the same threshold 2^n. It turns out, for coefficients of a wavelet transform, that is indeed the case, so that the significance bit stream corresponds to a small fraction in bits per pixel.

Transmissions of significance decisions would be unnecessary if one knew beforehand how to visit pixels in order of monotonically non-increasing bit significance, so that coefficients are automatically and successively ordered as they are in Fig 1. The key toward rapidly sorting coefficients according to bit significance is to partition the coefficients beforehand into sets. These sets would ideally contain coefficients with the same highest magnitude bit. Lacking such omniscience, a reasonable approach is to use the self-similarity of the multi-resolution subbands to partition into sets of coefficients with the same spatial frequency orientation, as those coefficients are likely to share the property of insignificance with respect to a given threshold. Then one can at least convey the insignificance of the entire set with transmission of a single bit. These sets are called spatial-orientation trees as their coefficients are linked by the tree structure shown in Fig. 2

The set partitioning necessitates the maintenence of three ordered lists of coefficient addresses: a list of significant pixels (LSP), as mentioned previously, a list of insignificant pixels (LIP), and a list of insignificant sets (LIS). An address or coordinate in the LIS designates that all elements in the descendant tree (excluding the root) are insignificant with respect to the current threshold. When the threshold is halved on the next iteration, the tree coefficients of LIS members are tested for significance of at least one of its members. The binary result is transmitted

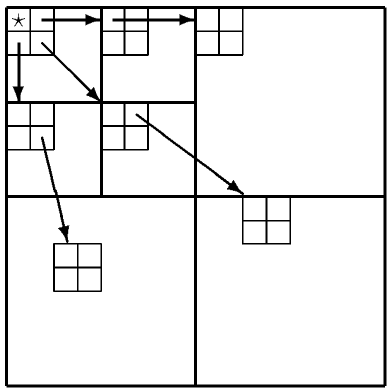

Figure 2. Examples of parent-offspring dependencies in the spatial-orientation tree.

to inform the decoder. If none of its members is signficant, the testing moves to the next member of the LIS, a new descendant tree. If there is a significant coefficient, then the tree is partitioned into its first generation (offspring) and all descendants from offspring. The offspring are then tested individually for significance, informing the decoder of the results, and their coordinates are moved to the LSP, if significant, and to the LIP, if not. The trees descending from offspring are treated as members of the LIS for further testing at the same threshold.

The order of testing, at a given bit significance, the LIP first and then the LIS is important to maintain efficient embedded coding. The significance or insignificance of individual coefficients provides information about the magnitude range of those coefficients. The corresponding positive significance decisions about sets do not provide information about individual pixels. Therefore, when the output bit stream is terminated at some given number of total bits, there are no bits wasted about set significance that would have been better put to use conveying individual coefficient significance.

A progressive bit plane transmission is all that is needed to provide true embedded coding and precise rate control. Examining Fig. 1, when one has found the coefficients in the table having bit n as the most significant (for magnitude), one can then transmit the nth bit of all those coefficients previously found significant at a larger threshold. Since n is successively decremented from its largest value n_{max}, only those bits which decrease the distortion the most (by placing a reconstruction value in the largest half-interval) are sent at each step. Although convenient, progressive bit plane transmission is not an essential aspect of the method. One could quantize and entropy code significant coefficients, perhaps obtaining better performance, but sacrificing true embedded coding and precise rate control.

2.1. Complete description of algorithm

For the sake of completeness, the complete coding algorithm (called SPIHT) is described below. For details, please refer to the original paper of Said and Pearlman[1].

The following sets of coordinates are used to present the new coding method:

- $\mathcal{O}(i,j)$: set of coordinates of all offspring of node (i,j);
- $\mathcal{D}(i,j)$: set of coordinates of all descendants of the node (i,j);
- $\mathcal{H}$: set of coordinates of all spatial orientation tree roots (nodes in the highest pyramid level);
- $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$.

