

Adaptive Block-based Image Coding with Pre-/post-filtering *

Wei Dai, Lijie Liu and Trac D. Tran

Department of Electrical and Computer Engineering
The Johns Hopkins University, Baltimore, MD 21218, USA
{wdai,ljliu,trac}@jhu.edu

Abstract

This paper presents an adaptive block-based image coding method, which combines the advantages of variable block size transform and adaptive pre-/post-filtering scheme. Our approach partitions an image into blocks with different sizes, which are best suitable for the characteristics of the underlying data in the rate-distortion (RD) sense. The adaptive block decomposition mitigates the ringing artifacts by adopting small block size transform in non-stationary regions, and improves the coding efficiency by using large block size transform in homogenous regions. Moreover, pre-/post-filtering is adaptively applied along the block boundaries to improve coding efficiency and minimize blocking artifacts. Simulation results show that the proposed coder can achieve competitive objective performance as well as yield superior reconstruction visual quality, compared with the RD-optimized JPEG2000 and H.264/AVC I-frame coder.

I. INTRODUCTION

The 8×8 block DCT has been widely used in image and video coding standards, e.g. JPEG [1] and MPEG-2 [2]. The particular block size has been believed to have the advantages of being dyadic, large enough to capture trends and periodicities while being small enough to minimize spreading effects due to local transients over the transform area [3]. Because each block is often coded independently, the typical problem with block transform coding schemes is blocking artifact, which is very annoying in smooth image areas at low bitrates. One way to avoid this problem is to use a global transform. In the latest image coding standard JPEG2000 [4], the discrete wavelet transform (DWT) is applied globally in an image which can yield much higher RD gains than JPEG and eliminate the blocking artifacts. JPEG2000 also provides resolution hierarchy, localization, SNR scalability and other interesting features. To facilitate fast practical implementations, JPEG2000 employs both block encoding and tiling, which can be interpreted as segmentation with larger data blocks. However it is very difficult to effectively make the DWT adaptive to local image statistics. All we have is the wavelet packet. Due to the long basis functions used, JPEG2000 image coding may result in ringing artifacts, especially in the image regions with rich of edges. Since a transform with short basis functions can produce fewer ringing artifacts, the smaller size block transform may work better in regions with discontinuities. The new video coding standard H.264/AVC uses a 4×4 block transform that can reduce ringing artifacts at edges and discontinuities [5]. Although H.264 is still a block-based coding scheme, it uses non-linear deblocking filters applied after the inverse block transform to eliminate the blocking artifacts. With some other new techniques, such as context-adaptive binary arithmetic coding (CABAC) and directional spatial prediction for intra block coding, H.264 can achieve an outstanding

* This work was supported by the National Science Foundation under NSF Grant No. CCR-0093262

performance in I-frame coding. Window media video 9 (WMV-9) takes the approach of allowing 8×8 blocks to be coded using either one 8×8 , two horizontally stacked 8×4 , two vertically stacked 4×8 , or four 4×4 block transforms [3]. It claims that WMV-9 has competitive RD performance with equaled or outperformed subjective quality compared with the optimized implementations of H.264/AVC [3]. Both H.264 and WMV-9 have high flexibility and adaptivity on the block level, e.g., the block size or shape can be dynamically changed with local image characteristics.

All those successful image and video coding schemes prove that smaller size transforms like 4×4 have advantages to code image regions with edges and discontinuities, while 8×8 or larger block size transforms are more efficient for trend and texture regions. Therefore, it is certainly very advantageous to be able to dynamically switch transforms of different sizes to match the local image characteristics. Actually, the idea of variable block image coding is not new. At the end of 1980's, Vaisey and Gersho did the pioneering work of image compression with variable block size segmentation [6], [7]. They first segmented an image into blocks with different sizes via a tree structure to isolate the perceptually more important image areas into smaller regions and separately identify larger texture blocks, then exploited different block coding approaches for different types of blocks. Although such method can preserve some fine region details in reconstructed images, its RD performance certainly cannot compete with the current state-of-art image coding schemes, e.g., JPEG2000 image coding.

