Local Bandwidth Constrained Fast Inverse Motion Compensation for DCT-Domain Video Transcoding

Shizhong Liu *student member* and Alan C. Bovik *Fellow* IEEE

Laboratory for Image and Video Engineering
Dept. of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712-1084, USA
Email: {sliu2, bovik} @ ece.utexas.edu

This work was supported in part by Texas Instruments, Inc. and by the Texas Advanced Technology Program.

Abstract

Discrete cosine transform (DCT) based digital video coding standards such as MPEG and H.26x are becoming more widely adopted for multimedia applications. Since the standards differ in their format and syntax, video transcoding, where a compressed video bit-stream is converted from one format to another format, is of interest for purposes such as channel bandwidth adaptation and video composition. DCT-domain video transcoding is generally more efficient than spatial domain transcoding. However, since the data is organized block by block in the DCT-domain, inverse motion compensation becomes the bottleneck for DCT-domain methods. In this paper, we propose a novel local bandwidth constrained fast inverse motion compensation algorithm operating in the DCT-domain. Relative to Chang's algorithm, the proposed algorithm achieves computational improvement of 25% to 55% without visual degradation. A by-product of the proposed algorithm is a reduction of blocking artifacts in very low bit-rate compressed video sequences. The proposed algorithm can be combined with other fast methods presented in the literature for more computational savings. We also present a look-up-table (LUT) based implementation method by modeling the statistical distribution of the DCT coefficients in natural images and video sequences. By this method, we obtain a further another 31-48% improvement in computation. The memory requirement of the LUT is about 800KB which is reasonable. Moreover, the LUT can be shared by multiple DCT-domain video processing applications running on the same computer or video server.

Keywords

DCT-domain, inverse motion compensation, MPEG video, video transcoding, video composition.

I. Introduction

Digital video data are becoming widely available as MPEG or H.26x bit-streams. Both MPEG and H.26x are based on the Discrete Cosine Transform (DCT) [1–3]. In a typical DCT compression scheme, the input image is divided into small blocks, and each block is transformed independently to convert the image elements to DCT coefficients. These DCT coefficients are then quantized using a scalar quantizer defined by the so-called quantization matrix. Since the DCT can efficiently concentrate most of the signal energy into relatively few coefficients, the DCT block is quite sparse after quantization. Finally, all quantized DCT coefficients are converted to a bit-stream via variable length encoding. Besides reducing the spatial redundancy of the image, video compression standards also exploit temporal redundancy by coding some video frames as P or B type frames to achieve higher compression ratio [1–3]. Different compression schemes have different goals and target applications, hence different properties. MPEG1/2 are mainly designed for video storage

and high quality video communications, so they make a tradeoff between random access and compression ratio via the Group of Picture (GOP) structure [1]. H.26x standards focus on low bandwidth real time visual communication without considering random access [2,3]. MJPEG is preferred for video editing because of its random access property and moderate compression ability [4]. In practice, certain applications require real-time manipulation of the video stream in order to implement video composition [5–8] or to adapt output channel bandwidth [9–12]. For instance, a video gateway that connects a fast network to a slower network might transcode the video coming from the fast network to a low bandwidth format better suited for the slow link, e.g., from MJPEG to H.261 [13], from MPEG to H.26x [12], or from MPEG to MPEG [11], etc. Meanwhile, in video composition [4–6], we may need to transcode a MPEG or H.26x bit-stream to MJPEG format to facilitate video editing.

There are two general approaches for processing compressed video bit-streams: spatial-domain processing and compressed-domain processing. In spatial-domain methods, the video bit-stream is first fully decompressed, then processed in the decompressed-domain (spatial-domain), and finally re-compressed for storage or transmission. In compressed-domain methods, the video bit-stream is first partially decoded to the DCT-domain, then processed in the DCT-domain, and finally re-encoded. Compressed domain processing delivers several potential advantages vis-à-vis spatial domain processing such as: A) smaller data volume to be processed; B) lower computational complexity since the process of complete decompression and compression can be avoided; C) preservation of image fidelity because of the absence of decompression-compression processes. Inverse motion compensation is a necessary step in most video composition or transcoding applications in order to convert the inter-coded frames to intra-coded frames [4] or compensate the transcoding errors in the anchor frames back to the prediction error frames for drift-free video transcoding [11]. Since the data is organized block by block in the DCT-domain, inverse motion compensation in the DCT-domain is more complex than its counterpart in the spatial-domain and hence becomes the bottleneck of DCT-domain video processing methods [4, 5, 11].

The problem of *DCT-domain inverse motion compensation* was studied by Chang *et*

al. [6]. The general setup is shown in Fig. 1, where $\hat{x}$ is the current block of interest, x_1, x_2, x_3 and x_4 are the reference blocks from which $\hat{x}$ is derived. According to [6], $\hat{x}$ can be expressed as a superposition of the appropriate windowed and shifted versions of x_1, x_2, x_3 and x_4, *i.e.*,

$$\hat{x} = \sum_{i=1}^{4} q_{i1} x_i q_{i2} \tag{1}$$

where q_{ij}, $i = 1, \ldots, 4, j = 1, 2$ are sparse 8×8 matrices of zeros and ones that perform windowing and shifting operations. For example, for $i = 1$,

$$q_{11} = \begin{pmatrix} O & I_h \\ O & O \end{pmatrix}, \quad q_{12} = \begin{pmatrix} O & O \\ I_w & O \end{pmatrix}, \tag{2}$$

where I_h and I_w are identity matrices of dimension $h \times h$ and $w \times w$, respectively. The values h and w are determined by the motion vector corresponding to $\hat{x}$. According to [6], we can obtain its DCT-domain counterpart as

$$\hat{X} = \sum_{i=1}^{4} Q_{i1} X_i Q_{i2} \tag{3}$$

where $\hat{X}$, X_i, Q_{i1} and Q_{i2} are the DCT's of $\hat{x}$, x_i, q_{i1} and q_{i2}, respectively. Be noted that the matrices Q_{i1} and Q_{i2} are constant hence can be pre-computed and stored in memory [6].

Brute-force computation of (3) in the case where the reference block $\hat{x}$ is not aligned in any direction with the block structure requires eight floating-point matrix multiplications and three matrix additions. Several algorithms have been proposed to reduce the computational complexity of the DCT-domain inverse motion compensation. In [7], Merhav *et al.* proposed to factorize the constant matrices Q_{ij} into a series of relatively sparse matrices instead of fully pre-computing them. As a result, some of the matrix multiplications in (3) can be replaced by simple addition and permutation operations such that computational complexity can be reduced. Assunção *et al.* [11] approximated the elements of Q_{ij} to binary numbers with a maximum distortion of $\frac{1}{32}$ so that all multiplications can be implemented by basic integer operations such as *shift* and *add*. They showed that in terms of operations (*shift*, *add*) required, their algorithm has only 28% of the computational complexity of the method proposed by Merhav *et al.* [7] while the distortion introduced

by the approximation is negligible (about 0.2 dB as reported in [11]). In general, motion compensation in MPEG stream is done on macro-block basis, meaning that all blocks in the same macro-block have the same motion vector(s). For example, in 4:2:0 format, one macro-block consists of four luminance blocks and two chrominance blocks. Based on this observation, Song *et al.* [14] presented a fast algorithm for DCT-domain inverse motion compensation by exploiting the shared information among the blocks within the same macro-block instead of constructing the DCT-domain values of each target block separately like the methods in [7,11]. They showed about 44% improvement over the method in [6]. One important aspect of the proposed method is that it can be implemented on top of the algorithms proposed in [7] or [11] for further computational savings. However, the proposed method doesn't apply to the case where one macro-block has multiple motion vectors. For instance, MPEG4 supports four motion vectors per macro-block. While all three methods above adopted the two dimensional procedure shown in Fig. 2, Acharya *et al.* [4] developed another implementation diagram shown in Fig. 3, where the 2-D problem was decomposed into two separate 1-D problems. They showed that the decomposition is more efficient than computing the combined operation. In Fig. 3, the intermediate blocks G_0, G_1 and the target block $\hat{X}$ are given by the following equations:

$$G_0 = X_1 Q_{x0} + X_2 Q_{x1} \tag{4}$$

$$G_1 = X_3 Q_{x0} + X_4 Q_{x1} \tag{5}$$

$$\hat{X} = Q_{y0} G_0 + Q_{y1} G_1 \tag{6}$$

where $Q_{12} = Q_{32} \stackrel{\text{def}}{=} Q_{x0}$, $Q_{22} = Q_{42} \stackrel{\text{def}}{=} Q_{x1}$ and $Q_{11} = Q_{21} \stackrel{\text{def}}{=} Q_{y0}$, $Q_{31} = Q_{41} \stackrel{\text{def}}{=} Q_{y1}$.