For instance, except at the highest and lowest pyramid levels, we have

$$\mathcal{O}(i,j) = \{(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)\}\,. \tag{1}$$

Needed also to describe the algorithm is the definition of the functional relationship between magnitude comparisons and message bits, as follows:

$$S_n(\mathcal{T}) = \begin{cases} 1, & \max\limits_{(i,j)\in\mathcal{T}}\{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

The function $S_n(\mathcal{T})$ indicates the significance of a set of coordinates $\mathcal{T}$ at the threshold 2^n. To simplify the notation of single pixel sets, we write $S_n(\{(i,j)\})$ as $S_n(i,j)$.

THE SPIHT ALGORITHM

1. **Initialization:** output $n = \lfloor \log_2 \left(\max_{(i,j)}\{|c_{i,j}|\}\right) \rfloor$; set the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type A entries.

2. **Sorting pass:**

   (a) for each entry (i,j) in the LIP do:

      i. output $S_n(i,j)$;

      ii. if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

   (b) for each entry (i,j) in the LIS do:

      i. if the entry is of type A then

      - output $S_n(\mathcal{D}(i,j))$;
      - if $S_n(\mathcal{D}(i,j)) = 1$ then
        - for each $(k,l) \in \mathcal{O}(i,j)$ do:
          - output $S_n(k,l)$;
          - if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
          - if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
        - if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type B, and go to Step **2.b.ii**; otherwise, remove entry (i,j) from the LIS;

      ii. if the entry is of type B then

      - output $S_n(\mathcal{L}(i,j))$;
      - if $S_n(\mathcal{L}(i,j)) = 1$ then
        - add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type A;
        - remove (i,j) from the LIS.

3. **Refinement pass:** for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the n-th most significant bit of $|c_{i,j}|$;

4. **Quantization-step update:** decrement n by 1 and go to Step **2**.

The decoding algorithm is the same as above except that "output" is replaced by "input". The largest significant magnitude bit n_{max} among all coefficients is sent by the encoder and received by the decoder. Note the simplicity of the description.

2.2. Idempotency

The algorithm can be stopped at any desired output bit count and the decoder receiving the output bit stream can reconstruct the source image for that number of bits. Embedded coding means that the compressed file for a given number of output bits is a preamble of the compressed file for a larger number of output bits. If we take the reconstructed image for a a particular compressed file having a certain number of bits and re-compress it specifying the same number of bits, we obtain the same compressed file. This property is called idempotency. It means that an image can be re-compressed any number of times to the same rate without degradation. In addition, since this coding is embedded, any smaller rate image can be reconstructed from the file also without degradation from re-compression. Idempotency results because the algorithm builds the same ordered lists and inserts the same refinement bits into the same places as when the original compression stopped execution at the given number of output bits.

2.3. Color image coding

The algorithm extends easily and naturally to color or multi-spectral imagery. Let us consider a tri-stimulus color space, such as RGB, YUV, YCrCb, etc., for simplicity. Each of the three planes is separately wavelet transformed with an analysis filter bank. Now one can encode the three transform planes together by initializing the SPIHT algorithm's LIS and LIP with the co-ordinates of the top or coarsest level in all three planes. The spatial orientation trees emanating from each one of these roots are defined as before and are mutually exclusive and exhaustive. Now the algorithm can proceed as if it were one image transform. It automatically assigns the bits among the planes according to the significance of the magnitudes of their coefficients. There is no need to prescribe or calculate a bit allocation among the three planes, as is done with most color image coding methods. Furthermore, the number of decomposition levels or the filter kernels need not be the same among the three planes. It does not matter that some trees will terminate sooner than others. This characteristic allows automatic accomodation of decimated chrominance planes, which are standard in video sequence formats. For the sake of efficiency, RGB planes should not be coded directly, as there is too much redundancy among the planes in this color representation.