In this paper, we propose an adaptive block transform coding scheme, which partitions a whole image into blocks with different sizes to match the underlying data characteristics in the RD sense. Pre-/post- filtering scheme [8] is adaptively applied along block boundaries to improve coding efficiency and eliminate blocking artifacts. At the encoder side, the applied pre-filter takes away the correlation between blocks, improving the DCTs energy compaction. At the decoder side, the post-filter serves as a smoothing filter eliminating blocking artifacts. Our pre- and post- filters appear in pairs and one is exactly the inverse of the other. Moreover, pre-filtering and post-filtering can be performed block-wise locally and separably just like DCT. By taking the advantages of variable block size transform and adaptive pre-/post-filtering, we demonstrate that the proposed coding scheme has superior visual reconstruction quality and competitive RD performance comparing to the RD-optimized JPEG2000 and H.264 I frame coding.

II. ADAPTIVE DCT-BASED BLOCK TRANSFORM WITH PRE/POST-FILTERING

In this section, we describe the proposed adaptive coding scheme in details on the encoder side. As shown in Fig. 1 (a), we partition an input image into 16×16 macroblocks (MBs). Each MB may be split into four 8×8 blocks. Each 8×8 block may be split further into four 4×4 basic blocks. These partitions give rise to 17 possible combinations within each MB as shown in Fig.1 (b). In each MB, the optimal splitting mode is selected first, then variable size DCT with pre-filtering are applied along block boundaries. The transform coefficients are properly coded into the final bitstream. More specifically, in section II-A, we describe how to apply the pre/post-filtering framework outside the variable size DCT. In section II-B, we explain how to code the transforms coefficients. In section II-C, we explicate how to get the optimal splitting mode based on the RD optimization criteria. Throughout the paper, we reserve (x, y) to indicate the MB position in an image and (i, j) to index the pixel position within a block.

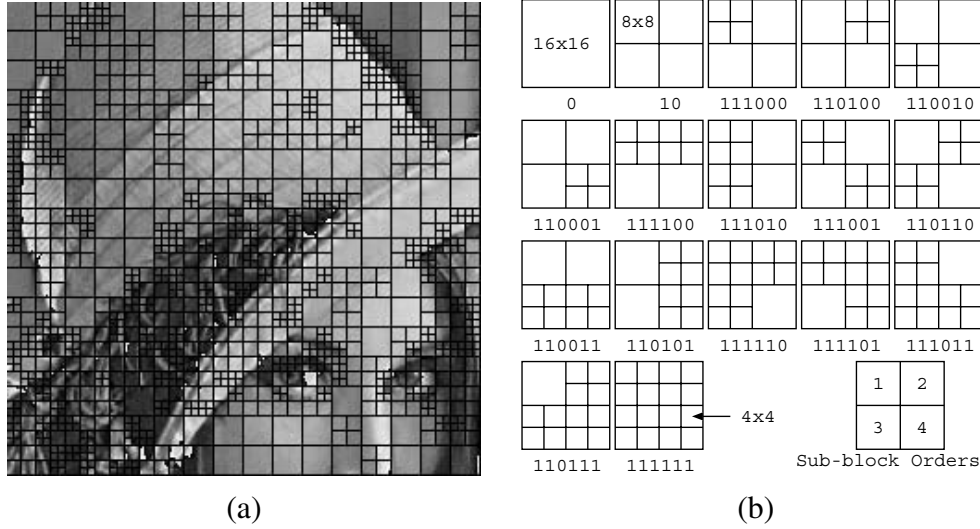


Fig. 1. (a) Splitting grid on pre-filtered Lena portion at pixel (112,64); (b) All possible macroblock partitioning modes and their associated binary representations.

