Embedded DCT Still Image Compression

Jiankun Li, Jin Li and C.-C. Jay Kuo, *Senior Member, IEEE*
Engineering-Systems, University of Southern California,
Los Angeles, California 90089-2564.

ABSTRACT

Motivated by both Shapiro's embedded zerotree wavelet (EZW) coding and Taubman's layered zero coding (LZC) approaches, we propose an embedded DCT still image compression scheme in this work. The new method generates a single embedded bit stream for DCT coefficients according to their importance. It is demonstrated that the new method performs better than the JPEG standard in rate-distortion tradeoff.

Keywords: Embedded coding, progressive quantization, arithmetic coder, JPEG.

I. Introduction

A typical image coding scheme consists of 3 step: transform, quantization and entropy coding. For example, in the JPEG compression standard, the DCT transform is first applied to 8×8 blocks. The resulting DCT coefficients are then quantized using a Q-table. Finally, the quantized coefficients are entropy encoded. The entropy coding is often viewed as a 2-step process. The first step converts the quantized coefficients into an intermediate sequence of symbols, while the second step converts the symbols to a bit stream in which the symbols no longer have externally identifiable boundaries. The JPEG baseline system uses the Huffman coder while the extended system use the arithmetic coder [1]. Both coders are performed using a coefficient-by-coefficient approach. That is, a coefficient has to be completely encoded before the coding of the next coefficient. To reconstruct the image, the decoder has to decode the entire bit stream. More recently, a concept known as embedded coding was proposed by Shapiro [2] and further developed by Taubman and Zakhor [3] in the context of wavelet transform coding. In contrast with the coefficient-by-coefficient approach, They adopted a new approach in which each coefficient is successively quantized into a certain number of bits. The most significant bits of all coefficients are grouped together to form one layer and encoded first. Next, we move to the layer of the second significant bits and so on. Such a coding order is consistent with the importance of each bit so that the decoder can stop at any time in decoding bit streams. The more bits are added, the closer the reconstructed image is to the original. The embedding coding method has many important applications such as rate-control, progressive image transmission and unequal error protection.

The embedded coding can also be applied to compression schemes using the block DCT transform. Even though the generalization is not difficult, we feel that it is valuable to make its detailed implementation available, which is the primary objective of the letter. Another contribution of this work is to demonstrate the performance improvement of the new method over JPEG. We show with experiments that the coding rate of the embedded DCT algorithm outperforms the JPEG Huffman coder by 30%, and the JPEG arithmetic coder by 5%.

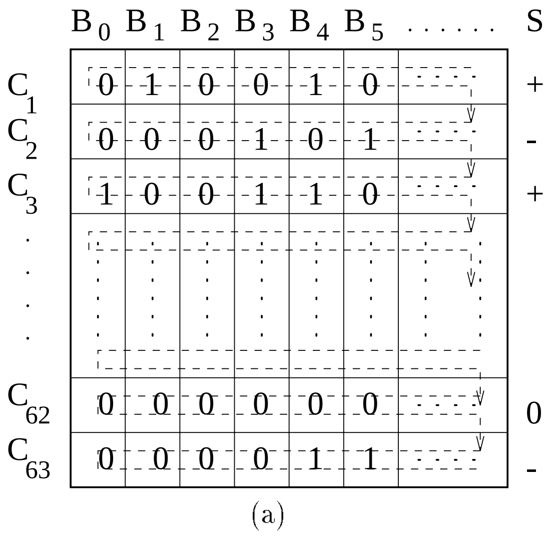
II. Embedded Coding

For a given 8×8 DCT block, let us arrange the 63 AC coefficients in the zig-zag scanning order and denote them by C_1, C_2, $\cdots$, C_{63}. Consider an example that the AC coefficient takes a value ranging from -1024 to 1024 so that it requires 11 bits for representation (including the sign bit). We label them with B_0, B_1, $\cdots$, B_{10}, where B_0 is the sign bit, B_1 the most significant bit (MSB) and B_{10} the least significant bit (LSB). The JPEG arithmetic coder encode this bit matrix in a coefficient-by-coefficient manner.

The scan order is illustrated in Fig. 1 (a). It first encodes all bits for coefficient C_1, next all bits for coefficients C_2, then C_3 and so on. For each coefficient, we perform a two-way scan. The first scan starts from B_{10} back to B_0 to locate the most significant bit B_i for the current coefficient C_j. A second scan records bits B_i, B_{i+1}, $\cdots$, B_{10} and the sign bit obtained from this scan. By this way, the number of intermediate symbols sent for arithmetic coding can be greatly reduced. In fact, the conversion of bit representations to intermediate symbols achieves a compression ratio comparable to that of the JPEG Huffman coder, and the following arithmetic coding procedure offers an additional compression ratio less than 1.5 to 1. For more details of the JPEG arithmetic coder, we refer to [1]. The major disadvantage with the above approach is that the bit streams cannot be truncated arbitrarily since such a truncation will eliminate all DCT coefficients in the bottom part (higher frequency components) and yield a reconstructed image of very poor quality. Also, rate control is difficult to implement since it is difficult to decide the number of coding bits generated for a given Q factor.