2.4. Complexity

The complexity of the coding algorithm itself, not counting the wavelet subband analysis and synthesis, is really small and simple. Finding the largest significant bits of coefficients or sets of coefficients is the chief arithmetic operation. Everything else is addressing, which is done by incrementing, decrementing, or bit shifts (factor of 2 integer multiplication), either individually or in combination. There are no floating point multiplications, error calculations, parameter estimations, rate allocation calculations, or searches which complicate and slow down other algorithms.

Concern about computation in the subband analysis and synthesis stages is fast diminishing, because of the emergence of fast and effective wavelet filters, which require only integer multiplication and bit shift division for their application. The simple operations inherent in the filtering, coding, and decoding should be easily and inexpensively realizable in hardware.

For certain hardware applications, memory is a critical consideration. As with any wavelet coder which works either within or across subbands, the wavelet transform, which is the size of the image, must be held in memory. In this particular coder, the LIP, LIS, and LSP lists, which contain just pixel coordinates or addresses must also be allocated memory. The sizes of the lists are directly proportional to the size of the iamge, but are usually a small fraction for typical coding rates in most images. By a clever addressing scheme whereby LIP and LIS significance of a 2x2 block is stored in a single byte, the combined size of the LIP and LIS can not exceed one quarter of the number of image bytes. The number of LSP entries, however, can be as large as the number of image pixels, but that is highly unlikely for wavelet transforms of images, even for lossless reproduction.

Image	Rate (bpp)	Coding Method			
		SPIHT-uncoded	SPIHT-arith code	LZC[6]	"Best"
Lena	0.25	33.70	34.11	34.12	34.57[7]
	0.50	36.84	37.21	37.25	37.68[7]
	0.75	38.56	39.04	-	-
	1.00	39.98	40.41	40.35	40.38[7]
Barbara	0.25	27.22	27.58	-	27.32[8] 28.29[3]
	0.50	30.94	31.40	-	30.94[8] 32.15[3]
	0.75	33.72	34.26	-	-
	1.00	35.94	36.41	-	37.03[3]
Goldhill	0.25	30.22	30.56	-	30.76[7]
	0.50	32.71	33.13	-	33.42[7]
	0.75	34.55	34.95	-	-
	1.00	36.00	36.55	-	36.96[7] 37.08[9]

Table 1. Peak signal to noise ratios (PSNR in dB) obtained with various coding methods.

2.5. Compression results

In order to assess the performance of a coding method by itself, one ought to start with the same data to be coded. In comparing methods that operate on wavelet transforms, one should start with the same transform and see how well the various methods encode this transform. Then extraneous factors such as the particular filters and the decomposition tree are equalized in all comparisons. Nevertheless, one may argue that in comparing methods with different transforms, such as DCT versus wavelet/subband, the transform is part of the method. Then there are methods which attempt to optimize jointly the transform or decomposition tree and the quantization [3,4]. In the comparisons to be shown here, we shall try as much as possible to compare wavelet methods with the same filters and decomposition tree, as we are trying to reveal the characteristics of the coding methods themselves. Also, in starting with a common image transform, relative PSNR rankings are likely to correspond to visual rankings.

Three standard monochrome test images of size 512×512 are Lena, Goldhill, and Barbara. The first two, Lena and Goldhill, seem to be the same ones used by different researchers, but there are different versions of Barbara cropped from the 720×576 JPEG standard test image. The comparative results quoted are therefore not guaranteed to be for the exact same images.

The PSNR versus rate results for SPIHT in Table 1 are obtained with a five level dyadic decomposition with biorthogonal 9/7 tap filters[5]. For SPIHT with a given image, the embedded property with exact rate control allowed the reconstruction to exact rates through corresponding truncations of the same compressed file. For the Lena image, we have also provided a list of coding and decoding times on a 120 MHz Pentium processor running under Windows 95 in Table 2.

