Fast Inverse Motion Compensation Algorithms for MPEG-2 and for Partial DCT Information

Neri Merhav
Computer Systems Laboratory and HP-ISC*

and

Vasudev Bhaskaran
Computer Systems Laboratory†

Keywords: video compression, MPEG, video browsing, compressed domain processing, motion vector, MPEG-2.

Abstract

In prior work, we developed a fast inverse motion compensation method that can be implemented directly on the DCT domain representation derived from the compressed bitstreams conforming to MPEG, H.261 and H.263 standards. That work was restricted to compressed-domain representations wherein the motion-vectors have integer pel accuracy. Here, we extend this work to sub-pel accurate motion-vectors. Here we also extend the prior work to speedup the inverse motion compensation process in the DCT domain by explicitly exploiting the sparseness of the DCT domain representation. Using partial DCT information we show that the DCT domain method has substantially lower operation count than the conventional spatial domain approach which requires decompression followed by inverse motion-compensation.

*On sabbatical leave at HP Laboratories. Current address: Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A. Email: merhav@hpl.hp.com. Permanent address: HP Israel Science Center, Technion City, Haifa 32000, Israel. E-mail: merhav@hp.technion.ac.il

†Address: Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A. E-mail: bhaskara@hpl.hp.com

1 Introduction

While standard image and video compression schemes like JPEG, MPEG, H.261 and H.263 reduce dramatically the required amount of storage and transmission bandwidth without sacrificing much quality, the compressed domain does not always lend itself to easy image processing and manipulation. Indeed, the last few years have witnessed a rapidly growing interest in developing sophisticated fast algorithms for manipulating compressed images and video streams directly in the compressed domain, namely, the discrete cosine transform (DCT) domain, without explicit transformation back to the uncompressed domain, which is computationally expensive.

In this work, we focus on compressed-domain based algorithms for two different problems associated with fast reconstruction of a video sequence from an MPEG-compressed sequence: inverse motion compensation for fractional motion vectors, and inverse motion compensation using partial DCT information.

The first problem that is treated in this paper is that of extending the fast inverse motion compensation algorithm developed in [4] so as to deal with motion vectors whose components are not necessarily integers. In MPEG, the components of each motion vector are integer multiples of 0.5, e.g., 5.5, 7, 14.5, etc., where the interpretation is that of averaging the neighboring pixel intensities. If only one component of the motion vector is noninteger, then there are two neighboring pixels to be averaged, and if both are noninteger, then the four nearest neighbors with integer coordinates are averaged.

We show that the algorithm proposed in [4] extends easily to this setting of fractional motion vectors and the additional computational complexity is relatively small.

In the second problem, by "partial DCT information" we mean, in particular, only very few DCT coefficients of each block. The underlying motivation is in accelerating the process of decompression for quick browsing applications, where the viewer is interested merely in rough information of the present scene, and the current location within the movie or the video clip. The method proposed here can also be used as the front-end of a video database query system. In [2] (see also [1], [3]) Yeo and Liu have developed exact and approximate algorithms for fast reconstruction of subsampled images for two possible levels of partial DCT information: The first level uses the DC component only, and the second uses the first three DCT coefficients, i.e., the DC, the $(0, 1)$, and the $(1, 0)$ DCT coefficients. This set of DCT coefficients will be henceforth referred to as $DC + 2AC$.

Sometimes DC-only images are sufficient for quick browsing applications, and this is a great advantage since the DC information is given almost free. However, DC-only images tend to be very blocky and are not useful as-are for browsing or for video scene analysis. In [2], it is shown that such images have to be subsampled in order to facilitate scene analysis. When the scene includes important text or other fine but critical information, the DC component alone might not suffice. In these cases, more DCT coefficients have to be used. For a small group of pictures (GOP) setting in MPEG-1 or MPEG-2 (e.g., GOP ≤ 4), $DC + 2AC$ turns out to be typically a good compromise between speed of decompression on one hand, and acceptable quality of the browsed scene on the other hand. For a typical GOP setting (e.g. GOP = 12 - 16), based on simulations reported here, the $DC + 2AC$ choice may not be acceptable for most video browsing applications; we need around 6 DCT coefficients in order to get acceptable quality. The 6 DCT coefficients that were found to provide good quality are: DC, $(0, 1)$, $(0, 2)$, $(1, 0)$, $(1, 1)$ and $(2, 0)$ - these are the first six coefficients obtained from zigzag ordering within the 8×8 DCT matrix; henceforth we will refer to this as the $3-2-1$ case.

