AN EMBEDDED DCT-BASED STILL IMAGE CODING ALGORITHM

David Nister and Charilaos Christopoulos

Ericsson Telecom AB.
HF/ETX/PN/XML, CLAB
126 25 Stockholm, Sweden
Email: {d.nister, ch.christopoulos}@clab.ericsson.se

ABSTRACT

In this paper, an embedded DCT-based image coding algorithm is described. The decoder can cut the bitstream at any point and therefore reconstruct an image at lower rate. The quality of the reconstructed image at this lower rate would be the same as if the image was coded directly at that rate. The algorithm outperforms any other DCT-based coders published in the literature, including the JPEG algorithm. Moreover, our DCT-based embedded image coder gives results close to the best wavelet-based coders. The algorithm is very useful in various applications, like WWW, fast browsing of databases, etc.

1. INTRODUCTION

Transform coding has been widely used in many practical image/video compression systems. The basic idea behind using a transformation is to make the task of compressing the image after transformation easier than direct coding in the spatial domain. The Discrete Cosine Transform (DCT) has been used as the transformation in most of the coding standards as JPEG, H261/H.263 and MPEG.

In recent years most of the research activities have shifted from the DCT to the wavelet transform, especially after Shapiro published his work on embedded zerotree wavelet (EZW) image coding [7]. Although wavelets appear to be capable of providing more flexible space-frequency solutions than the DCT, it is pointed out that this is mainly because of the good structuring and quantization of the data stream, and not necessarily to the superiority of wavelet transform [1].

This paper presents a DCT-based image coding method, from which the decoded images give better results over those from JPEG and other DCT-based coders published in the literature. In addition, an embedded bitstream is produced by the encoder. The decoder can cut the bitstream at any point and therefore reconstruct an image at a lower bitrate. The quality of the reconstructed image at this lower rate would be the same as if the image was coder directly at that rate. Near lossless reconstruction of the image is possible (up to the accuracy of the DCT coefficients).

The paper is organized as follows: Section 2 describes the coding scheme. The framework of embedded coding and the scanning of the coefficients is described in sections 3 and 4. The coding of the coefficients with context based arithmetic coding is described in section 5. Results and comparisons are given in section 6 and in section 7 conclusions are drawn and topics for further research are suggested.

2. THE CODING SCHEME

The basic scheme of our DCT-based coder is:

1. Partition the image into rectangular blocks.
2. Transform each block separately with the DCT. The transformation produces a block of DCT coefficients. Traditionally the blocks used are 8 x 8 in size, but any block size of power of two. The reason to restrict the size to a power of two is that fast algorithms exist for the efficient computation of the DCT [5,8].
3. Quantize and transmit/store the DCT coefficients in a progressive manner, so that the most important information is transmitted first. This is done by successive quantization where the coding residue is reduced step by step, as it will be described in following paragraphs.

The receiver of the information can now reverse these steps. The bitstream made is embedded and the decoder can cut the bitstream at any point and generate an image that has the same quality as if it was compressed directly at that bitrate.

3. FRAMEWORK OF EMBEDDED CODING

A major objective in a progressive transmission scheme is to select the most important information - which yields the largest distortion reduction- to be transmitted first [6]. This means that the coefficients $c_{i,j}$ with largest magnitudes should be transmitted first because they have a largest content of information. This also means that the information in the value of $|c_{i,j}|$ can also be ranked according to its binary representation, and the most significant bits should be transmitted first [6].

The compression takes place mainly because after the transformation most of the energy of the image is concentrated in low frequency coefficients, and the rest of the coefficients have very low values. This means that there are very many zeroes in the most significant bit planes of the coefficients. Until the first significant bit of a certain coefficient is found, the probability of zero is very high. The task of efficient encoding therefore becomes the task of encoding these zeroes in an efficient way.

For each coefficient, we call its first non-zero bit (starting from most significant to less significant bits) as the First significant bit (FSB). The bits of a coefficient prior to the first significant bit will be referred to as the Zero bits (ZBs). The sign information is represented by the Sign bit (SB), while the rest of the bits after the first significant bit are called Raw bits (RBs). This definitions are similar to the ones used in [3].

Coding is done bitplane by bitplane. In each bitplane, the coding is from the lowest frequency coefficient to the highest frequency.