Notice that SPIHT is comparable or superior to the best methods. The two methods deemed best do not provide embedded coding and are much more complex. The cited method of Joshi et al.[9] which obtaines the best results for Goldhill at 1.0 bpp (actual rate is 1.022 bpp) uses classification of subband statistics and trellis-coded quantization, which makes it much more complex and much slower than SPIHT. The filters for subbanding are the same biorthogonal 9/7 tap filters, but the dyadic decomposition teminates at four levels. We have also included

Rate	SPIHT- arith. code		SPIHT-uncoded	
(bpp)	enc	dec	enc	dec
0.25	0.25	0.22	0.16	0.06
0.50	0.55	0.49	0.38	0.22
0.75	0.71	0.99	0.55	0.28
1.00	1.05	1.04	0.88	0.44

Table 2. CPU times for images compression with 120 MHz Pentium processor running under Windows 95.

results of Tsai et al.[8] for stack-run coding, because it has similar complexity to SPIHT and in this case uses the same Barbara image and wavelet transform. At the two lower rates of 0.25 and 0.50 bpp, the PSNR results are slightly worse than those of SPIHT, but they are not expected to compare as well at higher rates, as the efficiency of the method depends heavily on long runs of zeroes. The results for SFQ (Space-Freguency Quantization) in [3] are indeed the highest for this same image and transform, but the method is very complex, since it tries to jointly optimize the actual rate-distortion tradeoff in the quantization and construction of zerotrees.

The other cited best method, EQ for estimation-quantization, by LoPresto et al.[7], is not as complex as the previous methods, but is still at least twice as complex as SPIHT. Again, none of these methods except SPIHT provides embedded coding and/or precise rate control.

The Layered Zero Coding of Taubman and Zakhor[6] is an embedded method, which provides performance comparable to SPIHT for the Lena image with similar complexity. There seems to be no other published results on other still images, perhaps because it was proposed as part of a video coding method.

3. AMPLITUDE AND GROUP PARTITIONING

3.1. Entropy coding methods

Recently, there have been some notable advances in efficiency for low complexity coding of already quantized images. One such advance, mentioned earlier, is stack-run coding[8], where run lengths of zeroes and quantized values are mapped into a very compact four-letter alphabet code which is then further entropy-coded with an arithmetic code. The efficiency depends on the presence of runs of zeroes, so requires a transform such as DCT or wavelet prior to quantization. Its overall complexity is about the same or perhaps slightly larger than SPIHT, due to its heavy reliance on adaptive arithmetic coding, and its performance is slightly inferior in tests on standard test images at rates of 0.25 and 0.50 bpp. The comparison will probably not stand up at higher rates, as shorter runs of zeroes occur more frequently and the entropy of these runs increase. Also stack-run coding does not deliver an embedded bit stream.

Another recently introduced method offering low complexity non-embedded coding of already quantized images is called amplitude- and group partitioning (AGP) [2]. It resembles the previous method in that it efficiently codes large groups of zeroes. However, it is really an efficient block entropy coding method, so it will work on unquantized data as well.

3.2. Amplitude partitioning

Consider the problem of lossless encoding of a block of integer data obtained from a suitable transformation (e.g., linear prediction, DCT, wavelet, etc.) and quantization of a source of data. The source need not be an image, but we will use an image as a generic example. Quantization is not entirely necessary, if the source data and transformation produce integer values. Typically, and importantly, the probability distribution of the data to be encoded is highly concentrated and peaked around zero, but with a significant number of values with large magnitude. This means

magnitude set (MS)	amplitude intervals	sign bit	magnitude-difference bits
0	[0]	no	0
1	[−1], [1]	yes	0
2	[−3,−2], [2,3]	yes	1
3	[−7,−4], [4,7]	yes	2
4	[−15,−8], [8,15]	yes	3
5	[−31,−16], [16,31]	yes	4
6	[−63,−32], [32,63]	yes	5
7	[−127,−64], [64,127]	yes	6
⋮	⋮	⋮	⋮

Table 3. Magnitude-sets used in the JPEG algorithm.

that large compression ratios are possible, but the coding process must deal with a large alphabet. In normal circumstances, it is unfeasible to consider the most efficient strategy of grouping the data symbols together as super symbols, as the extension alphabet becomes enormous. There have been attempts to deal with large alphabets. One popular method is to use an extra "overflow" symbol to indicate data to be coded with a special method. However, such methods are inefficient and have limited application.