A. Pre-/post-filtering

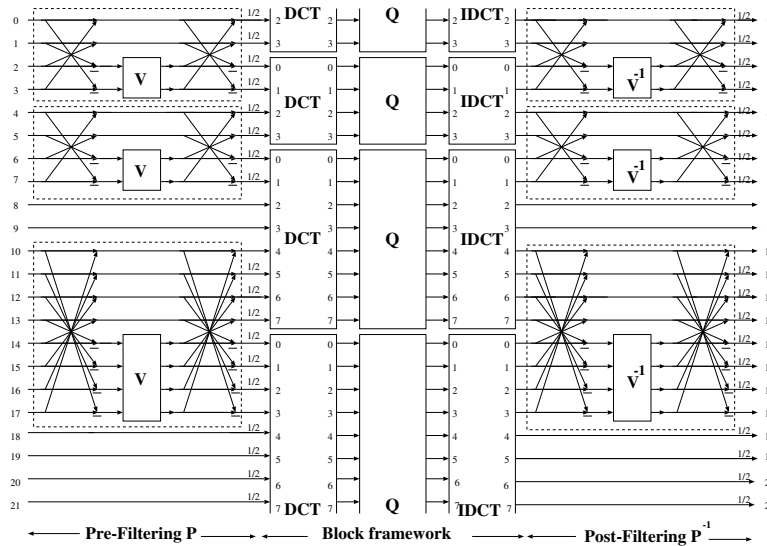


Fig. 2. Variable size DCT with adaptive borrowing.

Our one-dimensional time domain pre-/post-filtering framework is shown in Fig. 2 [8]. The pre-filter P processes the block boundaries, extracting inter-block correlation. The pre-processed samples are then fed to the DCT and encoded as usual. On the decoder side, P^{-1} serves as the post-filter, matching the signal from two sides of the block boundaries and eliminating blocking artifacts. In our framework, perfect reconstruction is structurally ensured by choosing the post-filter as the inverse of the pre-filter. As shown in Fig. 2, the linear pre-filter operator P consists of two stages of butterflies and a matrix V between them. Here, $N \times N$ matrix V is defined as $V = J C_N^{II^T} S C_N^{IV} J$, where $C_N^{II^T}$ is the N -point type-II IDCT matrix whereas C_N^{IV} is the N -point type-IV DCT matrix and J

is the reversal matrix. N also corresponds to the number of borrowing samples at block boundaries. By changing N , data samples could be pre-filtered using smaller or larger size of pre-filter P . Although at each DCT block boundary we can adapt N on the fly, this will increase the side information. To avoid sending extra bits to indicate N , we always borrow the maximal possible number of samples at each block boundary, i.e., one half of the smaller block size of two neighbor blocks. This is motivated by the fact that longer filter yields higher coding gain [9]. For example, in Fig. 2 where the block sizes are changing from 4 to 8 to 16, two samples are borrowed ($N = 2$) at the boundary of two 4×4 DCT blocks as well as at the boundary of the 4×4 and the 8×8 DCT block; while $N = 4$ is chosen at the boundary of the 8×8 and the 16×16 block.

Considering that pre-/post-filtering at block boundaries inside one MB dose not affect other MBs, we separately deal with pre-filtering at MB boundaries and pre-filtering at block boundaries inside MBs in our 2-D implementation. First, horizontal and vertical pre-filtering are performed along MB boundaries sequentially. Then, horizontal pre-filtering is applied inside the MB following by vertical pre-filtering. This process is demonstrated in Fig. 3. Pre-/post-filtering is performed in a "first-in last-out" manner, i.e., whoever pre-filtered first in the encoding process should be post-filtered last in the decoding process.

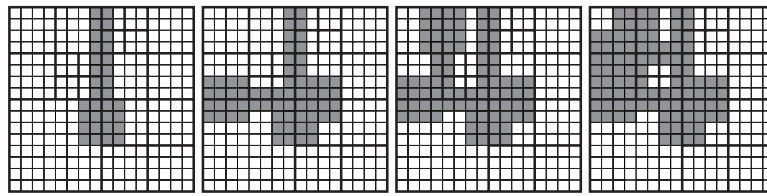


Fig. 3. From left to right: the order of horizontal and vertical pre-processing at MB boundaries and block boundaries inside the MB.