We develop and propose a fast algorithm for inverse motion compensation in the DCT domain for the $3-2-1$ case. Unlike the algorithm in [2], our algorithm produces images in the original size, and not a down-sampled version. This guarantees that first, the output stream complies with the original semantics in terms of display resolution, and second, it also allows the viewer to browse on a bigger and more detailed image sequence. Our development will focus on the bottleneck of the decoding process, which is the inverse motion compensation part, whose derivation is a further development on [4]. We also demonstrate that the operations count, can be further reduced by using a multiplication-free approximate version of our algorithm, without significant additional degradation in quality. In this paper, we include simulation results for the $3-2-1$ case and other cases of interest.

2 Preliminaries and Problem Description

The 8×8 2D-DCT transforms a block $\{x(n,m)\}_{n,m=0}^{7}$ in the spatial domain into a matrix of frequency components $\{X(k,l)\}_{k,l=0}^{7}$ according to the following equation

$$X(k,l) = \frac{c(k)}{2}\frac{c(l)}{2}\sum_{n=0}^{7}\sum_{m=0}^{7} x(n,m)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi), \tag{1}$$

where $c(0) = 1/\sqrt{2}$ and $c(k) = 1$ for $k > 0$. The inverse transform is given by

$$x(n,m) = \sum_{k=0}^{7}\sum_{l=0}^{7} \frac{c(k)}{2}\frac{c(l)}{2} X(k,l)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi). \tag{2}$$

In a matrix form, let $\boldsymbol{x} = \{x(n,m)\}_{n,m=0}^{7}$ and $\boldsymbol{X} = \{X(k,l)\}_{k,l=0}^{7}$. Define the 8-point DCT matrix $S = \{s(k,n)\}_{k,n=0}^{7}$, where

$$s(k,n) = \frac{c(k)}{2}\cos(\frac{2n+1}{16}\cdot k\pi). \tag{3}$$

Then,

$$\boldsymbol{X} = S\boldsymbol{x}S^t \tag{4}$$

where the superscript t denotes matrix transposition. Similarly, let the superscript $-t$ denote transposition of the inverse. Then,

$$\boldsymbol{x} = S^{-1}\boldsymbol{X}S^{-t} = S^t\boldsymbol{X}S \tag{5}$$

where the second equality follows from the unitarity of S.

Motion compensation of compressed video [5], [6] (see also [7]) means predicting each 8×8 spatial domain block $\boldsymbol{x}$ of the current frame by a corresponding reference block $\hat{\boldsymbol{x}}$ from a previous frame [1] and encoding the resulting prediction error block $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ by using the

[1] In some of the frames (B-frames) blocks are estimated from both past and future reference blocks. For the sake of simplicity, we shall assume here that only the past is used (P-frames). The extension to B-frames is straightforward.

DCT. The best matching reference block $\hat{\boldsymbol{x}}$ may not be aligned to the original 8×8 blocks of the reference frame. In general, the reference block may intersect with four neighboring spatial domain blocks, henceforth denoted $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, and $\boldsymbol{x}_4$, that together form a 16×16 square, where $\boldsymbol{x}_1$ corresponds to northwest, $\boldsymbol{x}_2$ to northeast, $\boldsymbol{x}_3$ to southwest and $\boldsymbol{x}_4$ to southeast. This is illustrated in Fig. 1.

In the first problem our goal is to compute the DCT $\boldsymbol{X}$ of the current block $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{e}$ given its motion-vector relative to a region in a previous frame, the current block's interframe DCT $\boldsymbol{E} = S\boldsymbol{e}S^t$ and the intraframe DCT's $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ of the previous frame which is referenced by the motion-vector. Here, the motion-vector is given at subpixel accuracy. We develop the algorithm in Section 4.

The goal of the second problem is nearly identical to that of the first problem; however, the key difference is that here we need to compute $\boldsymbol{X}$ using partial DCT information for $\boldsymbol{E}$ and $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$. In Fig. 2, we show several cases of interest for the partial DCT information.