One way to treat large alphabets is to partition it into sets of amplitude values, such as those in the JPEG standard compression algorithm. This partition is presented in Table 3. In this algorithm, the sets embrace successive octaves of magnitudes. The advantage of partitioning into amplitude sets is that a value can now be represented as a pair (set number, set index), where set number indicates the particular set, and set index indicates the particular position within the set. If one defines the sets such that inside a set the amplitudes are nearly equally probable, then the set indices need not require further entropy coding when their number is a power of 2. Then the set numbers can be entropy coded and the set indices sent in their uncoded binary representation. Note, most importantly, that the set numbers now can comprise a much smaller alphabet. For example, in the JPEG representation, a set number q represents the magnitude range of 2^{q-1} to $2^q - 1$. A 16-letter alphabet in this representation spans the magnitude range of 0 to $2^{15} - 1$.

It is possible to devise a fast converging greedy algorithm for designing amplitude set partitions whose change in entropy is least when replacing true conditional probaiblities of set indices by uniform probabilities in the corresponding set. However, it has been found that the magnitude set partitions in Table 4 are a good solution in most cases for the transforms of images.

3.3. Set number coding

Now we see that with the amplitude partitioning scheme it is only necessary to entropy code the set numbers. Hereafter, we assume sixteen partitions and set numbers indexed from 0 through 15 in non-increasing order of probability. So $\{0, 1, ..., 15\}$ can now be considered in a new source alphabet for entropy coding the set numbers. With such a drastic reduction in alphabet size, one can consider exploiting the dependence between samples by grouping or aggregating these new source symbols (set numbers) to form super symbols in a source extension with extension alphabet 16^r, where r is the number of aggregated symbols. The order r of the extension is constrained by complexity considerations, as it was with the original larger alphabet data, but not nearly to the same degree. What we propose is an adaptive method to handle the determination of extension order, so rates below the first order entropy may be achieved.

magnitude set (MS)	amplitude intervals	sign bit	magnitude-difference bits
0	[0]	no	0
1	[–1], [1]	yes	0
2	[–2], [2]	yes	0
3	[–3], [3]	yes	0
4	[–5,–4], [4,5]	yes	1
5	[–7,–6], [6,7]	yes	1
6	[–11,–8], [8,11]	yes	2
7	[–15,–12], [12,15]	yes	2
8	[–23,–16], [16,23]	yes	3
9	[–31,–24], [24,31]	yes	3
10	[–47,–32], [32,47]	yes	4
11	[–63,–48], [48,63]	yes	4
12	[–127,–64], [64,127]	yes	6
13	[–255,–128], [128,255]	yes	7
14	[–511,–256], [256,511]	yes	8
15	[–1023,–512], [512,1023]	yes	9
16	[–2047,–1024], [1024,2047]	yes	10
17	[–4095,–2048], [2048,4095]	yes	11
18	[–8191,–4096], [4096,8191]	yes	12
19	[–16383,–8192], [8192,16383]	yes	13
20	[–32767,–16384], [16384,32767]	yes	14
21	[–65535,–32768], [32768,65535]	yes	15
⋮	⋮	⋮	⋮

Table 4. Definition of the magnitude-sets used in the new image compression method.

3.4. Groups of 2x2 pixels

Let us consider encoding the set numbers, which we now call the values in a 2x2 group of pixels. A direct source extension produces an alphabet of $16^4 = 65,536$ symbols. Consider the following procedure.