B. Prediction and Block Transform

The motivation of employing adaptive block size is trying to make the blocks in each MB as homogenous as possible. This implies that there may still exist some correlations between pixels in the pre-filtered blocks, as shown in Fig. 1 (a). Thus prediction is used to remove the remaining intra-block redundancy. In the variable block size scheme, image stationarity is reflected by the arrangement of different block sizes. There is no reason to use the DC component in a 16×16 block to estimate that of a 4×4 block, because they represent different statistics: smooth and discontinue areas respectively. We predict the pixel values in a block using its causal neighbors. The prediction value p of pixels in

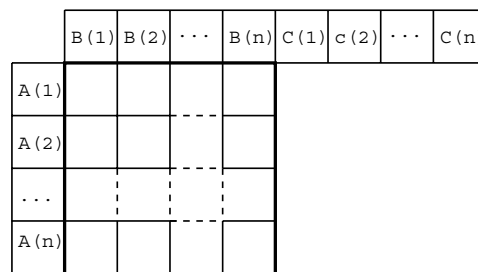


Fig. 4. Reference pixels used in $n \times n$ block prediction

a $n \times n$ block is defined as $p = \sum_{i=1}^m (A(i) + B(i) + C(i) + 0.5m)/m$, where $A(i)$, $B(i)$ and $C(i)$ are values of reference pixels illustrated in Fig. 4, and $m = |A| + |B| + |C|$, where $|\mathcal{S}|$ is defined as the number of elements in set \mathcal{S} . If \mathcal{S} does not exist i.e., $\mathcal{S} = \Phi$ is a NULL set, then $|\mathcal{S}| = 0$. After prediction, DCT, quantization and entropy coding are applied sequentially. Entropy decoding, dequantization, IDCT and compensation are then performed to reconstruct pixels $A(i)$, $B(i)$ and $C(i)$.

In each block, DCT coefficients are scanned using predetermined *zigzag* scan. The *zigzag* order is the same as that defined in JPEG. After *zigzag* scanning, resulting DCT coefficient $C^{i,j}$ at pixel position (i, j) can also be labelled as C^l , where l is the *zigzag* position at (i, j) . Define L to be the total number of coefficients in the current block. Its value dedicates by the block size, which can be 16, 64 and 256 for 4×4 , 8×8 and 16×16 blocks respectively. We assign two flags P^l and S^l to each coefficient C^l at *zigzag* scan position l . P^l indicates whether C^l is a nonzero coefficient, whereas S^l indicates whether C^l is the last nonzero coefficient (or whether there exist non-zero coefficients after C^l) in the current block. The details of block coefficients coding are described in Table I [10]:

TABLE I
BLOCK COEFFICIENTS CODING ALGORITHM

1:	Find l_{last} , the <i>zigzag</i> label of the last non-zero coefficients;
2:	Code P^0 ; code the sign and magnitude of C^0 if $P^0 = 1$;
3:	Code S^1 ; if $S^1 = 0$ then for $l = 1$ to l_{last} ,
	• code P^l ; if $P^l = 1$ then
	◊ code the sign and magnitude of C^l ;
	◊ if $l < L$ then code S^{l+1} , which is 0 if $l < l_{last}$ and 1 otherwise.

To capture the inter- and intra-block correlations, we use context-based arithmetic coding to code S^l and P^l . Let subscriptions $_{cur}$, $_{left}$ and $_{top}$ indicate the current block, the left block and the top block respectively. S^l_{cur} is coded conditioned on S^l_{left} and S^l_{top} ; the nonzero coefficient flag of $C^{i,j}_{cur}$ is coded conditioned on how many of its left block neighbor ($C^{i,j}_{left}$), top block neighbor ($C^{i,j}_{top}$), left coefficient ($C^{i-1,j}_{cur}$) within the block, and top coefficient ($C^{i,j-1}_{cur}$) within the block are found nonzero. If any of the block or coefficient neighbors lie outside of block or image boundaries, the associated S^l and P^l are set to be zeros.

