USC–SIPI REPORT #293

Embedded DCT Still Image Compression

by

Jiankun Li, Jin Li and C.-C. Jay Kuo

December 1995

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

Embedded DCT Still Image Compression

Jiankun Li, Jin Li, and C.-C. Jay Kuo*

December 4, 1995

Abstract

Motivated by 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

The JPEG baseline system uses the Huffman coder while the extended system use the arithmetic coder [1]. In both coders, one coefficient has to be completely encoded before the coding of the next coefficient. 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 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.

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

*This work was supported by the National Science Foundation Presidential Faculty Fellow Award ASC-9350309. The authors are with the Signal and Image Processing Institute and the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089-2564.

a value ranging from -1023 to 1023 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. It first encodes all bits for coefficient C_1, next all bits for coefficient 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 B_0. By this way, the number of intermediate symbols sent for arithmetic coding can be greatly reduced. In fact, the conversion 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. 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 remaining blocks and yield an incomplete image. Also, rate control is difficult to implement since it is difficult to estimate 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, $1 \leq j \leq 63$, together into layer L_i, and encode first 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 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 for each DCT block. Thus, there is no compression in converting bits into intermediate symbols. However, the resulting intermediate symbols can be effectively encoded 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 C'_j are scaled with the elements given by the quantization table Q, i.e. $C_j = \frac{C'_j}{Q_j}$, $j = 0, \cdots, 63$. 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 on whether a coefficient has been identified as significance, we apply two different rules: (1) significance identification and (2) refinement quantization.

To begin with, we apply the significance identification rule to 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 C_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 to indicate insignificance. For each advanced layer L_{i+1}, $i = 0, 1, \cdots$, we refine the quantization step size by half, i.e. $T_{i+1} = T_i/2$, and quantize all AC coefficients accordingly. For each AC coefficient, if it is 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:

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

   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

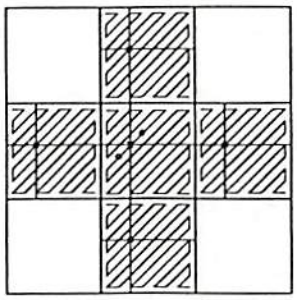

Figure 1: Illustration of significant symbol coding context, where $*$ is the current coding position, $\circ$ is the significant symbol of current layer and $\bullet$ is the significant symbol in the previous layer.

in Fig. 1. 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 symbol 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.

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 arithmetic 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. In Table 2, we fix the PSNR values of the decoded Lena and Baboon images, and list the compressed bit stream length

Table 2: Performance Comparison for Lena (with PSNR=29.22dB) and Baboon image (with PSNR=22.65dB)

	Huff.	Arith.	New	Gain (H) (%)	Gain (A) (%)
Lena	6540	4660	4370	33.2	6.2
Baboon	11476	8537	7959	30.6	6.8

(in terms of bytes) of the JPEG baseline Huffman coder, JPEG arithmetic coder and our embedded DCT coder in the first 3 columns. The gain of our new method over the other two methods is given in the last 2 columns. We see that the embedded DCT algorithm outperforms the JPEG Huffman coder and the JPEG arithmetic coder by about 30% and 6%, 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.

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.