1. Find the maximum value (set number) v_m in the 2x2 group of pixels.
2. Entropy code v_m; if $v_m = 0$ stop.
3. Create a binary mask μ with 4 bits, each bit corresponding to a pixel in the group. Each bit is set to 1 if the pixel value is equal to v_m, and otherwise set to 0.
4. Entropy code μ (the mask) using a 15-symbol alphabet (the case $\mu = 0$ never occurs). If $\mu = 15$ (binary [1111) or $v_m = 1$, stop.
5. Let r be the number of pixels with values smaller than v_m. If v_m^r is sufficiently small then aggregate the r symbols and entropy code them with a v_m^r symbol alphabet. Otherwise, entropy code each of the r values with a v_m symbol alphabet.
6. Stop.

Note here once again, as in SPIHT, identification of the maximum value in a group is a key factor in the compression. Of course, there exists overhead in coding that maximum value v_m in Step 2. The stop conditions are defined at the moment all values can be known at the decoder. In step 2, when the decoder is informed that $v_m = 0$, all four values in the group are known to be 0. In step 4, when mask value $\mu = 15$ is transmitted, the decoder sets all four values to v_m; and if $v_m = 1$, the binary mask contains the values of the pixels.

The binary mask prescribes that only r pixels with values smaller than v_m need to be aggregated and entropy coded. One chooses either a v_m^r or v_m symbol alphabet. So different size alphabets are used, depending on v_m. Since the maximum value is more likely to be small, the symols are often aggregated and therefore encoded to smaller than first order entropy.

The probability distribution of the binary mask values conveys one aspect of the statistical dependency between the pixels associated with the mask. In entropy-coding of the mask, one should use conditional coding to take advantage of the dependence of the mask values on the particular value of v_m. Note that entropy coding of the binary mask is always accomplished with a 15 symbol alphabet.

3.5. Groups of $2^n \times 2^n$ pixels

For a $2^n \times 2^n$ group of pixels, the 2×2 group algorithm is applied in a hierarchical and recursive manner. The steps are as follows.

1. Determine the maximum value v_m in the $2^n \times 2^n$ group.
2. Entropy code v_m.
3. Divide the $2^n \times 2^n$ group into four $2^{n-1} \times 2^{n-1}$ subgroups.
4. Define a 4-bit mask, each bit corresponding to a subgroup. Set bit to 1 if maximum in subgroup v_{ms} equals v_m, otherwise, set bit to 0.
5. Step 4 of the 2x2 group algorithm. (Entropy code the mask. Stop if $v_m = 0$ or $\mu = [1111]$).
6. For each subgroup, do
   (a) $v_m \leftarrow v_{ms}$
   (b) $n \leftarrow n - 1$
   (c) If $n > 1$, go to step 2.
   (d) If $n = 1$, do step 5 of 2x2 algorithm (encode the pixel values with v_m or v_m^r alphabets, r being the number of pixels smaller than v_m.
7. Stop.

The same algorithm proposed to code values in 2x2 groups is used to code $2^n \times 2^n$. The difference is that maximum values in subgroups are coded when $n > 1$ and pixel values are coded when $n = 1$.

Among the advantages of this extension to larger regions of the coding method are: exploitation of statistical similarity in larger regions; the automatic use of extension alphabets with a small number of elements in large regions where the values are uniformly small; and use of a single symbol to encode large zero valued regions, allowing compression to rates much smaller than 1 bit/pixel.

The exploitation of statistical dependence occurs through the series of binary masks. For a given global maximum v_m, the mask value conveys whether a maximum v_{mi} of a subgroup is less than or equal to v_m. Again, the mask value is conditionally dependent on v_m, so that conditional coding would be beneficial. As the subgroups become further divided, the same information is conveyed with respect to a certain decreasing set of global maxima. Since this set of thresholds is a global maximum at each step, the binary masks for these thresholds convey the most information about the statistical dependence of set number values among the associated pixels or groups of pixels in the mask.

3.6. A general coding scheme

What has been described is a specific example of a general coding scheme comprising the following steps:

1. Order the source symbols according to their probabilities, with "0" having the largest probability, "1" the next most probable, etc. A crude estimate of the order of probabilities is sufficient.