C. Splitting Modes Selection and Coding

Optimal split modes are sought causally by traversing the blocks in a left-to-right, top-to-down manner. Each MB is coded using all the 17 splitting modes. The choice of the optimal mode is dictated by the RD curve: we always select the mode with the minimal RD cost. Since splitting modes are sought causally, the splitting modes on the right and at the bottom of the MB are not available when we are testing the splitting mode for MB (x, y) . To avoid this conflict, we always borrow two samples at all MB boundaries in this RD test. This is different from the final coding routine, where the maximal possible number of samples are borrowed at MB boundaries. For each possible splitting mode of a MB, the inside blocks are pre-filtered and encoded as described in the previous subsections. To calculate the RD cost, we reconstruct the blocks inside one

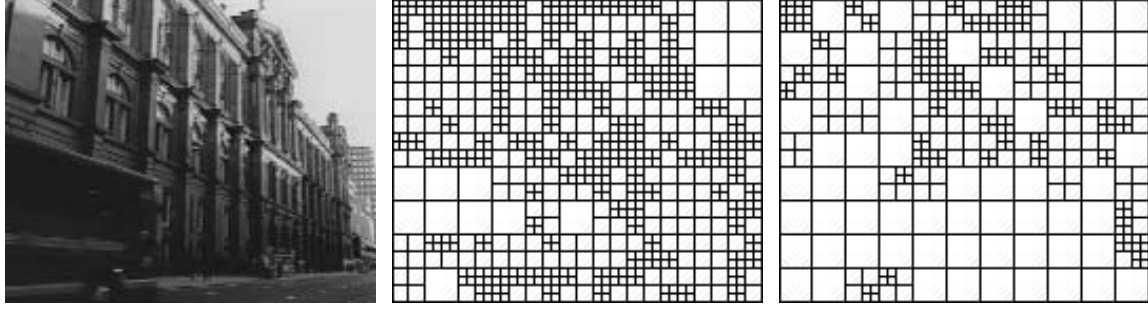


Fig. 5. Splitting grid at different bit rates. From left to right: original Glasgow image; grid at 1 bpp; grid at 0.125 bpp.

MB in a reverse order. We do not apply any post-filtering at the MB boundaries since the RD-test is performed after pre-filtering at the MB boundaries. The RD cost for one MB is $\mathcal{C} = SSD + \lambda R$, where R is the number of bits used for this MB, SSD is the sum of the squared difference of this MB, and the Lagrange parameter λ is an empirical function of quantization level q : $\lambda = 0.3q^{1.6}$. Because of the RD method used, the split mode changes with respect to bitrates. At high bitrates, the coder can afford more bits to code splitting modes. Hence, more 4×4 blocks are used to catch the discontinuity in the image. However, more bits will be allocated to code the transform coefficients at low bitrates, and large 16×16 blocks are dominant. In Fig. 5, we plot different splitting grids of the 176×144 Glasgow test image generated at 1 bpp and 0.125 bpp. At both bitrates, the smoothest areas are always coded using 16×16 blocks such as the sky at the top-right corner and the bus at the left-bottom corner. The most discontinuous areas are usually coded using 4×4 blocks to capture more details such as the middle part of the building. In all other regions, grids change with bitrates.

TABLE II
BIN STRING REPRESENTATIONS OF SPLITTING MODES

	D^0	D^1	d^1	d^2	d^3	d^4
16 × 16 block	0					
Four 8 × 8 blocks	1	0				
≥ 1 Four 4 × 4 blocks	1	1	x	x	x	x

Regarding side information, splitting mode is signalled at the MB level. Each splitting mode is represented as a binary sequence, a so-called *bin string*, as shown in Table II and Fig. 1 (b). The first bit D^0 indicates whether or not MB (x, y) is partitioned into four 8×8 blocks. If $D^0 = 1$, then the second bit D^1 indicates whether at least one 8×8 block is partitioned into four 4×4 blocks. If $D^1 = 1$, then the other four bits d^1, d^2, d^3, d^4 are required to indicate whether to split each 8×8 block into four 4×4 blocks or not. To keep the side information small, we use the context-based binary arithmetic coder to code splitting modes. Each bit in the *bin string* is coded conditioned on the number of similar partitions of its left, top and left-top neighbor blocks. For example, Fig. 1 (a) depicts the splitting modes for MB $(0, 2)$, $(1, 2)$, $(0, 3)$ and $(1, 3)$ are '0', '111100', '110101' and '111010' respectively. $D_{1,3}^0$ is coded conditioned on $D_{0,3}^0 + D_{1,2}^0 + D_{0,2}^0 = 1 + 1 + 0 = 2$ because its left and top 16×16 neighbors are split into smaller blocks; $d_{1,3}^1$ is coded