The coding algorithm is as follows:

1. Find the mean value (*DC_mean*) of all DC coefficients. Subtract this value from each DC coefficient.
2. Choose a quantizer that is half the size of the largest magnitude coefficient in the image. Transmit this quantizer.
3. Send/encode the information of which new coefficients are significant with respect to the current quantizer and also the sign of these coefficients. A coefficient is said to be significant with respect to a quantizer if its magnitude is larger than the current quantizer (in absolute terms).
4. Subtract the current quantizer (*curr_quant*) from the magnitude of the coefficients found to be significant in this bit plane. Replace the significant coefficients magnitude c_{ij} by $c_{ij} = c_{ij}$ - *curr_quant*. The difference corresponds to keeping only the raw bits.
5. For all coefficients that have been significant in previous bit planes, send/encode the information of whether the coefficients have a larger or smaller magnitude than the quantizer. Subtract the quantizer from the magnitude of the ones that do and replace those coefficients by the resulting value. This corresponds to transmitting a raw bit.
6. Divide the current quantizer by two. This corresponds to going down to a less significant bit plane of the coefficients.
7. Repeat from step 3 until the bit budget is exhausted or some desired quality is reached.

Notice that step 1 above is optional. If it used, the mean value of the DC coefficients has to be stored/transmitted. The reconstruction is done as follows:

1. Set all coefficients to zero.
2. Receive the first quantizer (*curr_quant*).
3. Receive the information about the new significant coefficients.
4. Reconstruct these as (1.5 * curr_quant * the coefficient sign) . This is because at this stage we know that the coefficient's magnitude is between *curr_quant* and (2 * *curr_quant*). This puts (1.5 * *curr_quant*) in the middle of the uncertainty interval. The addition or subtraction performed in step 5 below will update the coefficients so that they are always in the middle of the uncertainty interval.
5. For all previously significant coefficients, check if the coefficients have a larger magnitude than *curr_quant*. Add *curr_quant/2* to the magnitude of the ones that do and subtract *curr_quant/2* from the magnitude from the ones that don't.
6. Divide the *curr_quant* by two
7. Repeat from step 3 until the desired quality is reached or no more information exists.

If step (1) had been performed at the encoder, then the decoder has also received the mean value of the DC coefficients and this value is added to the reconstructed DC coefficients.

4. SCANNING ORDER OF THE COEFFICIENTS

For updating the coefficients in every bit plane a scan order needs to be defined. One coefficient is updated in all blocks before proceeding to the next coefficient.

Inside a block, the DCT coefficients are scanned in a diagonal order, bit plane by bit plane, as seen in figure 1. After each scanned diagonal, a flag is sent telling if there are any new significant coefficients in the rest of the block. This is similar to the JPEG EOB symbol and will be referred to as the block *cut_off*. The block *cut_off* is used because in the first bit planes there are so many zeroes that in practice an explicit symbol performs better than trying to code all the zeroes with a good prediction. The block cut off symbols only concern the new significant coefficients.

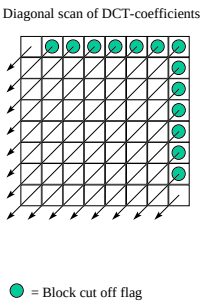

Figure 1 Scanning of coefficients in each block

As explained before a sign bit has already been sent for the previously significant coefficients. The uncertainty interval is therefore twice as big for the coefficients not yet significant and these should be considered first in the new scan. Therefore, the coding of each bitplane, first we encode the significance identification and then the refinement quantization (steps 3 and 5 of the encoding process).

During the coding of each bit plane, the scanning of the coefficients is done in the following manner: first all DC coefficients, then all AC coefficients with the same index, in the diagonal manner. The zig-zag scanning used in JPEG could also be used without affecting the property of the embedded coding that the algorithm has.

5. ARITHMETIC CODING CONTEXT

After the correct scan order has been chosen, it remains to code the scan in an efficient way. The issue is mainly how to encode the mask of the new significant coefficients. Many approaches could be taken. Zero tree coding [6,7,9], run length encoding, address switching [1]. The results presented here were achieved with context-based arithmetic coding. Context coding has also been used to code bi-level images in standards such as JBIG and also in wavelet encoders [3]. An arithmetic coder implemented according to the guidelines provided in [10] was implemented. The adaptive probability estimation

was expanded and customized for bit plane coding as described below.