2. Partition the source samples into a certain number of groups. These source samples may be the direct output of the source, a transform of the source, or a quantized version of the source or transform. The groups can be defined by image regions, image subbands, spatial orientation trees, time ranges, etc.

3. Create a list with initial groups, find the maximum sample value v_m inside those groups and entropy code those numbers.

4. For each group with $v_m > 0$, do

   (a) remove the group from the list and divide it into subgroups;

   (b) entropy code a binary mask with n bits (one for each subgroup) such that a bit is set to 1 if the maximum in that subgroup v_{m_i} is equal to v_m, and otherwise set to 0;

   (c) if $v_m > 1$ then entropy code every $v_{m_i} < v_m$, possibly aggregating them to create the proper source extensions;

   (d) add to the list each subgroup with more than one element and with $v_{m_i} > 0$.

5. If the group list is empty then stop; otherwise, go to step 4.

Note that within this general scheme, there is a great deal of flexibility in the choices of rules to partition the groups, the number of groups, the entropy coding scheme, and the sizes of extensions.

This coding scheme, like SPIHT, requires simple, fast arithmetic operations and is extremely fast. The searches for maximum values are in an integer alphabet of 0 to 15 in subgroups which are not large. The partitioning and binary mask creation are even simpler and faster operations. There are no floating point multipliications or intensive iterative search operations of any kind.

3.7. Coding results

We shall now present the results of amplitude-and-group-partitioning (AGP) coding in several scenarios to show its overall flexibility, efficiency, and speed. We will use it on block DCT transformed images and wavelet transformed images. We shall consider two types of groups for partitioning: groups of pixels within subbands (denoted QT) and spatial orientation trees (denoted WV) with partitioning rules as in SPIHT. We also consider both lossy and lossless coding. Note that the entropy coding method is adaptive Huffman, which is faster than adaptive arithmetic, but slightly less efficient. A summary of the combination of different transforms and group partitioning implementations is shown in Table 5. The results for these different implementations are presented in Tables 6.

In order to obtain the cited rates for this non-embedded coding technique, several quality factors related to quantization step size had to be tried before converging on the desired rate. A significant result is that the AGP method produced the best results yet seen on block DCT's, surpassing the efforts of Xiong et al.[10,11] using SPIHT. The results for 16×16 DCT's were better than those for 8×8, especially at the lower rates. However, at these low rates, blocking effects, such as those seen in JPEG reconstructions, are still prevalent.

The wavelet transform with AGP obtained better looking results than with DCT, especially at low bit rates, despite the similarity in PSNR's. What is especially noteworthy is that the QT-10/18 method, which encodes

Method	Image transform	group partitioning	Entropy coding
WV-9/7	wavelet pyramid - 9/7 tap filters[5]	spatial orientation tree	adaptive Huffman
WV-10/18	wavelet pyramid - 10/18 tap filters[8]	spatial orientation tree	adaptive Huffman
QT-10/18	wavelet pyramid - 10/18 tap filters[8]	recursive $2^4 \times 2^4$	adaptive Huffman
DCT-8	8×8 discrete cosine	recursive $2^3 \times 2^3$	adaptive Huffman
DCT-16	16×16 discrete cosine	recursive $2^4 \times 2^4$	adaptive Huffman

Table 5. Methods implemented to test the proposed AGP coding schemes.

Image	Rate	Coding Method				
	(bpp)	DCT-8	DCT-16	QT-10/18	WV-9/7	WV-10/18
Lena	0.25	32.51	33.46	34.16	34.10	34.25
	0.50	36.18	36.82	37.23	37.21	37.30
	0.75	38.27	38.70	39.01	39.00	39.04
	1.00	39.81	40.13	40.38	40.38	40.45
Barbara	0.25	26.79	28.56	28.19	27.81	28.05
	0.50	30.86	32.48	32.17	31.61	32.48
	0.75	33.88	35.19	34.95	34.36	34.89
	1.00	36.24	37.35	37.17	36.55	37.02
Goldhill	0.25	29.89	30.41	30.50	30.53	30.60
	0.50	32.69	33.11	33.17	33.13	33.17
	0.75	34.64	34.98	35.08	34.94	35.13
	1.00	36.28	36.56	36.67	36.53	36.67

Table 6. Peak signal to noise ratios (PSNR, in dB) obtained with different realizations utilizing the alphabet and group partitioning (AGP) scheme.

subbands independently, achieves nearly the same PSNR's as the WV-10/18, which uses spatial orientation trees across subbands. The QT-10/18 method can therefore deliver progressive resolution coding.