conditioned on $d_{0,2}^4 + d_{1,2}^3 + d_{0,3}^2 = 0 + 0 + 1 = 1$ as only its left 8×8 block neighbor in MB (0, 3) is split into 4×4 blocks. Other symbols are coded in the similar fashion.

D. Algorithm

The whole encoding and decoding process are summarized in Table III.

TABLE III
CODING ALGORITHM

Encoding:	
1:	<i>Initialization</i> — Reset output bitstream and all context models;
2:	<i>Splitting mode selection based on RD test</i> — For each MB(x, y) <ul style="list-style-type: none"> • Do horizontal and vertical pre-processing at MB boundaries, where two samples are borrowed; • Save MB(x, y), output bitstream, context models and bitstream length; • For $i \in \{\text{valid splitting modes}\}$ <ul style="list-style-type: none"> ◊ Code the splitting mode i; ◊ Do pre-processing on the block boundaries inside MB(x, y), where maximal samples are borrowed; ◊ For each block, do prediction, DCT, quantization, entropy coding (see Table I) and their inverse processing; ◊ Do post-processing on the block boundaries inside MB(x, y); ◊ Compute RD cost \mathcal{C}_i; ◊ Restore MB(x, y), output bitstream and context models; • Select the splitting mode i that yields the minimum RD cost \mathcal{C}_i;
3:	<i>Reinitialization</i> — Reset output bitstream and all context models;
4:	<i>Main pass</i> — The encoder codes each MB sequentially from left to right and top to bottom. For each MB(x, y) <ul style="list-style-type: none"> • Code the splitting mode • Do pre-processing at the current MB(x, y) boundary and block boundaries inside MB(x, y), where maximal possible number of samples are borrowed; • For each block, do prediction, DCT, quantization, entropy coding (see Table I).
Decoding:	
1:	<i>Initialization</i> — Reset input bitstream and all context models;
2:	<i>Main pass</i> — For each MB(x, y) <ul style="list-style-type: none"> • Decode the splitting mode; • For each sub-block, do entropy decoding, dequantization, IDCT and compensation; • Do post-processing on the sub-block boundaries within MB(x, y) and current MB(x, y) boundaries, where maximal possible number of samples are borrowed.

III. IMAGE CODING EXPERIMENTS

In this section, we demonstrate the competitive coding efficiency of the proposed adaptive framework via several image coding examples. The RD performance (PSNR in dB verse bpp) is shown in Fig. 6. The test images are monochrome 8-bit gray scale images: 176×144 Glasgow, 512×512 Barbara, 1280×720 City, 2048×2560 Woman, 3072×2048

Cats. and 768×768 Fingerprint. Glasgow and City are the first luminance frame of H.264/AVC test sequences. Barbara and Woman are test images that have been widely used by image coding community. And the miscellaneous images Cats and Fingerprint are used in parts for the JPEG2000 standardization work [11]. We compare our adaptive block based coding scheme with two codecs: JPEG2000 in the RD-optimized non-scalable single layer (SL) mode [4] and I-frame codec of H.264 with the RD optimization [5]. To be fair, the 114-byte header of JPEG2000 has been compensated when calculating the coded bitrates. In our experiments, we extract the I-frame coding routine from H.264-JM [12] to compress the test images. RD-optimization and CABAC entropy coding are used to ensure its best performance. From Fig. 6, it can be observed that the proposed adaptive coder has great advantages on complex nonhomogeneous images such as City, which captures an overview of the New York City with the Empire State Building in the middle. In such a busy image, it is difficult to do prediction well. JPEG2000 also begins to lose its advantage because of the lack of large smooth areas. On the other hand, the finer flexible grids and adaptive pre-/post-filtering of our algorithm yield superior image quality and PSNR which is evident in Fig. 7. Two other images that we perform well on are Barbara and Fingerprint, both contains a lot of texture information. Especially, Fingerprint has periodic texture, stationary characteristics which our large block size transform can better capture. Our codec consistently improves the PSNR by more than 1dB on Barbara and 0.3-1dB on City.