We provide a detailed algorithm for computing $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{e}$ using only 6 DCT coefficients (we refer to this choice of DCT coefficients as the $3-2-1$ case) for $\boldsymbol{E}$ and $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$. Specifically our goal for the $3-2-1$ case is to compute the DCT $\boldsymbol{X}$ of the current block $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{e}$ from the $3-2-1$ part of the DCT $\boldsymbol{E} = S\boldsymbol{e}S^t$, that is, $\boldsymbol{E}(0,0)$, $\boldsymbol{E}(0,1)$, $\boldsymbol{E}(0,2)$, $\boldsymbol{E}(1,0)$, $\boldsymbol{E}(1,1)$, $\boldsymbol{E}(2,0)$ and the $3-2-1$ parts of the DCT's $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ of $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$, respectively. Since $\boldsymbol{X} = \hat{\boldsymbol{X}} + \boldsymbol{E}$, $\hat{\boldsymbol{X}}$ being the DCT of $\hat{\boldsymbol{x}}$, the main problem that remains is that of calculating $\hat{\boldsymbol{X}}$ directly from the $3-2-1$ parts of $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$, i.e., $\boldsymbol{X}_i(0,0)$, $\boldsymbol{X}_i(0,1)$, $\boldsymbol{X}_i(0,2)$, $\boldsymbol{X}_i(1,0)$, $\boldsymbol{X}_i(1,1)$ and $\boldsymbol{X}_i(2,0)$, $i = 1,2,3,4$. Inverse motion compensation for other settings of partial DCT information, e.g., $DC+2AC$ case can be performed in a similar manner. The algorithm and simulation results are provided in Section 3.

For both problems, the common goal is to perform inverse motion compensation in the DCT domain given the inter and intraframe DCT information. Referring to Fig. 1, let the intersection of the reference block $\hat{\boldsymbol{x}}$ with $\boldsymbol{x}_1$ form a $h \times w$ rectangle (i.e., h rows and w columns), where $1 \leq h \leq 8$ and $1 \leq w \leq 8$. This means that the intersections of $\hat{\boldsymbol{x}}$ with $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, and $\boldsymbol{x}_4$ are rectangles of sizes $h \times (8-w)$, $(8-h) \times w$, and $(8-h) \times (8-w)$, respectively. Following Chang and Messerschmitt [8], [9] (see also [2], [4]), it is readily seen that $\hat{\boldsymbol{x}}$ can be expressed as a superposition of appropriate windowed and shifted versions of $\boldsymbol{x}_1, ...\boldsymbol{x}_4$. For integer pel accurate motion vectors, we can express $\hat{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{x}} = U_h \boldsymbol{x}_1 L_w + U_h \boldsymbol{x}_2 U_{8-w} + L_{8-h} \boldsymbol{x}_3 L_w + L_{8-h} \boldsymbol{x}_4 U_{8-w}, \tag{6}$$

where for $n = 1, 2, ..., 8$,

$$U_n \triangleq \begin{pmatrix} 0 & I_n \\ 0 & 0 \end{pmatrix} \tag{7}$$

and

$$L_n \triangleq \begin{pmatrix} 0 & 0 \\ I_n & 0 \end{pmatrix}, \tag{8}$$

I_n being the $n \times n$ identity matrix. Note that the general form for $\hat{\boldsymbol{x}}$ given in eq. (6) needs to be modified when h and w posess fractional pel accuracy. We develop this modification in Section 4.

Since the input blocks are given in the DCT domain, we shall rewrite eq. (6) as

$$\hat{\boldsymbol{x}} = U_h S^t \boldsymbol{X}_1 S L_w + U_h S^t \boldsymbol{X}_2 S U_{8-w} + L_{8-h} S^t \boldsymbol{X}_3 S L_w + L_{8-h} S^t \boldsymbol{X}_4 S U_{8-w}. \tag{9}$$

In Section 3, we will develop the algorithms for fast DCT-domain based inverse motion compensation for partial DCT information, specifically the $3-2-1$ case. In Section 4, we will use eq. (6) as the starting point to develop the algorithm for DCT-domain based inverse motion compensation for subpixel accurate motion-vectors.

3 Partial DCT Information

In this section, we develop an algorithm for DCT domain inverse motion compensation when only partial DCT information is used in the inverse motion compensation process. The algorithm is based on modifying the basic equation in eq. (6). The algorithm development here is based on the assumption that the motion-vectors are specified at only integer pel accuracy. The same concepts developed here extend easily to motion-vectors with fractional pel accuracy as is the case in MPEG-1 and MPEG-2. A brief description of the approach that can be adopted for DCT-domain inverse-motion compensation for fractional pel accurate motion-vectors is given in Section 4.