The proper comparison to SPIHT as a coding technique is the WV-9/7 method, which uses the same filter and decomposition. The PSNR's are almost exactly the same. However, some potential advantage was yielded in favor of speed for AGP by use of Huffman code instead of arithmetic code. Referring to Tables 2 and 7 and accounting for the differences in processor speeds, the AGP method with Huffman code is roughly two times faster than SPIHT with arithmetic code in encoding and four times faster in decoding.

4. CONCLUSIONS

We have presented the principles and performance results of the SPIHT and AGP coding methods. Both deliver high performance with low complexity. SPIHT can furnish progressive fidelity with precise rate control, while AGP can deliver progressive resolution with compression controllable by a quality factor. Although not discussed fully in this paper, these methods are applicable and efficient for a wide range of rates from very low to the highest possible with lossless recovery. In these methods, one can consider ways to improve performance, but one must always be mindful of the increase in complexity and the possible destruction or impairment of other desirable attributes, such as progressive capabilities and precise control of rate.

Rate	WV - 10/18		QT - 10/18		DCT - 16	
(bpp)	enc	dec	enc	dec	enc	dec
0.25	0.16	0.03	0.20	0.03	0.22	0.03
0.50	0.21	0.07	0.23	0.05	0.25	0.06
0.75	0.26	0.10	0.25	0.07	0.27	0.07
1.00	0.31	0.13	0.28	0.09	0.30	0.10

Table 7. CPU times for compression of Lena with 133 MHz Pentium processor running under Windows NT.

REFERENCES

1. A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology* **6**, pp. 243–250, June 1996.
2. A. Said and W. A. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning," in *Visual Communications and Image Processing '97*, J. Biemond and E. J. Delp, eds., *Proc. SPIE* **3024**, pp. 25–37, 1997.
3. Z. Xiong, K. Ramchandran, and M. T. Orchard, "Wavelet packet image coding using space-frequency quantization," *submitted to IEEE Trans. on Image Processing* , 1996.
4. Z. Xiong, K. Ramchandran, and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. on Image Processing* **6**, pp. 677–693, May 1997.
5. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Processing* **1**, pp. 205–220, April 1992.
6. D. Taubman and A. Zakhor, "Multirate 3-d subband coding of video," *IEEE Trans. on Image Processing* **3**, pp. 572–588, Sept. 1994.
7. S. LoPresto, K. Ramchandran, and M. Orchard, "Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework," in *Proceedings of Data Compression Conference*, J. Storer and M. Cohn, eds., pp. 221–230, IEEE Computer Society Press, March 1997.
8. M. J. Tsai, J. D. Villasenor, and F. Chen, "Stack-run image coding," *IEEE Trans. on Circuits and Systems for Video Technology* **6**, pp. 519–521, October 1996.
9. R. Joshi, T. Fischer, and R. Bamberger, "Optimum classification in subband coding of images," in *Proc. 1997 Int. Conf. on Image Processing*, vol. 3, pp. 883–887, IEEE Computer Society Press, Nov. 1994.
10. Z. Xiong, O. Guleryuz, and M. Orchard, "A dct-based embedded image coder," *IEEE Signal Processing Letters* **3**, pp. 289–290, Nov. 1996.
11. Z. Xiong, M. T. Orchard, and K. Ramchandran, "A comparative study of dct and wavelet based coding," in *Applications of Digital Image Processing XX*, A. Tescher, ed., *Proc. SPIE* **3164**, 1997.