IV. CONCLUSION AND FUTURE WORK

In this paper, we demonstrate that the proposed adaptive block-based image coder with pre-/post-filtering can achieve competitive RD performance comparing to state-of-the-art JPEG2000 and H.264/AVC I-frame coders. Our coder can choose suitable block sizes on the fly in the RD sense according to the underlying data statistics. The choice of the partitioning has a significant impact not only on compression performances but also in the reconstruction visual quality. The combination of adaptive block size transform and adaptive pre-/post-filtering is superior at preserving edges, capturing periodic textures as well as stationary areas. Our novel codec yields consistent superior objective coding performance along with subjective visual quality: reconstructed images have minimal blocking and ringing artifacts.

The future work to improve our codec includes two folds: fast mode selection strategy and error resilient coding. The complexity of the proposed algorithm is higher than JPEG2000, while lower than H.264 with RD optimization. However, fast adaptive strategy is achievable. A lot of experiments show that smaller blocks are constantly along edges. This motivates a possible fast mode selection strategy: choosing the splitting mode by evaluating the edge strength using a simple edge detector. In our coding scheme, the visual information of a block is distributed into several different blocks because of the *overlapped* nature of the pre-filtering operators. If one block is lost, partial information of that block is still salvageable at the decoder. Thus pre-/post-filtering will be beneficial in recovery of lost data.

REFERENCES

- [1] W. Pennebaker and J. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.
- [2] J. Mitchell, W. Pennebaker, C. Frogg, and D. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, New York, 1996.