3.1 Mathematical Derivation

In [4], we developed an algorithm to efficiently implement eq. (9). Now, since only the $3-2-1$ part of each $\boldsymbol{X}_i$ is assumed non-zero (see Fig. 2), eq. (9) is equivalent to

$$\hat{\boldsymbol{x}} = U_h S^t T \boldsymbol{X}_1 T S L_w + U_h S^t T \boldsymbol{X}_2 T S U_{8-w} + L_{8-h} S^t T \boldsymbol{X}_3 T S L_w + L_{8-h} S^t T \boldsymbol{X}_4 T S U_{8-w}, \quad (10)$$

where

$$T \triangleq \begin{pmatrix} I_3 & 0 \\ 0 & 0 \end{pmatrix}. \quad (11)$$

DCT-domain inverse motion compensation for the $3-2-1$ case can be performed by straightforward application of the algorithm for eq. (9) as described in [4]. However, eq. (10), suggests that a faster algorithm might be possible by combining the operation of T with these of the DCT and the window and shift operation in one of two different approaches. One approach, which is preferrable when U_n or L_n are associated with a relatively small n, is simply to precompute the fixed products $U_n S^t T$, $L_n S^t T$, TSU_n, and TSL_n for all n, and to use these in a straightforward manner. The second approach, to be used if n is relatively large, is to embed T into a sparse matrix factorization of S. Specifically, similarly as in [4], we shall use a factorization of S that corresponds to the 8-point Winograd DCT due to Arai, Agui, and Nakajima [10] (see also [11]). According to this factorization, S is represented as follows.

$$S = DPB_1B_2MA_1A_2A_3 \quad (12)$$

where D is a diagonal matrix given by

$$D = \mathrm{diag}\{0.3536, 0.2549, 0.2706, 0.3007, 0.3536, 0.4500, 0.6533, 1.2814\}, \quad (13)$$

P is a permutation matrix given by

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{14}$$

and the remaining matrices are defined as follows:

$$B_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}, B_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}, \tag{15}$$

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.9239 & 0 & -0.3827 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.3827 & 0 & 0.9239 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{16}$$

$$A_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{17}$$

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{18}$$

and

$$A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}. \tag{19}$$

It is easy to see that pre-multiplication by T causes the factors of S to degenerate to even sparser matrices. Specifically,

$$TS = TDPB_1B_2(MA_1A_2A_3) = \hat{D}\hat{P}\hat{B}_1\hat{B}_2G, \tag{20}$$

where

$$\hat{D} = \text{diag}\{0.3536, 0.2549, 0.2706, 0, 0, 0, 0, 0\}, \tag{21}$$

$$\hat{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{22}$$

$$\hat{B}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \hat{B}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{23}$$

and

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (1+a) & a & -a & (-1-a) & (-1-a) & -a & a & (1+a) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & a & a & 0 & 0 & -a & -a & -1 \\ c & c & b & b & -b & -b & -c & -c \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad (24)$$

where $a = 0.7071$, $b = 0.9239$, and $c = 0.3827$. Now, the idea is that post-multiplication by $\hat{P}$, $\hat{B}_1$, and $\hat{B}_2$, are all computationally costless. Since the multiplication by $\hat{D}$ can be absorbed in the de-quantization step, the computational bottleneck remains merely in the multiplication by $J_n = GL_n$ and $K_n = GU_n$, which are very structured, and can be implemented similarly as in [4, Sect. 4].