To achieve embedded coding, we adopt a different scanning order. That is, we group the bit B_i of coefficients C_j, $1 \le j \le 63$, together into a layer L_i of bits, and encode first the layer L_0 formed by the most significant bits B_0 followed by layers L_1, L_2 and so on. Within each layer L_i, the scanning order follows the coefficient order, i.e. starting with coefficient C_1, then C_2, C_3, $\cdots$, C_{63}. The scanning order is shown in Fig. 1 (b). The layer-by-layer scanning provides a better rate-distortion trade-off (see Table 2 in Section IV). Another important advantage is the embedding property. Since the output bit stream is organized in an order of decreasing importance, it is easy to perform rate control by truncating the bit stream at the desired rate.

It is worthwhile to point out that the new scanning method produces more intermediate symbols than the previous one. A scan of layer L_i generates exactly 63 intermediate symbols. Thus, there is no compression in converting bits into intermediate symbols. However, the resulting intermediate symbols can be effectively en-



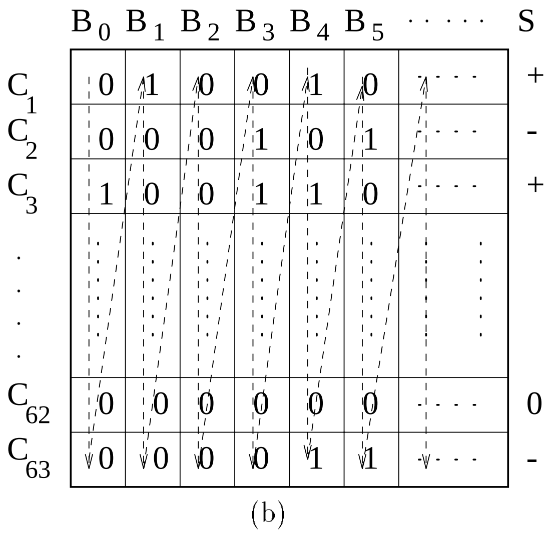


Fig. 1. Bit scanning orders for (a) JPEG and (b) the proposed embedded coding scheme.

coded by exploring the property that the arithmetic coder is very suitable for the coding of long sequences of 0's. In comparison with the compression factor of 1.5 associated with the coefficient-by-coefficient approach, the intermediate symbols are compressed by the arithmetic coder by a factor ranging from ten to several hundreds (see Table 1 in Section IV).

III. Detailed Implementation

The detailed implementation of the proposed embedded DCT compression algorithm is described in this section.

Step 1: Block DCT transform and DCT coefficient scaling

We partition the input image into 8×8 blocks, and apply the block DCT transform to each block. Furthermore, the DCT coefficients are scaled with the elements given by the quantization table Q, i.e. $V_j = \frac{C_j}{Q_j}$, $j = 0, \cdots, 63$. The scaling is performed to emphasize the visual importance of low frequency components.

Step 2: Coding of DC coefficients

The layer coding concept cannot be easily applied to DC coefficient V_0 due to differential error accumulation. Thus, we encode the DC coefficients in the same way as adopted by the JPEG arithmetic coder. An alternative is to apply a differential layer coding [3]. We observed that the performance of the two schemes were about the same.

Step 3: Successive quantization of AC coefficients

The conventional coding applies the one-step quantization right after scaling. The purpose is to map the DCT coefficient to a finite index set which can be conveniently converted to intermediate symbols and encoded by an entropy coder. In the proposed new scheme, we adopt a different approach to AC coefficient quantization. It is a successive quantization procedure using a gradually refined step size. Two symbols "0" (insignificant) or "1" (significant) are produced for each AC coefficient to form a layer of bits as the result of each quantization step. Depending whether a bit in one of the previous layers has been identified as nonzero, we consider two different rules: (1) significance identification and (2) refinement quantization.

To begin, we apply the significance identification rule only to determine the bit-layer L_0. The initial quantization step T_0 for this layer is chosen to be one half of the maximum magnitude of the scaled AC coefficients. For each scaled AC coefficient V_j, if its magnitude is greater than threshold T_0, we use symbol 1 to denote its significance and record its sign as well. Otherwise, we generate symbol 0. In terms of mathematics, we have

$$\begin{array}{lll} V_j > T_0, & S_{j,0} = 1, & E_{j,0} = V_j - 1.5 \cdot T_0, \\ V_j < -T_0, & S_{j,0} = -1, & E_{j,0} = V_j + 1.5 \cdot T_0, \\ \text{otherwise}, & S_{j,0} = 0, & E_{j,0} = V_j, \end{array}$$

for $1 \leq j \leq 63$, where symbol $E_{j,0}$ in the last column denotes the quantization residue at layer L_0. For bit layer L_{i+1}, $i = 0, 1, \cdots$, we consider a successively refined quantization step size defined by $T_{i+1} = T_i/2$. For each AC coefficient, if it is marked as insignificant in all previous layers, the significance identification rule is applied. Otherwise, the refinement quantization rule is applied. In terms of mathematics, we have

