

AN EMBEDDED DCT-BASED STILL IMAGE CODING ALGORITHM

David Nister and Charilaos Christopoulos

Ericsson Telecom AB.

HF/ETX/PN/XML

Compression Lab.

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 algorithm outperforms other DCT-based coders published in the literature, including the JPEG algorithm.

EDICS: SPL.IP.1.1

1. INTRODUCTION

The Discrete Cosine Transform (DCT) has been used for the transformation in most of the coding standards as JPEG, H.261/H.263 and MPEG. Recently, advanced DCT-based coding schemes have been proposed which share many of the ideas used in wavelet-based coders [1,2,3,5].

This paper presents an embedded DCT-based image coding method, which gives better decoded images than JPEG and other DCT-based coders published in the literature. The decoder can cut the bitstream at any point and thereby reconstruct an image at a lower bitrate. The quality of the reconstructed image at this lower rate will be the same as if the image was coded directly at that rate.

2. THE CODING SCHEME

The image is partitioned into rectangular blocks of size 8×8 and each block is transformed separately with the 2D-DCT. The DCT coefficients are quantized and transmitted/stored 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 described below.

For each coefficient, we call its first non-zero bit (starting from most significant to less significant bits) 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). These definitions are similar to the ones used in [1,2].

After the transformation most of the energy of the image is concentrated in the low frequency coefficients, and the rest of the coefficients have very small values. This means that there are very many zeroes in the most significant bitplanes 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 [2].

Coding is done bitplane by bitplane. In each bitplane, the coding is done from the lowest frequency coefficient to the highest frequency. One coefficient (with the same index) is updated in all blocks before proceeding to the next coefficient. 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 - $curr_quant$ (in absolute terms).
4. Subtract the $curr_quant$ 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 $curr_quant$ 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 found 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 $(1.5 * curr_quant * \text{the coefficient sign})$. 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) has 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. Notice that in contrast with [5], the DC coefficients are included in the progressive coding method. This allows for a completely embedded bitstream and to reach very low bit rates, which is useful in progressive transmission schemes.

3. SCANNING ORDER OF THE COEFFICIENTS

Inside a block, the DCT coefficients are scanned in a diagonal order, bit plane by bit plane. 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.

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.

4. ARITHMETIC CODING CONTEXT

After the scanning order has been defined, it remains to code the scan in an efficient way. The issue is mainly how to encode the mask of the new significant coefficients. The results presented here were achieved with context-based arithmetic coding.

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 the 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 bitplane is '1' if the coefficient in question has been found to be significant in the current bitplane 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 1). 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 8 DC coefficients in all the neighboring blocks. 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 (left, above, left-above, right-above).

5. RESULTS AND COMPARISONS

Results on the JPEG 2000 test images ‘hotel’, ‘gold’ and ‘cmpnd1’ are shown in Table I. The block size used in our embedded DCT is 8x8 (results with 16x16 can be found in [1]). The results are also compared with JPEG as well as with a state of the art wavelet coder [2] and the embedded DCT algorithm of [3]. 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). Clearly, as seen by 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 [3], which is based on zero-tree coding and the performance of the proposed algorithm is very close to the performance of the state of the art wavelet coder [2].

Notice that the results can be improved significantly with a deblocking filter, as has been demonstrated in [1]. In this case, the results obtained for natural images are very close to those obtained with the wavelet coders [2]. In fact, the proposed algorithm coupled with a deblocking filter outperformed all DCT-based proposals in the first round of JPEG 2000 evaluation, and for certain images like ‘hotel’, ‘chart’, ‘gold’ ‘compound’ and ‘target’ at a bitrate of 0.25 bits per pixel was judged visually close to the best wavelet-based proposed schemes [4].

7. CONCLUSIONS

This paper presented a 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 the DCT is still capable of delivering much better performance than JPEG, close to the performance of wavelets. Therefore, the proposed algorithm allows an elegant mechanism for easy transcoding to/from

To appear in IEEE Signal Processing Letters, June 1998

current JPEG while it provides state of the art performance, coupled with the property of being a fully embedded coder.

8. ACKNOWLEDGMENTS

This work was supported by the EU ACTS project SCALAR (AC077).

9. REFERENCES

- [1] 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.
- [2] 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.
- [3] 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.
- [4] AHG on Testing, "JPEG 2000 testing results", ISO/IEC JTC1/SC29/WG1 N705, November 10-14, Sydney, Australia , 1997
- [5] J. Li, J. Li, and C.-C. J.K., "Layered DCT still image compression", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 7, No. 2, pp. 440-443, April 1997.

FIGURE AND TABLE CAPTIONS

Figure 1 Choosing the context for the AC coefficients zero bits

TABLE I: RMSE results for various images

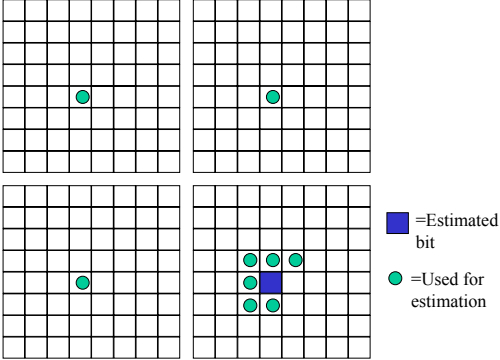


Figure 1

TABLE I

IMAGE	COMPRESSION RATIO	JPEG	[3]	[2]	PROPOSED EDCT
hotel	128:1	-	19.48	15.83	18.01
	64:1	35.54 (72:1)	13.75	11.65	13.29
	32:1	10.96	9.08	8.21	8.92
	16:1	6.40	5.72	5.30	5.57
	8:1	4.11	3.45	3.18	3.36
gold	128:1	30.66 (75:1)	13.74	11.33	12.78
	64:1	15.15 (62:1)	10.05	9.01	9.95
	32:1	8.46 (30:1)	7.54	6.92	7.40
	16:1	6.05	5.41	5.05	5.41
	8:1	4.20	3.55	3.37	3.57
cmpnd1	128:1	-	30.10	26.62	27.38
	64:1	42.11 (74.5)	24.87	19.75	22.56
	32:1	23.01 (33:1)	17.57	12.19	14.68
	16:1	12.41	9.36	5.78	7.24
	8:1	4.07	3.30	2.07	2.41