For $n \leq 3$, multiplication by J_n (K_n) costs more than multiplication by the entire precomputed operator matrix TSL_n (TSU_n), which takes n multiplications and n additions per one 8-vector (again, by taking advantage of the de-quantizer). Thus, for these values of n, a full precomputation will be preferrable. For $n \geq 4$, the factorization approach is more efficient, and gives the following figures per one column vector: For $n = 4$: 4 multiplications and 10 additions, $n = 5$: 4 multiplications and 11 additions, $n = 6$: 4 multiplications and 13 additions, $n = 7$: 4 multiplications and 14 additions, and $n = 8$: 4 multiplications and 15 additions. The same is true for pre-multiplication by the transposed matrices U_nS^tT and L_nS^tT. Now, similarly, as in [4], if the above factorization of S is incorporated into eq. (10), we obtain

$$\begin{aligned} \hat{\boldsymbol{x}} &= J_h^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}(\boldsymbol{X}_1\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w + \boldsymbol{X}_2\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}) + \\ &\quad K_{8-h}^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}(\boldsymbol{X}_3\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w + \boldsymbol{X}_4\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}), \end{aligned} \qquad (25)$$

or, equivalently,

$$\begin{aligned} \hat{\boldsymbol{x}} &= (J_h^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}\boldsymbol{X}_1 + K_{8-h}^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}\boldsymbol{X}_3)\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w + \\ &\quad (J_h^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}\boldsymbol{X}_2 + K_{8-h}^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}\boldsymbol{X}_4)\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}. \end{aligned} \qquad (26)$$

The heart of the proposed inverse motion compensation algorithm lies in the implementation of either eq. (25) or (26), whichever requires less computations for the given w and h. Again, as explained above, the products of the factors are precomputed when L_n and U_n correspond to $n \leq 3$. For applications that require a compressed domain output (e.g., when eq. (25) or eq. (26) is carried out by the server), the resultant $\hat{\boldsymbol{x}}$ has to be transformed back to the DCT domain.

3.2 The Algorithm

Both eqs. (25) and (26) are structured in the sense that they are formed by repetitions of a basic set of operations associated with multiplying one matrix by J_n, another matrix by

K_{8-n}, and then adding the results. (The latter addition is trivial since the summands are nonzero on disjoint subsets of entries.) We shall refer to such a computational set as an *n-set*. Since eq. (25) contains two w-sets and one h-set, while eq. (26) has one w-set and two h-sets, the choice between the two equations depends on whether an h-set or a w-set requires less computations. For the $3-2-1$ case, the matrix that multiplies with J_n or K_{8-n} has three types of vectors. We denote the number of multiplications and additions per vector as m_n^i and a_n^i (the superscript denotes the vector type and takes on values 1,2 and 3). The multiplication and addition cost per vector is:

- For vector type 3, we have $m_0^3 = m_8^3 = 4$, $m_1^3 = m_7^3 = 6$, $m_2^3 = m_6^3 = 7$, $m_3^3 = m_4^3 = m_5^3 = 8$ and $a_0^3 = a_8^3 = 15$, $a_1^3 = a_2^3 = a_6^3 = a_7^3 = 17$, $a_3^3 = a_5^3 = 19$ and $a_4^3 = 20$.
- For vector type 2, we have $m_0^2 = m_8^2 = 3$, $m_1^2 = m_7^2 = 4$, $m_2^2 = m_6^2 = 5$, $m_3^2 = m_4^2 = m_5^2 = 6$ and $a_0^2 = a_1^2 = a_4^2 = a_7^2 = a_8^2 = 10$, and $a_2^2 = a_3^2 = a_5^2 = a_6^2 = 9$.
- For vector type 1, no multiplications and additions are needed.

Thus, for example if the cost of a multiplication is equivalent to that of α additions, then if $\alpha m_w^i + a_w^i \leq \alpha m_h^i + a_h^i$ implement eq. (25), else implement eq. (26).

The computation algorithm is then summarized as follows: (In parenthesis we provide the operations count in the form of an expression $mM + aA$, which means m multiplications and a additions)

1. Decide on whether eq. (25) or eq. (26) is implemented. In the former case, perform steps 2-8 and terminate. In the latter case, go to 9.
2. Post-multiply the first row of $\boldsymbol{X}_1$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the first row of $\boldsymbol{X}_2$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(m_w^3M + a_w^3A)$
3. Post-multiply the first row of $\boldsymbol{X}_3$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the first row of $\boldsymbol{X}_4$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(m_w^3M + a_w^3A)$
4. Post-multiply the second row of $\boldsymbol{X}_1$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the second row of $\boldsymbol{X}_2$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(m_w^2M + a_w^2A)$
5. Post-multiply the second row of $\boldsymbol{X}_3$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the second row of $\boldsymbol{X}_4$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(m_w^2M + a_w^2A)$
6. Post-multiply the third row of $\boldsymbol{X}_1$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the third row of $\boldsymbol{X}_2$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(0M + 0A)$
7. Post-multiply the third row of $\boldsymbol{X}_3$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2J_w$ and the third row of $\boldsymbol{X}_4$ by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2K_{8-w}$. $(0M + 0A)$
8. Pre-multiply the result of 2, 4 and 6 by $J_h^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}$, and the result of 3, 5 and 7 by $K_{8-h}^t\hat{B}_2^t\hat{B}_1^t\hat{P}^t\hat{D}$. $(8m_h^3M + 8a_h^3A)$