In this paper, we propose a novel technique to speed-up the DCT-domain inverse motion compensation. While most algorithms proposed in the literature so far focus on how to reduce the computational complexity of (3) via matrix factorization or approximation, we approach the problem from a different angle by analyzing the statistical properties of natural image/video data. By modeling a natural image as a 2-D separable Markov Random Field [15], we estimate the local bandwidth of the target block to be reconstructed from the reference blocks. The algorithm can reduce the processing time by avoiding the computations of those DCT coefficients outside the estimated local bandwidth. To

compute the DCT coefficients inside the estimated local bandwidth, other fast algorithms proposed in the literature such as [6, 7, 11, 14] can be employed. However, to evaluate the computational improvement achieved only by our algorithm, we implement Chang's method to calculate those DCT coefficients inside the estimated local bandwidth. Note that the separable diagram shown in Fig. 3 is used in our implementation. Experimental results show that the proposed algorithm achieves computational improvement of 25% to 55% without visual degradation, compared to Chang's algorithm in [6]. A by-product of the proposed algorithm is a reduction of blocking artifacts in very low bit-rate compressed video sequences. The proposed algorithm can work on top of other fast methods presented in the literature (*e.g.*, algorithms in [7, 11, 14]) for more computational savings. We also present a look-up-table (LUT) based implementation of our algorithm by modeling the statistical distribution of the DCT coefficients in natural images and video sequences. By this method, we obtain a further another 31-48% improvement in computation. The memory requirement of the LUT is about 800KB which is reasonable. Moreover, the LUT can be shared by multiple DCT-domain video processing applications running on the same computer or video server. One advantage of the LUT method is that the computational cost for motion vectors with half-pixel precision is the same as that for integer-pixel accurate motion vectors. However, in methods proposed in [6, 11, 16], the computational complexity for sub-pixel accurate motion vectors is higher than that for integer-pixel accurate motion vectors.

The rest of this paper is organized as follows. In Section II, a novel algorithm for DCT-domain inverse motion compensation is introduced and its performance is analyzed. In Section III, we present a LUT based implementation of our algorithm. Section IV is experimental results and discussions. Finally, we conclude this paper in Section V.

II. Local Bandwidth Constrained Inverse Motion Compensation

A. The Basic Idea

As discussed in the last section, inverse motion compensation consists of two basic operations, *i.e., windowing and shifting*. The *windowing* operation keeps the data inside the window unchanged but zeros all data outside the window (See Fig. 2 and Fig. 3). As

a result, it usually introduces a steep change at the edge of the window, which means that many artificial high frequency components are possibly introduced by the algorithm. To clarify, let us study the 1-D case. Fig. 4 shows a narrow-band signal $y(n)$ obtained by summing two functions, *i.e.*, $y(n) = y_l(n) + y_r(n)$, $0 \leq n < N$. Let $w_l(n)$, $w_r(n)$ be two window functions, *i.e.*,

$$w_l(n) = \begin{cases} 1, & 0 \leq n \leq M; \\ 0, & \text{otherwise.} \end{cases}$$

$$w_r(n) = \begin{cases} 1, & M < n < N; \\ 0, & \text{otherwise.} \end{cases}$$

We can write $y_l(n) = y(n)w_l(n)$ and $y_r(n) = y(n)w_r(n)$. Let $Y(e^{j\omega})$, $Y_l(e^{j\omega})$, $Y_r(e^{j\omega})$, $W_l(e^{j\omega})$ and $W_r(e^{j\omega})$ be the Discrete Time Fourier Transforms of $y(n)$, $y_l(n)$, $y_r(n)$, $w_l(n)$ and $w_r(n)$, respectively. Then the following equations can be obtained:

$$Y_l(e^{j\omega}) = Y(e^{j\omega}) \otimes W_l(e^{j\omega}) \tag{7}$$

$$Y_r(e^{j\omega}) = Y(e^{j\omega}) \otimes W_r(e^{j\omega}) \tag{8}$$

where $\otimes$ denotes convolution of two periodic functions with the limits of integration extending over only one period. We also have

$$|W_l(e^{j\omega})| = \frac{sin[\omega(M+1)/2]}{sin(\omega/2)}. \tag{9}$$

Let B, B_l, B_r, B_w^l and B_w^r be the bandwidths of $y(n)$, $y_l(n)$, $y_r(n)$, $w_l(n)$ and $w_r(n)$, respectively. From (9), we can roughly estimate $B_w^l \approx \frac{2\pi}{M+1}$, M being the length of the window. B_w^r has similar format as B_w^l. From (7) and (8), we can obtain the following inequalities:

$$B_l > max(B, B_w^l) \tag{10}$$

$$B_r > max(B, B_w^r). \tag{11}$$

Let E_l be the frequency components beyond B in B_l, and E_r be the frequency components beyond B in B_r. Since $y(n) = y_l(n) + y_r(n)$, the following equation must be satisfied

$$E_l + E_r = 0. \tag{12}$$

This means that all frequency components beyond B will disappear after summation, implying that there is no need to compute them. Therefore, if we can estimate the frequency bandwidth B of $y(n)$ before constructing $Y(e^{j\omega})$, we need only compute those frequency components inside B when computing $Y_l(e^{j\omega})$ and $Y_r(e^{j\omega})$. This is the basic idea of the proposed algorithm described in the next subsection.

B. Local Bandwidth Constrained Inverse Motion Compensation

Generally, neighboring pixels are highly correlated in images. This inter-pixel correlation is often modeled by using Markov Random Field (MRF) models [17]. In [15], Sikora *et al.* also assumed that the 2-D image random field is separable with identical and stationary correlation along each image dimension and that the simple first order AR(1) Markov model was adopted to model the pixel-to-pixel correlation along image rows and columns. For each image row, the variance-normalized AR(1) 1-D auto-correlation function can expressed as

$$R_x = \alpha^{|n|}, \tag{13}$$

where n describes the distance between two images pixels and α denotes the pixel-to-pixel correlation in the row. α typically takes values ranging from 0.9 to 0.98 [15, 18]. Fig. 5 shows two 1-D eight point adjacent blocks L_1 and L_2 in an image row. According to the above model, L_1 and L_2 should have the same power spectral density function, hence the same bandwidth because they have the same correlation function [19]. Similarly, if we want to extract L_3 (shown in Fig. 5) from L_1 and L_2, we can predict that L_3 also has the same bandwidth as L_1 and L_2 based on the model. However, images are usually non-stationary, so the bandwidth of L_1 is often different from that of L_2. To account for this, we take the maximum bandwidth as the estimate for L_3, *i.e.*,

$$B_3 \approx max(B_1, B_2), \tag{14}$$

where B_1, B_2 and B_3 are the bandwidth of L_1, L_2 and L_3, respectively. For example, if the maximum index of the non-zero DCT coefficients (here we use DCT coefficients as the representations of frequency components) is 2 in L_1 and 4 in L_2, we estimate that the maximum index of the non-zero DCT coefficients in L_3 is 4. To extract the DCT

coefficients directly from the DCT's of L_1 and L_2, we only need to compute those DCT coefficients with index no greater than 4 in L_3.