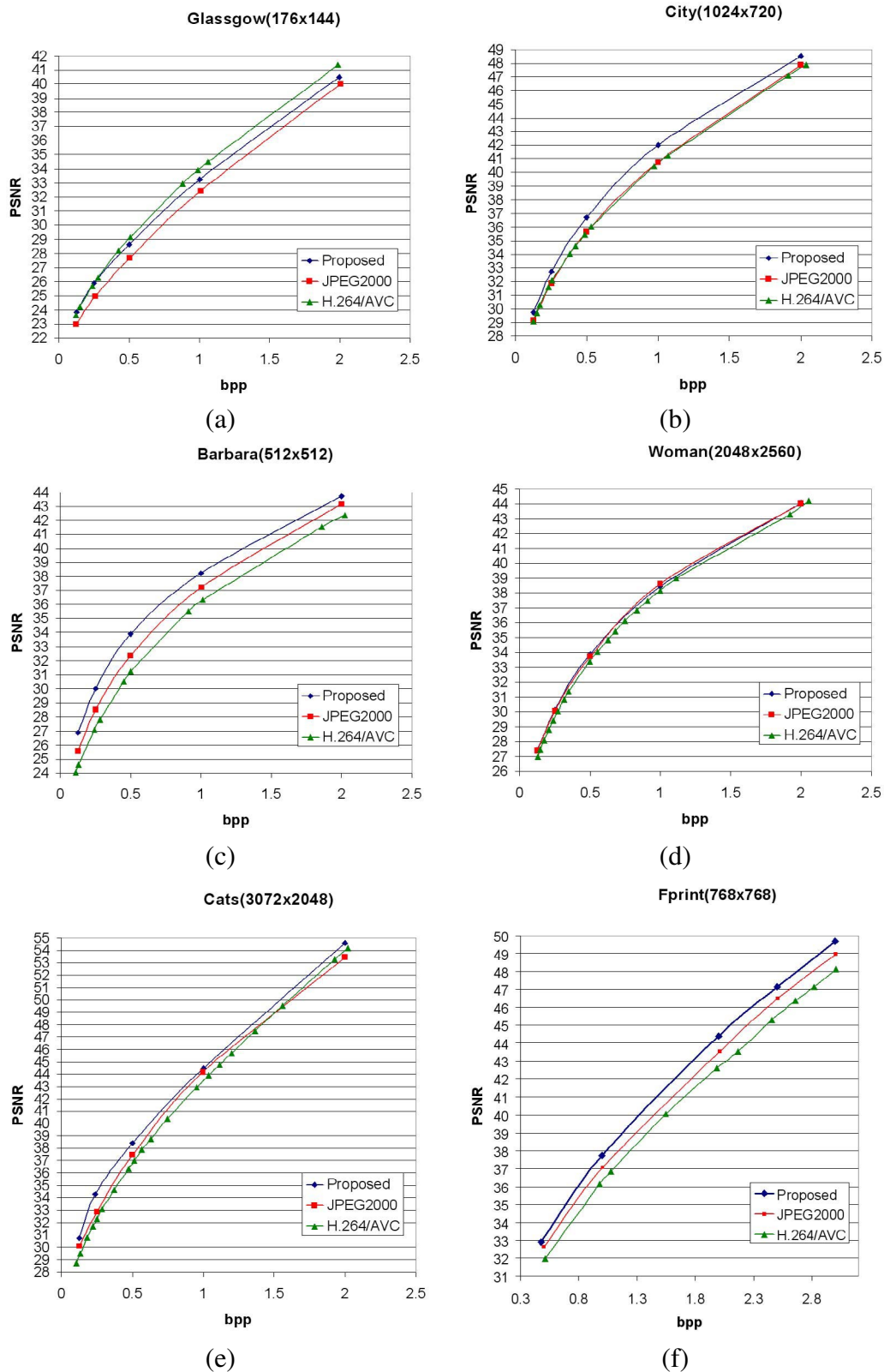


Fig. 6. Comparisons of objective coding performance: PSNR in dB versus bits per pixel (bpp).



Fig. 7. 256×256 portion at (480,416) of the City image coded at 0.17bpp. Clockwise: original image; coded by JPEG 2000; coded by H.264-JM; coded by the proposed adaptive block transform with pre/post-filtering.

- [3] Sridhar Srinivasan, Pohsiang (John) Hsu, Tom Holcomb, Kunal Mukerjee, Shankar L. Regunathan, Bruce Lin, Jie Liang, Ming-Chieh Lee, and Jordi Ribas-Corbera, "Windows media video 9: Overview and applications," *Signal Processing: Image Communication*, vol. 19, pp. 851–875, October 2004.
- [4] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer Academic Publishers, 2001.
- [5] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264AVC video coding standard," *IEEE Trans. on Circuits and Systems For Video Technology*, vol. 13, pp. 560–576, July 2003.
- [6] J. Vaisey and A. Gersho, "Variable block-size image coding," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Dallas, TX, 1987, pp. 1051–1054.
- [7] J. Vaisey and A. Gersho, "Image compression with variable block size segmentation," *IEEE Trans. Signal Processing*, vol. 40, no. 8, pp. 2040–2060, Aug. 1992.
- [8] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-filtering," *IEEE Transaction on Signal Processing*, vol. 51, pp. 1557–1571, Jun. 2003.
- [9] S. Gangaputra and T. D. Tran, "Adaptive pre- and post-filtering for block based systems," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Orlando, 2002, vol. 4, pp. 3297–3300.
- [10] C. Tu and T. D. Tran, "Context based entropy coding of block transform coefficients for image compression," *IEEE Trans. on Image Processing*, vol. 11, pp. 1271–1283, Nov. 2002.
- [11] Michael D. Adams, "The jasper project home page," <http://www.ece.uvic.ca/~mdadams/jasper/>.
- [12] "H.264/avc software ver. jm86," <http://iphome.hhi.de/suehring/tml/>.