9. Pre-multiply the first column of $\boldsymbol{X}_1$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the first column of $\boldsymbol{X}_3$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(m_h^3 M + a_h^3 A)$

10. Pre-multiply the first column of $\boldsymbol{X}_2$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the first column of $\boldsymbol{X}_4$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(m_h^3 M + a_h^3 A)$

11. Pre-multiply the second column of $\boldsymbol{X}_1$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the second column of $\boldsymbol{X}_3$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(m_h^2 M + a_h^2 A)$

12. Pre-multiply the second column of $\boldsymbol{X}_2$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the second column of $\boldsymbol{X}_4$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(m_h^2 M + a_h^2 A)$

13. Pre-multiply the third column of $\boldsymbol{X}_1$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the third column of $\boldsymbol{X}_3$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(0M + 0A)$

14. Pre-multiply the third column of $\boldsymbol{X}_2$ by $J_h^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$ and the third column of $\boldsymbol{X}_4$ by $K_{8-h}^t \hat{B}_2^t \hat{B}_1^t \hat{P}^t \hat{D}$. $(0M + 0A)$

15. Post-multiply the result of 9, 11 and 13 by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2 J_w$, and the result of 10, 12 and 14 by $\hat{D}\hat{P}\hat{B}_1\hat{B}_2 K_{8-w}$. $(8m_w^3 M + 8a_w^3 A)$

The computations associated with steps 6, 7, 13, and 14 are not counted since each one of them involves only one multiplication by a fixed number, which can be absorbed in the de-quantization stage. The total number of computations is then $(8m_h^3 + 2m_w^3 + 2m_w^2)$ multiplications and $(8a_h^3 + 2a_w^3 + 2a_w^2)$ additions if steps 2-7 are performed, and $(8m_w^3 + 2m_h^3 + 2m_h^2)$ multiplications and $(8a_w^3 + 2a_h^3 + 2a_h^2)$ additions if steps 7-11 are performed. In case of perfect alignment of the motion vector in one dimension, e.g., $h = 8$, only two terms in eqs. (25) and (26) remain nonzero, and the total number of computations is $(8m_8^3 + m_w^3)$ multiplications and $(8a_8^3 + a_w^3 + a_w^2)$ additions. If both $w = 8$ and $h = 8$, then only one term remains and then $9m_8^3$ multiplications and $9a_8^3$ additions are needed. Averaging with respect to a uniform distribution over w and h, gives 72.83 multiplications and 186.95 additions. Note that for the $DC + 2AC$ case, a similar analysis yields 41.95 multiplications and 93.01 additions. The uniform distribution is however, a conservative assumption. From the simulations for a video test sequence of 150 frames taken from a test set used by the MPEG standards committee, the actual joint probability distribution of w and h has the profile shown in Fig. 3. Note that the distribution is more peaked at the corners, i.e. near $h = 8$ or $w = 8$. Referring to Fig. 1, $h = 8$ or $w = 8$ implies that for a large number of $\hat{\boldsymbol{x}}$, only one or at most two of $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3, \boldsymbol{X}_4$ in eq. (25) or eq. (26) are used which in turn leads to fewer multiplies and adds than the numbers stated here.

The DCT domain approach developed here for the $3 - 2 - 1$ case has significantly lower operations count than a naive spatial-domain approach wherein four full inverse DCT operations requiring 320 multiplications and 1856 additions (one full inverse DCT based on the Arai and Agui approach[10] requires 80 multiplications and 464 additions) have to be performed. A sophisticated spatial-domain approach can exploit the fact that a full inverse DCT need not be performed for the $3 - 2 - 1$ case. The complexity for such an approach is 128 multiplications and 632 additions. This has significantly higher computational cost compared with the proposed DCT-domain approach.