The separable implementation scheme (See Fig. 3) can be employed to convert the 2-D problem into two 1-D problems. In (4), (5) and (6), the constant matrices Q_{x0}, Q_{x1}, Q_{y0} and Q_{y1} can be pre-computed and stored in memory. In horizontal translation, we estimate the local bandwidth row by row. For example, in (4), we take the first row of X_1 as L_1, and take the first row of X_2 as L_2. Accordingly, the first row of G_0 corresponds to L_3 in Fig. 5. Then we can utilize the same method as described above to estimate the maximum index of the non-zero DCT coefficients in the first row of G_0, thereafter extract those DCT coefficients with indices no greater than the estimate local bandwidth. All other rows are processed in the same fashion as the first row. After extracting the two intermediate DCT blocks G_0 and G_1 in (4) and (5), we round all elements in the two blocks to the nearest integers in order to eliminate those small numbers without introducing significant errors. Similarly, in (6), the vertical translation can be implemented column by column to extract the final target block $\hat{X}$.

Let us go through a specific example to further clarify the algorithm. Suppose we have two horizontally adjacent DCT blocks X_1 and X_2 shown in (15), we extract G_0 from X_1 and X_2 by (4) using the proposed algorithm. The maximum indices of non-zero coefficients in the first row of X_1 and X_2 are 0 and 1, respectively. According our algorithm, only two coefficients with the indices of 0 and 1 will be computed in the first row of G_0, while the rest of entries in this row are set to zero. In the fifth row, the maximum index of the non-zero coefficients is 2 in X_1 , and there is no non-zero coefficient in X_2. Therefore in the fifth row of G_0, three coefficients with indices from 0 to 2 are computed, and the rest are set to zero. Since all items in the eighth row of both X_1 and X_2 are zero, no computation is needed for this row, *i.e.*, all entries of this row are set to zero in G_0.

$$X_1 = \begin{bmatrix} 1096 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -29 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 11 & 27 & 0 & 0 & 0 & 0 & 0 & 0 \\ 27 & 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & 0 & -15 & 0 & 0 & 0 & 0 & 0 \\ 0 & -15 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -15 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 1120 & -19 & 0 & 0 & 0 & 0 & 0 & 0 \\ -29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{15}$$

C. Accuracy of Local Bandwidth Estimation

As discussed, image/video data is usually a non-stationary random signal. To account for this, the maximum bandwidth of two adjacent blocks is taken as the estimate of the bandwidth of the block to be extracted. However, estimation error still exists under certain circumstances. For example, in Fig. 5, assume L_1 and L_2 are both constant blocks but there is a discontinuity at the boundary between the two blocks. According to the proposed algorithm, the block L_3 should also be a constant block since both L_1 and L_2 only have DC component in the frequency domain. However, the block L_3 actually contains a step discontinuity. The probability of such kind of estimation error will be higher in the images containing lots of edge information than in the relatively smooth images. In addition, since each block in the frame is independently quantized by certain quantization factor, the correlation between adjacent blocks is reduced, which may also make the local bandwidth estimation inaccurate. Several monochrome images with the dimension of 512×512 have been selected to examine the accuracy of the proposed method for local bandwidth estimation in real images. In the experiment, if the estimated bandwidth is smaller than the actual bandwidth of the target block, the estimation is considered incorrect. Otherwise, the estimation is correct. The results are shown in Fig. 6. It can be seen that more than 97% of the estimations are correct for all quantization parameters. The correctness of estimation declines as the quantization increases, implying that more distortion would be introduced in the image/video with large quantization parameter.

III. LUT Based Implementation Method

The proposed algorithm, described in Section II, can reduce the computational complexity of DCT-domain inverse motion compensation by avoiding the computations corresponding to the DCT coefficients outside the estimated local bandwidth. To compute the DCT coefficients inside the estimated local bandwidth, the algorithms proposed in the literature such as [6, 7, 11, 14] can be employed. However, one problem of the existing algorithms is that the complexity for extracting a block with half-pixel accurate motion vector is much higher than that with integer-pixel motion vector. This may cause jerkiness in applications such as real-time video transcoding. Here we present a look-up-table

(LUT) based implementation method by modeling the statistical distribution of the DCT coefficients in typical images and video sequences. One advantage of the LUT method is that the computational cost for motion vectors with half-pixel precision is the same as that for integer-pixel accurate motion vectors. This can help reduce the jerkiness in real-time applications such as video transcoding.

A. Modeling distribution of DCT coefficients

All DCT coefficients in MPEG or H.26x coded images are quantized to integers with value ranging from -2048 to 2047. Since the DCT concentrates most of the signal energy into relatively few coefficients, most AC coefficients have small values. The distribution of AC coefficients can be modeled as a Laplacian distribution with zero mean as follows [20, 21]:

$$p(x) = \frac{\lambda}{2} exp(-\lambda|x|) \tag{16}$$

where

$$\lambda = \frac{1}{E[|X|]}. \tag{17}$$

Let σ^2 be the variance of X, then $\sigma = \frac{\sqrt{2}}{\lambda}$. Given a positive threshold TH, one can have

$$\begin{aligned} P(|X| \leq TH) &= \int_{-TH}^{TH} p(x)dx \\ &= \int_{0}^{TH} \lambda \ exp(-\lambda x)dx \end{aligned} \tag{18}$$

A few JPEG-coded images and I frames from several MPEG-coded video sequences are selected to estimate the value of λ according to (17). As a result, we obtain $\lambda \approx 0.0284$. If we set a threshold $TH = 2\sigma \approx 100$, then more than 94% of AC coefficients have absolute value smaller than the threshold TH according to (18). This implies that a lot of computation can be saved by pre-computing the multiplication results for all those coefficients having absolute value smaller than the threshold TH. In the following, the implementation of the LUT based method will be discussed.

B. LUT design

By using the separable approach [4], only 1-D case needs to be considered to build the LUT. So two tables are needed to save all multiplication results with Q_{x0} and Q_{x1}

in (4) and (5). In Fig. 1, w has 16 possible values including half-pixel resolution, *i.e.*, $w = 0, 0.5, 1, \ldots, 7.5$. For 1-D case, each non-zero element in $\{X_i\}$ contributes to eight entries of G_0 or G_1. As a result, a four dimensional table is needed to save the pre-computed results, *i.e.* Table$[v][p][w][i]$, where v represents the absolute value of DCT coefficients, p represents the column position of DCT coefficients in $\{X_i\}$, w is shown in Fig. 1 and i represents the column position of the pre-computed results in the target block. Both p and i have eight possible values each. v has 100 possible values since TH is 100. If four bytes are used to store each entry of the table, the size of table is

$$size = 4 \times v \times p \times w \times i = 4 \times 100 \times 8 \times 16 \times 8 = 400KB. \tag{19}$$

Hence, the total memory requirement for two tables is about $800KB$ when $TH = 100$. This is reasonable based on the current computer memory capacity. Furthermore, the LUT can be shared by multiple DCT-domain video processing applications running on the same computer or video server. The vertical operations in (6) can be converted to the horizontal operations via matrix transposition so that the LUT can be reused.

C. DC coefficients

In DCT-coded images, the distribution of DC coefficients has a large mean value (*e.g.*, 1000) and a larger variance relative to the distribution of AC coefficients as shown in Fig. 7(a). Since the LUT is created by modeling the distribution of AC coefficients, it doesn't apply to DC coefficients because most DC coefficients are much larger than the threshold TH. For adjacent DCT blocks, the DC coefficients are highly correlated in typical images [15]. Therefore, the difference between adjacent DC coefficients should have much smaller mean value and dynamical range than the DC coefficient itself. With this observation, (4) can be rewritten as

$$\begin{aligned} G_0 &= X_1 Q_{x0} + X_2 Q_{x1} \\ &= \frac{X_1 + X_2}{2}(Q_{x0} + Q_{x1}) + \\ &\quad \frac{X_1 - X_2}{2}(Q_{x0} - Q_{x1}). \end{aligned} \tag{20}$$

Let $Q_+ = Q_{x0} + Q_{x1}$, Q_+ has the property:

$$Q_+(0,0) = 1; Q_+(0,j) = Q_+(j,0) = 0; j = 1, \ldots, 7.$$

This means that the summation of the DC components of X_1 and X_2 only contributes to the DC component of G_0. So we can just sum up both DC coefficients from X_1 and X_2, then fill it in the DC entry of G_0 without any further computation. The difference between the DC component of X_1 and that of X_2 has distribution similar to the distribution of AC coefficients as shown in Fig. 7(b). In the selected JPEG-coded images, more than 70% of the difference values have absolute value below the threshold TH so that their multiplication results in (20) can be directly obtained from the LUT.