Since the symbol alphabet used is binary, all that needs to be estimated for each symbol, is the probability of that symbol being zero. Without contexts, or with only one context, the probability of a zero is estimated as the number of zeroes seen so far, divided by the total number of symbols coded. Using contexts, a number of surrounding or preceding symbols are used to choose one out of several contexts. The contexts are chosen so that the symbols coded in the same context are expected to have similar statistics. In the case of binary symbols, every context holds the number of zeroes and the total number of symbols seen in this context. The probability of zero is then estimated as the number of zeroes divided by the total number of symbols. When the symbol is coded the context is updated.

Several ways of choosing the context for the symbols have been tried. Also different ways of updating the contexts. It turned out that the best approach was to restart the contexts for every new bit plane. This is due to the fact that different bit planes have different statistics and that statistics inside a bit plane is stationary enough.

5.1 Choosing the context

The bits used for the estimation of a symbol that is to be encoded are put together and considered to form an integer number. This number is used to index a context. The indexed context holds all previous statistics seen when the bits used for the estimation had this exact configuration.

In the following, the information of whether a coefficient is significant or not, is considered to be a bit plane of its own, called the significance plane. This bit plane is '1' if the coefficient in question has been found to be significant in the current bit plane or any previous bitplane.

For the raw bits of all coefficients only one context was used. This is only slightly better than sending the bits raw without entropy coding. This is also true for the AC coefficients sign bits and these were also encoded using only one context.

The DC sign bit is coded in a context chosen by adding together the number of DC neighbors that are marked in the significance plane and have a positive sign. The AC coefficients zero bits (and significant bit) are coded taking into account 6 neighboring coefficients in the block and the same coefficient in three neighboring blocks (see figure 2). The information in the significance plane for these coefficients, is used for the context.

For the DC coefficient zero bits the context is chosen using the DC coefficients in all the neighboring blocks (see figure 3). Also in this case the only thing considered is the significance plane.

The block *cut_off* is coded in the context of the *cut_off* symbols in 4 neighboring blocks (see figure 4). The diagonal number is also taken into account. This is done by using the four bits of the related cut off symbols and the diagonal, which is a 4 bit number and compose them into an 8 bit integer that indexes the context.

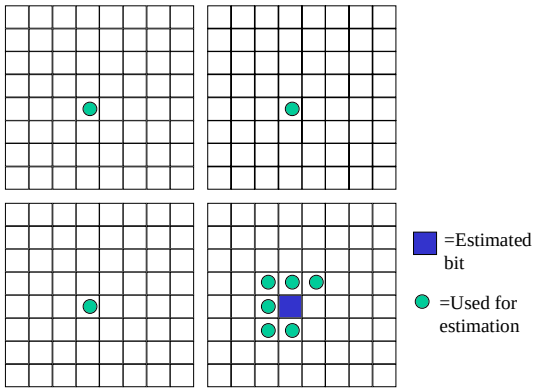

Figure 2 Choosing the context for the AC coefficients zero bits

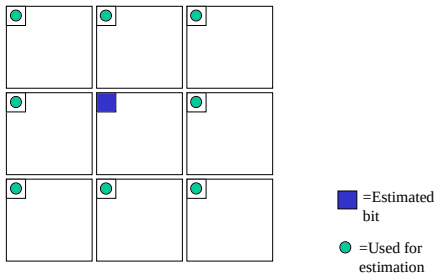

Figure 3 Choosing the context for the DC coefficients zero bits

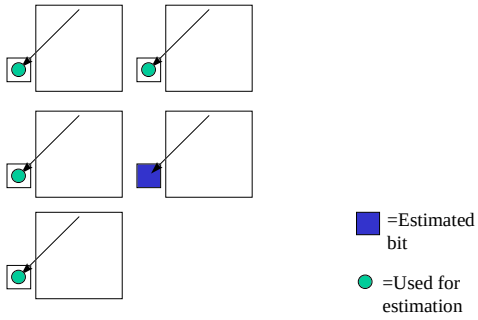

Figure 4 Choosing the context for the block cut_off

6. RESULTS AND COMPARISONS

Results for some of the tests images of JPEG 2000 are shown in figures 5 and 6. The block size used in our embedded DCT is 8x8 and 16x16. The results are also compared with JPEG as well as with a state of the art wavelet coder [6] and the embedded DCT algorithm of [9]. Notice that as in the case of our proposed algorithm, the algorithms in [6,9] produce an embedded bitstream. Notice that JPEG could not compress the images at all bit rates required. The number in brackets in

JPEG results show the actual compression ratio achieved with JPEG (the publicly available ISG JPEG software was used, without any optimizations). A trial and error procedure was tried in order to achieve the desired compression ratios with JPEG.