If the desired output is in the DCT domain, then as mentioned earlier, one has to perform DCT on the resulting $\hat{\boldsymbol{x}}$, which requires 80 more multiplications and 464 more additions in general. If, however, $\hat{\boldsymbol{X}}$ is approximated in the $3-2-1$ form as well, then using the same technique, the complexity of the DCT stage can be reduced to 55 multiplications and 198 additions, which overall gives 127.83 multiplications and 384.95 additions on the average over all motion vectors. For the $DC + 2AC$ case, the overall cost is 71.95 multiplications and 273 additions.

A more detailed comparison of the proposed DCT-domain approach and the naive spatial-domain approach as well as the sophisticated spatial-domain approach is provided in Section 3.4.4.

3.3 An Approximate Multiplication-Free Algorithm

An approximate multiplication-free version of our algorithm is associated with quantization of the entries a, b and c of the matrix G. According to this approximation, a, b and c, are replaced by $\hat{a}$, $\hat{b}$, and $\hat{c}$, respectively, given by $\hat{a} = 0.75$, $\hat{b} = 1$, and $\hat{c} = 0.375$, whose multiplications can be implemented by one SHIFT & ADD operation on some general purpose CPUs with multimedia-enhanced instruction sets [13]. For the $3-2-1$ case, the cost of computing $\hat{x}$ according to eq. (25) or (26) using the quantized G is approximately 260 additions. For the $DC + 2AC$ case, we require 170 additions.

To roughly assess the worst case SNR associated with this approximation, let $\hat{S} = S + \Delta$ denote the approximated DCT operator matrix associated with $\hat{a}$, $\hat{b}$, and $\hat{c}$ in place of a, b, and c, respectively. Every term in eq. (25) and (26) is a shifted and windowed version of $S^tT\boldsymbol{X}_iTS$, $i = 1, 2, 3, 4$, which when replaced by $\hat{S}^tT\boldsymbol{X}_iT\hat{S}$, gives

$$\hat{S}^tT\boldsymbol{X}_iT\hat{S} = S^tT\boldsymbol{X}_iTS + S^tT\boldsymbol{X}_iT\Delta + \Delta^tT\boldsymbol{X}_iTS + \Delta^tT\boldsymbol{X}_iT\Delta, \qquad (27)$$

where the first term is the desired term and the three other terms are error terms. Transforming the last equation to a column stacked form [12, Sections 5.3-5.4], the desired term is given by UX_i^{cs}, while the error term is given by VX_i^{cs}, where X_i^{cs} is the column-stacked version of $\boldsymbol{X}_i$ (that is, a 64-dimensional column vector formed by concatenating the columns of $\boldsymbol{X}_i$ from left to right), $U = (S^tT) \otimes (S^tT)$, $V = (\Delta^tT) \otimes (S^tT) + (S^tT) \otimes (\Delta^tT) + (\Delta^tT) \otimes (\Delta^tT)$, and $\otimes$ denotes the Kroenecker tensor product. By a simple numerical analysis, we find that the worst case SNR given by $10\log_{10}[\mathrm{tr}(U^tU)/\mathrm{tr}(V^tV)]$ is about 24dB. This means that the additional distortion due to this quantization is typically small compared to the distortion induced by the $3-2-1$ structure and the quantization error due to the lossy compression.

This worst case SNR estimation applies only to one-step prediction, i.e. when only frame is inverse motion-compensated. In practice all video coding standards employ multi-step prediction within a group of 12-16 pictures and in this context, each frame within this group is predicted from a previously reconstructed frame which is also within this group. Thus there will be error buildup since each frame is reconstructed inaccurately due to the quantization of G and the worst case SNR will be lower than 24dB. However, due to the use of partial DCT information in each $\boldsymbol{X}_i$, $i = 1, 2, 3, 4$, the error buildup will not cause the SNR to degrade significantly. We will demonstrate this in the simulations described in Section 3.4.3.

3.4 Experimental Results

Several simulations were performed to assess the image quality for DCT-domain based inverse motion compensation approach. The primary objective of the simulations was to investigate the tradeoffs between the operations count and image fidelity.

For the simulations, the basic steps employed in MPEG coding were used to generate the intraframe and interframe DCT information. The encoder used in the simulations is as shown in Fig. 4 and performs the functions of an MPEG encoding scheme. We have, however, simplified the coder by using a single quantizer characteristic for all blocks in the frame and the motion-estimation strategy restricts the motion-vectors to integer pixel accuracy.