IV. Experimental Results and Discussions

To evaluate the performance of the proposed algorithm and LUT based implementation method, We implement three methods for DCT-domain inverse motion compensation, which are

- Method I: Chang's method in [6] with full computation of all DCT coefficients in each target block to be extracted.
- Method II: The proposed local bandwidth constrained algorithm which skips the DCT coefficients outside the estimated bandwidth and calculates those DCT coefficients inside the estimated bandwidth using Chang's method.
- Method III: Method II with LUT based implementation.

All three methods are integrated into our DCT-domain video transcoder, for comparison, as the inverse motion compensation module, respectively. The input of the transcoder is a MPEG-coded video bit-stream with the frame rate of 30 frames per second. The GOP structure of the encoded video is $M = 3, N = 12$, *i.e.*, IBBPBBPBBPBB. To evaluate the performance of the proposed algorithm, we transcode all P and B frames in the incoming bit-stream back to I frames by DCT-domain inverse motion compensation. Since the proposed algorithm only computes those DCT coefficients inside the estimated bandwidth, we first investigate the distortion caused by the algorithm by comparing the PSNR values of those I frames recovered from P or B frames using both Method I and Method II, respectively. Then we measure the computing time of all three methods to show the computational improvement of the proposed algorithms. Be noted that LUT based implementation does not introduced further distortion. Hence all frames reconstructed by Method III have the same PSNR values as those by Method II.

A. Distortion Introduced by the Proposed Method

The complexity of converting one P or B frame to an I frame really depends on the characteristics of the test video sequence. If there is little motion in the video sequence such as "header and shoulder" sequences, most of the macro-blocks are skipped in the P or B frame with zero motion vectors. To recover those skipped macro-blocks with zero motion vectors, we simply copy the DCT coefficients from the reference blocks without any computation, because in this case, the motion vectors are aligned to the block boundaries. To test our algorithm more efficiently, we select four video sequences with intensive motion activities, i.e., *Foreman*, *Coastguard*, *Mobile* and *Stefan*. All sequences are CIF resolution with 352 pixels per line and 288 lines. To evaluate the performance of our algorithm at different coding bit-rate, the sequences are encoded at 4 Mb/s and 1 Mb/s, respectively. The PSNR results for each frame after inverse motion compensation are shown in Fig. 8 and Fig. 9, respectively. For the bit-rate of 4 Mb/s, the average PSNR degradation is 0.11 dB in *Foreman*, 0.12 dB in *Coastguard*, 0.22 dB in *Mobile* and 0.16 dB in *Stefan*. For the bit-rate of 1 Mb/s, the average PSNR degradation is 0.29 dB in *Foreman*, 0.35 dB in *Coastguard*, 0.51 dB in *Mobile* and 0.36 dB in *Stefan*. The PSNR degradation depends on the images being tested. For example, in the sequence *mobile*, the pictures have a lot of strong edges and are very dynamic; hence the AR model of our algorithm is inaccurate, which increases the error probability of local bandwidth estimation as discussed in Section II(c). As a result, the PSNR of *mobile* degrades more than that of other sequences. Similarly, at low bit-rates, since each block in the frame is independently quantized by a large quantization factor, the correlation between adjacent blocks is reduced. Thus, the error probability of local bandwidth estimation increases as shown in Fig. 6, which causes more degradation at low bit-rates as shown in the experimental results. However, for both encoding bit-rates, the amount of distortion introduced by the proposed algorithm is hardly visible. A by-product of our algorithm is to reduce the blocking artifacts in images coded at very low bit-rates because the local bandwidth constrained inverse motion compensation also works as a low-pass filter to the reconstructed block. To show this, we encode the gray-level image *Lena* at 0.25 bits/pixel (Fig. 10(a)) using JPEG. Then we shift the image by 4 pixels in the vertical direction and reconstruct the shifted image

using inverse motion compensation. The images reconstructed by Method I and Method II are shown in Fig. 10(b) and Fig. 10(c), respectively. Fig. 10(d) shows the difference between the two reconstructed images. Clearly, image reconstructed by Method II is smoother than that by Method I. Additionally, we applied the blocking artifact measuring method proposed in [22] to both reconstructed images. The measuring results are 6.1 for the image reconstructed by Method I, and 3.5 for the image reconstructed by Method II, respectively. The results also show that the blocking artifacts in the image reconstructed by Method II are much less severe than in the image reconstructed by Method I. Since our algorithm is equivalent to adaptive low-pass filtering according to the local bandwidth, in Fig. 10(d), no large errors exist in areas with high spatial frequencies.

B. Computational Savings

In the experiments, all three methods for DCT-domain inverse motion compensation are integrated into our DCT-domain video transcoder, for comparison, as the inverse motion compensation module, respectively. Only the computing time to convert one P or B frame to an I frame is measured. The computing time is measured on a Windows NT workstation with 512MB memory and 300MHz Pentium II Processor (32K non-blocking, level-one cache, and 512K unified, non-blocking, level-two cache.). Table I and Table II list the average time to reconstruct one P or B frame to one I frame by three methods at bit-rates of 4 Mb/s and 1 Mb/s, respectively. Relative to Method I, Method II achieves 25 - 30% computational savings at the bit-rate of 4 Mb/s, and 45 - 55% at the bit-rate of 1 Mb/s, respectively. Generally, the computational savings achieved by the proposed algorithm is inversely proportional to the encoding bit-rate since the DCT block becomes sparser as the bit-rate goes down. On the average, Method III, employing the LUT based implementation, obtains another improvement of 48% at the bit-rate of 4 Mb/s and 31% at the bit-rate of 1 Mb/s, compared to Method II. Fig. 11 shows the computing time corresponding to each P or B frame in the sequence *Mobile* for all three methods. As can be seen, the computing time in Method III is almost constant for different P or B frames while the computing time in Method I and Method II changes dramatically. One reason is the complexity involved in the processing of motion vectors with half-pixel precision. When a half-pixel accurate motion vector is used, either two or four pixels are needed to

compute the actual prediction of one pixel, depending on the motion vector has half-pixel precision in one or two directions. In the DCT-domain, this means that either two or four blocks need to be extracted from the reference frame(s) to obtain the final target block by averaging the extracted blocks. As a result, in Method I and II, the complexity for half-pixel accurate motion vector is two or four times that for motion vector with integer-pixel resolution. Although some fast algorithms for fractal motion vectors have been proposed in [11, 16], the complexity is still high compared to that for integer motion vectors. However, in the LUT implementation, the complexity corresponding to fractal motion vectors is the same as that for integer motion vectors because the results for half-pixel resolution are pre-computed and saved in the LUT.