The results are presented in terms of Root Mean Square Error (RMSE). The RMSE as a measure of image quality is not ideal since it generally does not correlate well with the perceived image quality. Nevertheless, it is commonly used in the evaluation of compression techniques and it does provide some measure of relative performance. Clearly, as shown from the results below, the proposed algorithm outperforms JPEG significantly, especially at high compression ratios. It also gives better results compared to the embedded DCT algorithm of [9], which is based on zero-tree coding (the source code of the algorithm in [9] can be found in http://www.ee.princeton.edu/~zx/articles.html). Clearly, the performance of the proposed algorithm is very close to the performance of the state of the art wavelet coder [6] (the code for the algorithm in [6] can be found in http://ipl.rpi.edu/SPIHT).

Cr	JPEG	[9]	[6]	Prop. EDCT (8x8)	Prop. EDCT (16x16)
128:1	-	19.48	15.83	18.01	16.39
64:1	35.54 (Cr=72:1)	13.75	11.65	13.29	12.10
32:1	10.96	9.08	8.21	8.92	8.33
16:1	6.40	5.72	5.30	5.57	5.37
8:1	4.11	3.45	3.18	3.36	3.38

Figure 5 RMSE results for the image 'hotel' (Cr= compression ratio, Prop.= proposed)

Cr	JPEG	[9]	[6]	Prop. EDCT (8x8)	Prop. EDCT (16x16)
128:1	-	18.67	14.96	17.22	16.43
64:1	18.17 (Cr=57:1)	14.22	12.04	13.90	12.83
32:1	11.71 (Cr=29:1)	10.75	9.59	10.76	10.01
16:1	8.88	8.08	7.53	8.09	7.63
8:1	6.55	5.74	5.50	5.79	5.55

Figure 6 RMSE results for the image 'aerial2' (Cr= compression ratio, Prop.= proposed)

A comparison of the 8x8 and 16x16 DCT shows that there is a small improvement by the use of larger blocksize. This however might not justify the complexity of the 16x16 DCT approach. Notice that the results can improve significantly with the use of postfilters, as has been demonstrated in [4]. In this case, the results obtained are indistiguisable from the results obtained with the wavelet coders [4].

7. CONCLUSIONS AND FURTHER RESEARCH

This paper presented a low-complexity DCT-based embedded image coder that is better than JPEG and the other DCT-based coders and its performance is close to the state of the art wavelet coders. More important, we showed that by clever quantizer design, DCT is capable of delivering much better performance than JPEG, close to the performance of wavelets. Therefore, the proposed algorithm allows an elegant mechanism for backwards compatibility with current JPEG while it provides state of the art performance. Future research would be in embedded color image coding and using the ideas for coding of moving images.

8. REFERENCES

[1] N. K. Laurance and D. M. Monro, "Embedded DCT coding with significance masking", Proc. IEEE ICASSP 97, Vol. IV, pp. 2717-2720, 1997.

[2] J. Li, J. Li, C.-C. Jay Kuo, "Layered DCT still image compression", IEEE Trans. On Circuits and Systems for Video Technology, Vol. 7, No. 2, April 1997, pp. 440-442

[3] K. Nguyen-Phi and H. Weinrichter, "DWT image compression using Contextual bitplane coding of wavelet coefficients", Proc. ICASSP 97, pp. 2969-2971, 1997.

[4] D. Nister and C. Christopoulos, "An embedded DCT-based still image coding algorithm", ISO/IEC JTC1/SC29/WG1 N610, November 10-14, Sydney, Australia , 1997 (submitted).

[5] W. B. Pennebaker, and J. L. Mitchell, JPEG Still image data compression standard, Van Nostrand Reinhold, New York, 1993.

[6] 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, Vol. 6, No. 3, pp. 243-250, June 1996.

[7] J. M. Shapiro, "Embedded Image Coding using zerotrees of wavelet coefficients", IEEE Trans. on Signal Processing, Vol. 41, No. 12, pp. 3445-3462, Dec. 1993.

[8] A. N. Skodras, "Fast Discrete Cosine Transform pruning", IEEE Trans. On Signal Processing, Vol. 42, No. 7, pp. 1833-1837, July 1994.

[9] Z. Xiong, O. Guleryuz, M.T. Orchard, "A DCT-based embedded image coder", IEEE Signal Processing Letters, Vol. 3, No. 11, pp. 289-290, Nov. 1996.

[10] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression", Communications of the ACM, Vol. 30, No. 6, pp. 520-540, June 1987.