1. significance identification:

$$\begin{array}{lll} V_j > T_i, & S_{j,i} = 1, & E_{j,i} = V_{j,i} - 1.5 \cdot T_i, \\ V_j < -T_i, & S_{j,i} = -1, & E_{j,i} = V_{j,i} + 1.5 \cdot T_i, \\ \text{otherwise}, & S_{j,i} = 0, & E_{j,i} = V_j, \end{array}$$

   where symbol $E_{j,i}$ in the last column denotes the quantization residue at layer L_i.

2. refinement quantization:

$$\begin{array}{lll} E_{j,i-1} \geq 0, & R_{j,i} = 1, & E_{j,i} = E_{j,i-1} - T_i, \\ E_{j,i-1} < 0, & R_{j,i} = -1, & E_{j,i} = E_{j,i-1} + T_i. \end{array}$$

Note that symbols $S_{j,i}$ and $R_{j,i}$ are generated with a decreasing importance order to form an embedded coding bit stream.

Step 4: Context adaptive arithmetic coding

We adopt the context adaptive arithmetic coder used in the JPEG extended system to encode the significance identification symbols S_i, $i = 0, 1, 2, \cdots$ for layer L_i, and refer to [1] (Chapters 12–14) for its detailed implementation. The context used in our embedded DCT coder is illustrated in Fig. 2. For each circle position, we use 1 bit to represent the current significance status of the symbol. We combine the 6 bits to form a context for arithmetic coding to predict the current significance identification symbols S_i, consisting of 4 spatial prediction points (since spatially neighboring blocks are likely to have similar DCT coefficients) and 2 frequency prediction points (since frequency neighboring coefficients are likely to have similar scale of magnitude). Our layer-by-layer coding turns out to be more efficient than the coefficient-by-coefficient JPEG coding. This can be explained by the reason that for the layer-by-layer coding, we only predict whether a symbol is significant or not, which is much easier than the prediction of the coefficient value.

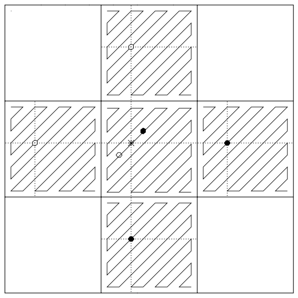


Fig. 2. Illustration of significant symbol coding context, where $*$ is the current coding position, $\diamond$ is the significant symbol of current layer and $\blacklozenge$ is the significant symbol in the previous layer.

The refinement symbol R_i is distributed evenly between 1 and -1, so we simply use an equal probability arithmetic coder.

IV. Experimental Results

In Table 1, we show some experimental data associated with each layer coding for the Lena image, which are results of Steps 3 and 4 described in the previous section. We see that the arithmetic coder does an excellent job of compressing the intermediate symbols.

Table 1. Intermediate symbols and coding results for Lena

layer	S symbol	R symbol	compressed size	compression ratio
0	258048	0	69 bytes	467:1
1	257999	49	385 bytes	83.8:1
2	257549	499	982 bytes	32.8:1
3	256234	1814	1865 bytes	17.2:1

Next, we compare our embedded DCT algorithm with the JPEG baseline Huffman coder and the JPEG arithmetic coder. For a fair comparison, we strip the header of the JPEG coding bit stream. The experimental images are Lena and Baboon of size 512×512. The results are shown in Table 2. With the same PSNR quality, the embedded DCT algorithm outperforms the JPEG Huffman coder and arithmetic coder by about 30% and 5%, respectively. Besides, our coder generates an embedded bit stream which is organized with the significant order. This property is very useful for rate-control, unequal error protection and progressive transmission.

Table 2. Performance Comparison

	PSNR	Huffman	Arith	Gain	New	Gain
Lena.512	29.22	6540	4660	28.7	4370	33.2
Baboon.512	22.65	11476	8537	25.6	7959	30.6
elaine.512	30.74	8156	6512	20.1	6111	25.1
tank.512	30.12	8002	5887	26.4	5417	32.2

References

[1] W. B. Pennebaker, *JPEG still image data compression standard* New York: Van Nostrand Reinhold, 1993.

[2] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, Vol. 41, No. 12, pp. 3445–3462, 1993.

[3] D. Taubman and A. Zakhor, "Multirate 3-D Subband Coding of Video," *IEEE Trans. on Image Processing*, Vol. 3, No. 3, pp. 572–588, 1994.