TABLE I

AVERAGE TIME TO CONVERT ONE P OR B FRAME TO ONE I FRAME AT THE BIT-RATE OF 4 MB/S

(UNIT: SECONDS)

Video Sequence	P frame			B frame		
	Method I	Method II	Method III	Method I	Method II	Method III
Foreman	0.3137	0.2387	0.0931	0.4738	0.3644	0.1423
Coastguard	0.2374	0.1700	0.0912	0.3417	0.2464	0.1190
Mobile	0.3487	0.2513	0.1642	0.4136	0.3113	0.2007
Stefan	0.2057	0.1370	0.0780	0.3667	0.2484	0.1416

TABLE II

AVERAGE TIME TO CONVERT ONE P OR B FRAME TO ONE I FRAME AT THE BIT-RATE OF 1 MB/S

(UNIT: SECONDS)

Video Sequence	P frame			B frame		
	Method I	Method II	Method III	Method I	Method II	Method III
Foreman	0.2512	0.1324	0.0651	0.3987	0.2152	0.1036
Coastguard	0.1912	0.0937	0.0656	0.3099	0.1490	0.0964
Mobile	0.2983	0.1550	0.1150	0.3686	0.2061	0.1702
Stefan	0.1636	0.0743	0.0605	0.2941	0.1408	0.1147

C. Discussions

In our experiments, Chang's method [6] is employed to compute the DCT coefficients within the estimated local bandwidth. Experimental results have shown that the proposed algorithm achieves computational improvement of 25% to 55% without visual degradation. Meanwhile, the proposed algorithm can work on top of other existing fast algorithms (*e.g.*, algorithms in [7,11,14]) for further computational savings. For example, to extract a block from the reference blocks, the proposed method can be first employed to estimated the local bandwidth of the target block; then the method in [7] or [11] may be used to calculate the DCT coefficients within the estimated local bandwidth; finally, the shared information among the blocks inside the same macro-block can be exploited by the method in [14].

In the experiments, all video sequences are encoded with frame based motion compensation and frame DCT coding mode. However, MPEG-2 standard supports both progressive and interlaced video format. In interlaced video, each frame consists of two fields, *i.e.*, top field and bottom field. The two fields of a frame may be coded separately (field pictures) or coded together as a frame (frame pictures). Both frame pictures and field pictures may be used in a single video sequence. In a frame picture, both frame DCT coding and field DCT coding can be used on macro-block basis. Therefore, if the reference blocks have different coding format from the target block (*e.g.*, reference blocks are field DCT coded and the target block is frame DCT coded.), the reference blocks must be converted into the same coding format as that of the target block before the proposed algorithm for DCT-domain inverse motion compensation is applied. Similarly, for those different motion compensation (MC) modes supported by MPEG-2, such as frame prediction, field prediction, 16×8 and dual prime modes, corresponding conversion may also be needed before inverse motion compensation. While in this paper, we focus on developing the novel algorithms for DCT-domain inverse motion compensation by assuming the current frame has the same coding format with the reference frame(s), the detailed descriptions of the DCT-domain algorithms for the conversion between different coding modes in MPEG-2 can be found in [11, 23].

V. Conclusion

Inverse motion compensation is the bottleneck for DCT-domain video transcoding. In this paper, we propose a novel local bandwidth constrained inverse motion compensation algorithm, in which only those DCT coefficients inside the estimated bandwidth are computed. The local bandwidth estimation is based on the assumption that the image can be modeled as two separable AR sequences in the horizontal and vertical directions, respectively. Relative to Chang's algorithm, the proposed algorithm achieves computational improvement of 25% to 55% without visual degradation. A by-product of the proposed algorithm is a reduction of blocking artifacts in very low bit-rate compressed video sequences. The proposed algorithm can be combined with other fast methods presented in the literature for more computational savings. We also present a LUT based implementation method by modeling the statistical distribution of the DCT coefficients in natural images and video sequences. By this method, we obtain a further another 31-48% improvement in computation. One advantage of the LUT method is that the computational cost for motion vectors with half-pixel precision is the same as that for integer-pixel accurate motion vectors. This can help reduce the jerkiness in real-time video transcoding. The memory requirement of the LUT is about 800KB which is reasonable. Moreover, the LUT can be shared by multiple DCT-domain video processing applications running on the same computer or video server. Some other applications, such as DCT-domain scene-cut detection and DCT-domain feature extraction for video indexing, can also benefit from the fast inverse motion compensation algorithm proposed in this paper.

References

[1] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard.* New York, NY: Chapman & Hall, 1997.

[2] "Video codecs for audiovisual services at $p \times 64kb/s$." ITU-T Rec. H.261, Mar. 1993.

[3] "Video coding for low bitrate communication." ITU-T Rec. H.263, 1998.

[4] S. Acharya and B. Smith, "Compressed domain transcoding of MPEG." Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS), Austin, TX, June 1998.

[5] S. F. Chang and D. G. Messerschmitt, "A new approach to decoding and compositing motion-compensated DCT based images," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Minneapolis, MN), pp. 421–424, Apr. 1993.

[6] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. on Selected Areas in Comm.*, vol. 13, pp. 1–11, Jan. 1995.

[7] N. Merhav and V. Bhaskaran, "Fast algorithm for DCT-domain image down-sampling and for inverse motion compensation," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 7, pp. 468–476, June 1997.

[8] B. Shen, K. Sethi, and V. Bhaskaran, "DCT convolution and its application in compressed domain," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 8, pp. 947–952, Dec. 1998.

[9] H. Sun, W. Kwok, and J. W. Zdepski, "Architectures for MPEG compressed bitstream scaling," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 6, pp. 191–199, Apr. 1996.

[10] K.-S. Kan and K.-C. Fan, "Video transcoding architecture with minimum buffer requirement for compressed MPEG-2 bitstream," *Signal Processing*, vol. 67, pp. 223–235, 1998.

[11] P. A. A. Assuncao and M. Ghanbari, "A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 8, pp. 953–967, Dec. 1998.

[12] T. Shanableh and M. Ghanbari, "Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats," *IEEE Trans. on Multimedia*, vol. 2, pp. 101–110, June 2000.

[13] E. Amir, S. McCanne, and H. Zhang, "An application-level video gateway," in *Proc. of the Third ACM International Conference on Multimedia*, (San Francisco, CA), Nov. 1995.

[14] J. Song and B.-L. Yeo, "A fast algorithm for DCT-domain inverse motion compensation based on shared information in a macroblock," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 10, pp. 767–775, Aug. 2000.

[15] T. Sikora and H. Li, "Optimal block-overlapping synthesis transforms for coding images and video at very low bitrates," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 6, pp. 157–167, Apr. 1996.

[16] N. Merhav and V. Bhaskaran, "Fast inverse motion compensation algorithms for mpeg and for partial dct information," *J. Visual Communication and Image Representation*, vol. 7, pp. 395–410, Dec. 1996.

[17] G. R. Cross and A. K. Jain, "Markov random field texture models," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 25–39, Jan. 1983.

[18] I. M. Pao and M. T. Sun, "Modeling DCT coefficients for fast video encoding," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 9, pp. 608–616, June 1999.

[19] M. H. Hayes, *Statistical Digital Signal Processing and Modeling.* New York, NY: John Wiley & Sons, Inc., 1996.

[20] K. A. Birney and T. R. Fischer, "On the modeling of DCT and subband image data for compression," *IEEE Trans. on Image Processing*, vol. 4, pp. 186–193, Feb. 1995.

[21] E. Y. Lam and J. W. Goodman, "A mathematical analysis of the DCT coefficient distributions for images," *IEEE Trans. on Image Processing*, vol. 9, pp. 1661–1666, Oct. 2000.

[22] Z. Wang, A. C. Bovik, and B. L. Evans, "Blind measurement of blocking artifacts in images," in *Proc. IEEE Int. Conf. Image Proc.*, (Vancouver, Canada), pp. 981–984, Oct. 2000.

[23] J. Song and B.-L. Yeo, "Fast extraction of spatially reduced image sequences from MPEG-2 compressed video," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 9, pp. 1100–1114, Oct. 1999.

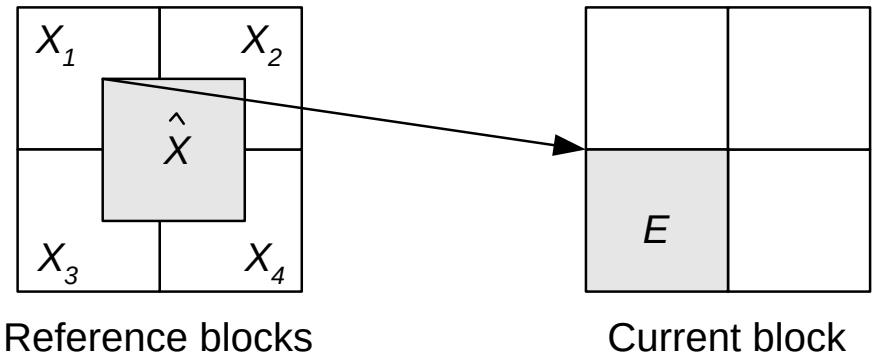


Fig. 1. DCT domain inverse motion compensation.

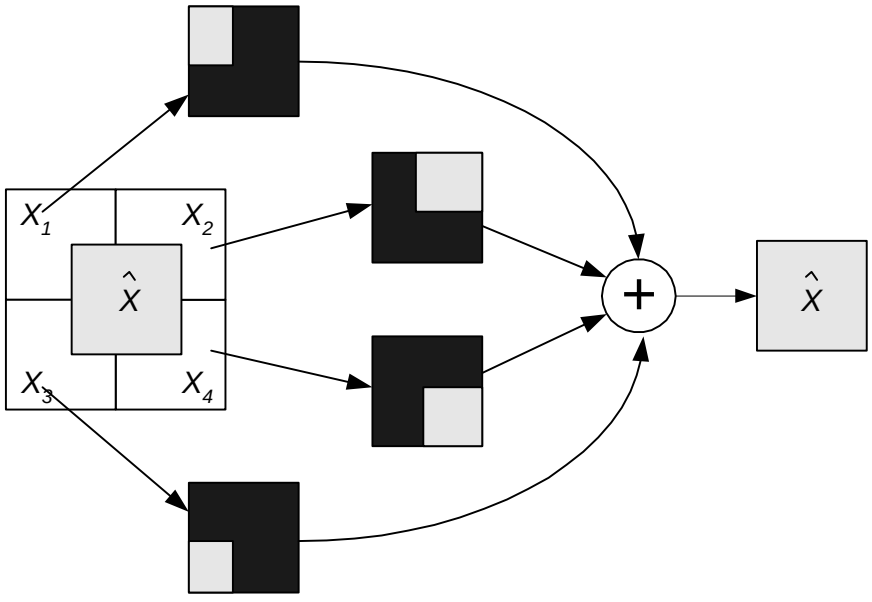


Fig. 2. Illustration of 2-D block windowing and shifting operations (Black area means zero).

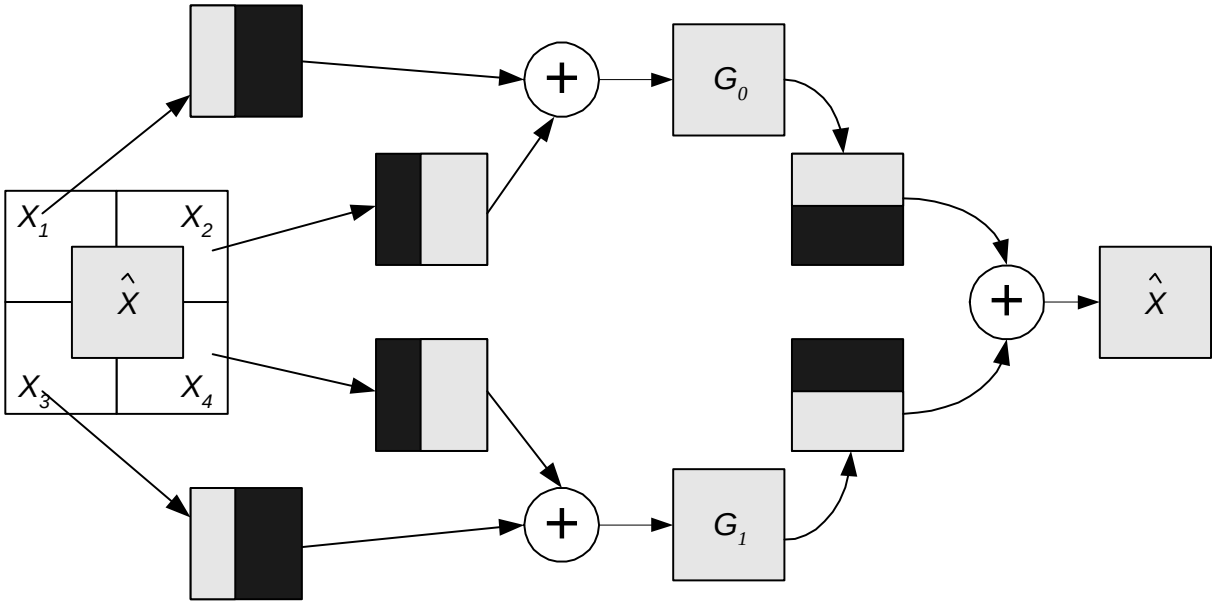


Fig. 3. Illustration of separable block windowing and shifting operations (Black area means zero).

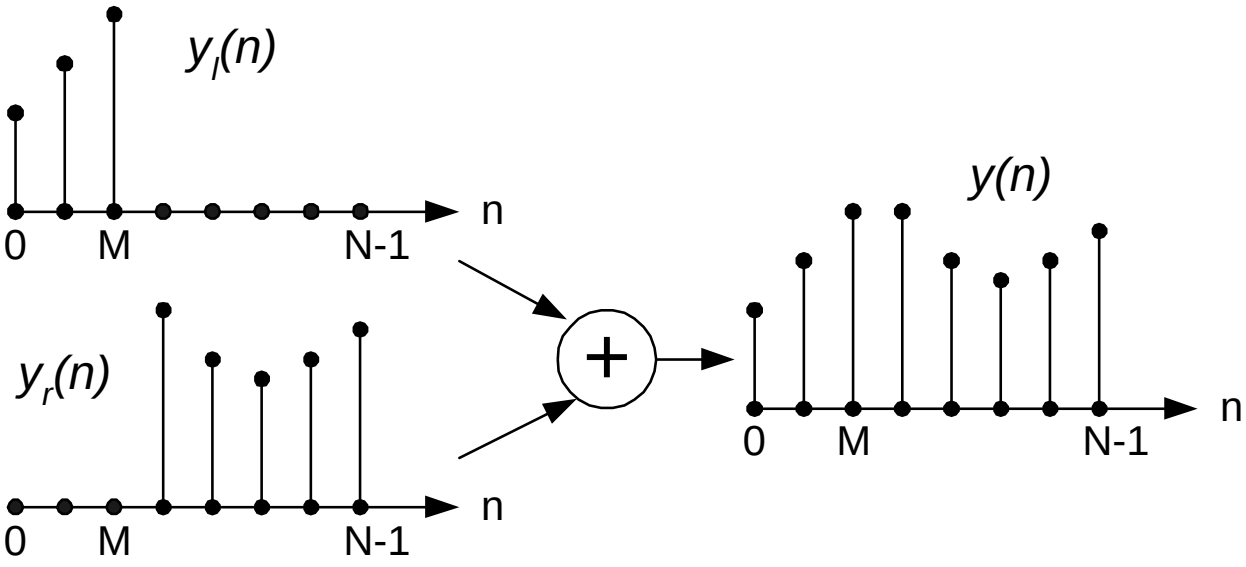


Fig. 4. 1-D windowing operation.

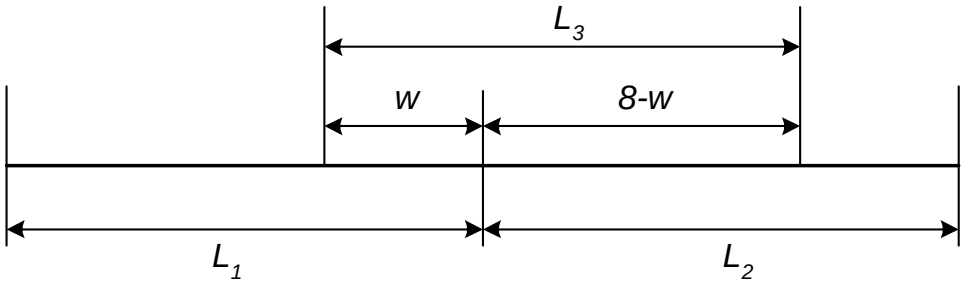


Fig. 5. 1-D block extraction.

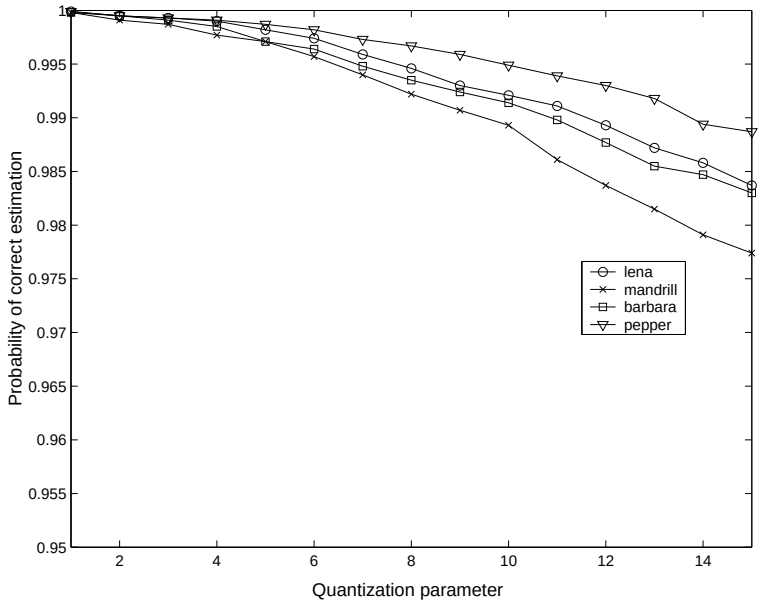
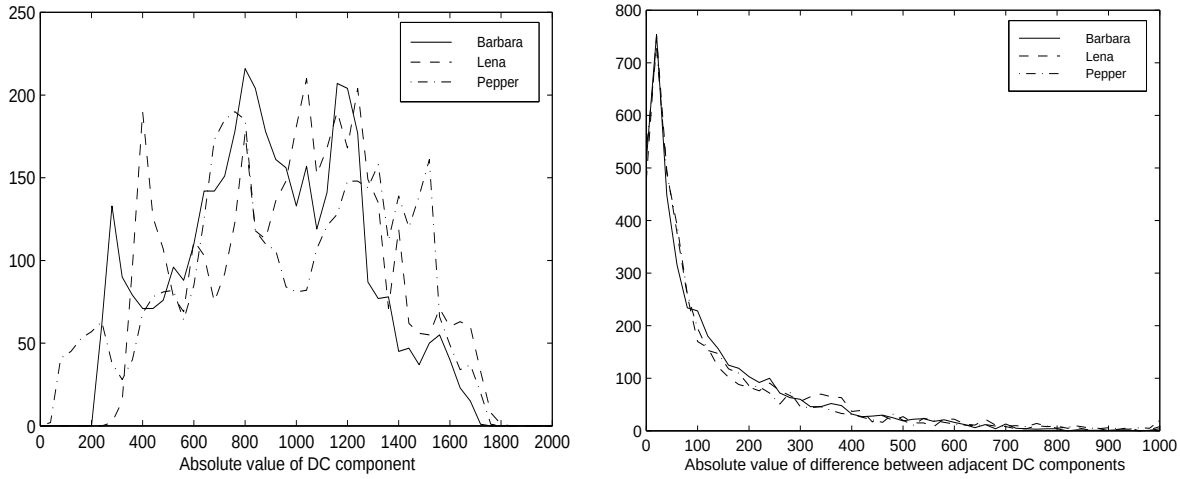


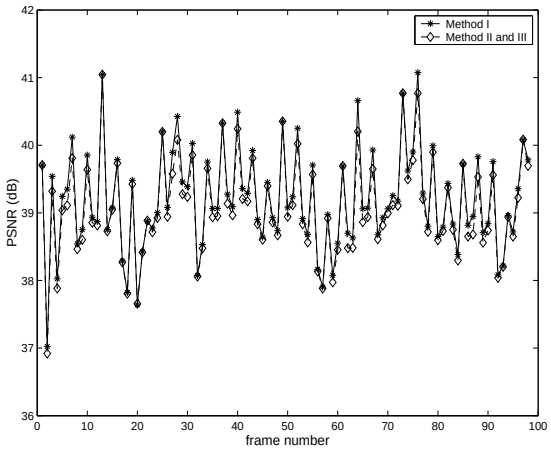
Fig. 6. Accuracy of local bandwidth estimation.



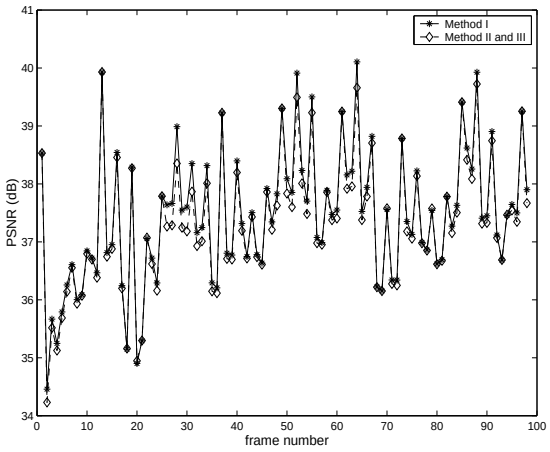
(a) Histograms of DC coefficients in images with the Bin size 40.

(b) Histograms of difference between adjacent DC coefficients in images with the Bin size 20.

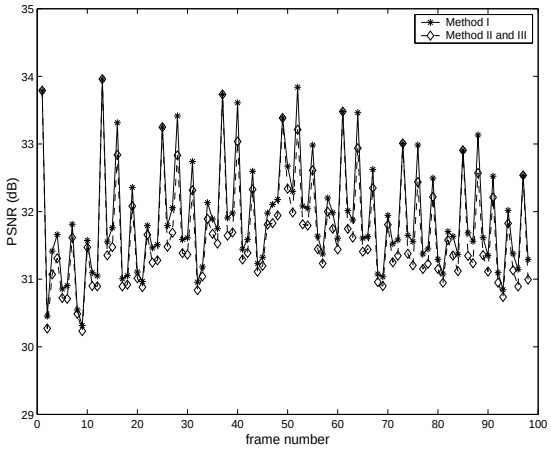
Fig. 7. Distribution of DC component and the difference of adjacent DC components in natural images.



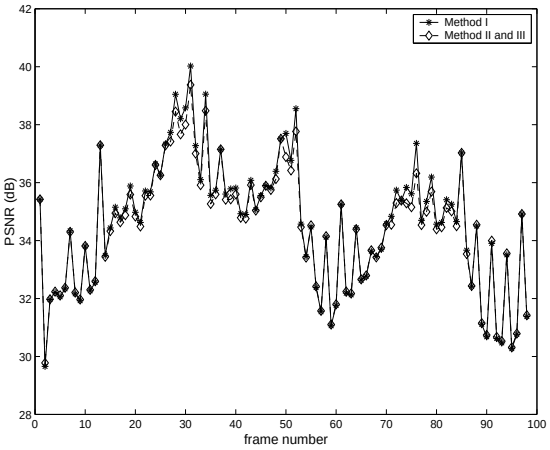
(a) PSNR of *Foreman* at 4 Mb/s.



(b) PSNR of *Coastguard* at 4 Mb/s.

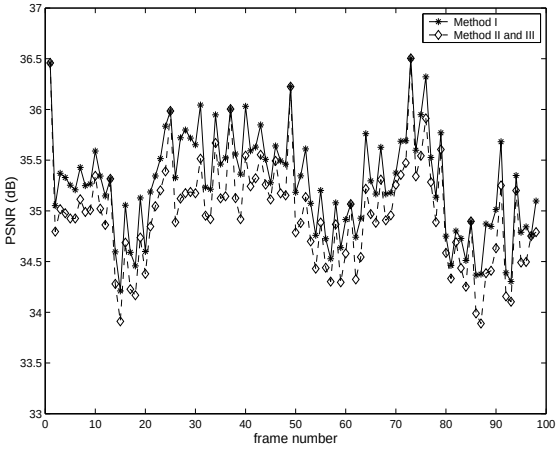


(c) PSNR of *Mobile* at 4 Mb/s.

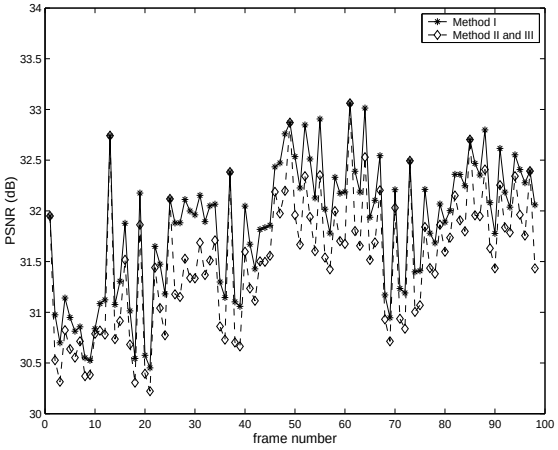


(d) PSNR of *Stefan* at 4 Mb/s.

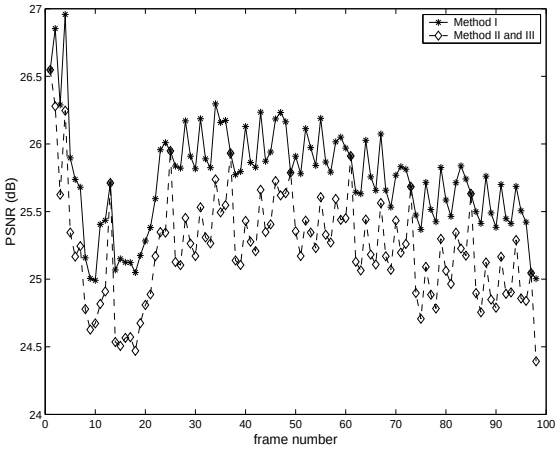
Fig. 8. PSNR value of each frame reconstructed by different inverse motion compensation algorithms.



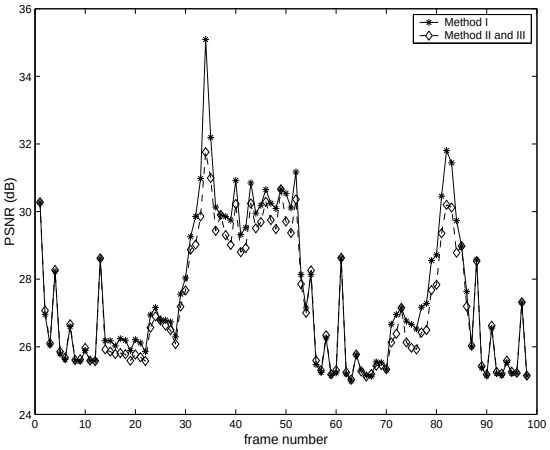
(a) PSNR of *Foreman* at 1 Mb/s.



(b) PSNR of *Coastguard* at 1 Mb/s.



(c) PSNR of *Mobile* at 1 Mb/s.



(d) PSNR of *Stefan* at 1 Mb/s.

Fig. 9. PSNR value of each frame reconstructed by different inverse motion compensation algorithms.

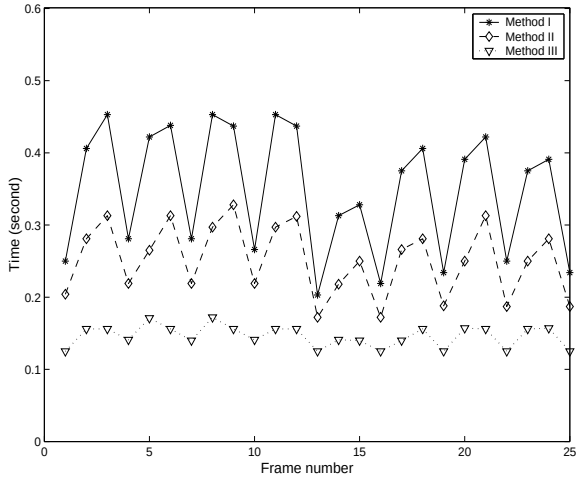
(a) Image *Lena* JPEG-coded at 0.25 bits/pixel.

(b) Image reconstructed by Method I.

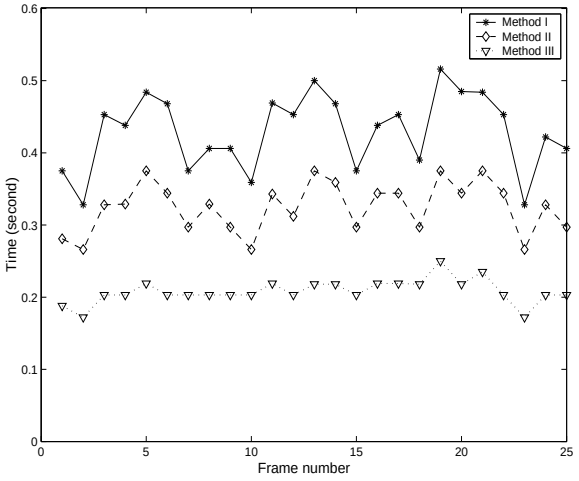
(c) Image reconstructed by method II and III.

(d) Difference between (b) and (c).

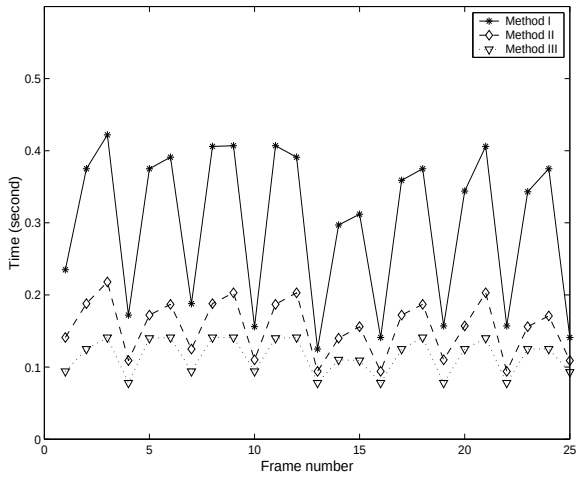
Fig. 10. Illustration of reduction of blocking artifacts by the proposed algorithm.



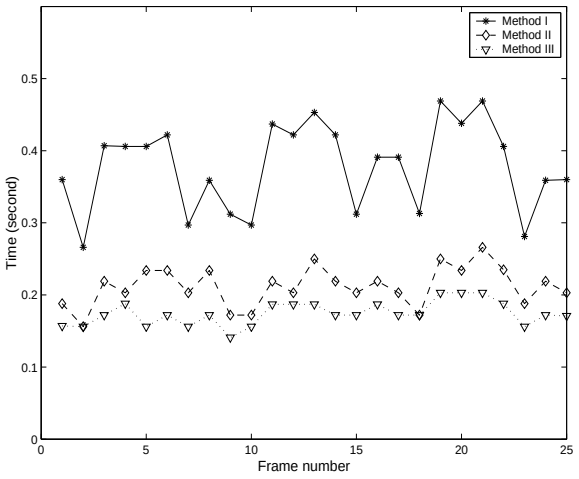
(a) Time for reconstructing each P frame at 4 Mb/s.



(b) Time for reconstructing each B frame at 4 Mb/s.



(c) Time for reconstructing each P frame at 1 Mb/s.



(d) Time for reconstructing each B frame at 1 Mb/s.

Fig. 11. The computing time for reconstructing each P or B frame in the video sequence *Mobile*.