3.4.1 Two-frame Simulations

The encoder of Fig. 4 was used to encode a group of two frames. The two frames used in this set are shown in Fig. 5. Note that the encoder outputs only the DCT information for the I frame and the motion-compensated prediction residual for the P frame.

For the frame tensif.66 coded as a P frame in Fig. 5, the corresponding reconstructed I frame representation is shown in Fig. 6. Here, we show the I frame reconstruction for various cases:

1. ten8x8.exact: all 64 coefficients of the four blocks $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3, \boldsymbol{X}_4$ are used,
2. ten4x4.exact: 4×4 case,
3. ten321.exact: $3-2-1$ case,
4. ten2x2.exact: 2×2 case,
5. ten2x1.exact: the $DC + 2AC$ case, and
6. tendc.exact: the DC only case.

For the $3-2-1$ case, the intraframe reconstruction was based on eq. (25) or eq. (26). Similar algorithms can be devised for the 4×4, 2×2 and the $DC + 2AC$ case.

This simulation suggests that the $DC{+}2AC$ case may be acceptable in browsing applications.

3.4.2 Error Propagation Due To Partial DCT Information

The simulation of Fig. 6 is not realistic since we have used a simple two frame IP sequence, whereas, in most MPEG coding systems, a group of pictures might consist of one intraframe and perhaps 12-15 P or B frames. Since the intraframe reconstructions of previous interframes is used in the reconstructions of subsequent frames, there is a potential for error propagation. The error propagation effect will be more pronounced when partial DCT information is used in the inverse motion compensation process.

In order to assess the quality of image reconstructions due to error propagation when partial DCT information is used, we used a group of 16 frames and applied the encoding procedure of Fig. 4. The coding type for this group was set to IBBPBBPBBPBBPBBP, i.e. 2 B frames for each P frame. For this group of 16 frames, erroneous reconstruction of each P frame due to partial DCT information will cause the largest error buildup in the 14th and 15th frame. In Fig. 7, we show the image reconstruction for the 14th frame. The original frame (input to the encoder of Fig. 4) is tensif.66 shown in Fig. 5.

This simulation suggests that in a multiframe setting for MPEG coding, the $DC + 2AC$ case may not be adequate even for browsing applications when browsing is done at the original spatial resolution. If the reconstructed image is further down-sampled by a factor of two in each direction [2] ((see also [1], [3]), the $DC + 2AC$ may be an adequate choice for browsing. For browsing while maintaining the original spatial resolution, the $3 - 2 - 1$ case (ten321p.exact in Fig. 7) is needed for acceptable image quality.

We have repeated these experiments for the approximate multiplication-free algorithm described in section 3.3. The simulation results are shown in Fig. 8 and Fig. 9.

Note that the only difference between Fig. 6 and Fig. 8 is that in the latter, a quantized G matrix is used in eq. (25) or eq. (26); comparing the image reconstruction results indicates no additional degradations due to the use of the approximate multiplication-free method. We can make the same observation when comparing Fig. 7 and Fig. 9. Thus, in low-cost hardware or software-only implementations, the proposed multiplication-free method is preferable.

3.4.3 Signal-to-Noise Ratio Results

The results of these experiments are summarized in Fig. 10 wherein, we show the PSNR (peak-to-peak signal-to-noise ratio) for various cases: full/partial DCT information, exact DCT or the approximate multiplication-free method, two-frame IP sequence as well as the 16 frame IBBP,...P sequence.

Note that there is significant PSNR degradations when partial DCT information is used; for instance, using a two frame packet and the exact representation for the matrix G, the 8×8 case has a PSNR of 35.63dB whereas the $3-2-1$ case has a PSNR of 22.89dB. Thus, partial DCT information based processing is usable only for browsing applications or for classifying images for the purposes of scene change detection. From this figure, we also observe that the PSNR degradations using the approximate multiplication-free method leads to little or no PSNR degradations when partial DCT information is used for inverse motion-compensation. The PSNR degradation due to error propagation is around 2dB for the $DC + 2AC$ relative to the $3-2-1$ case; perceived image quality degradation is quite substantial for 2dB loss in PSNR. The PSNR numbers also suggest that in many applications wherein multiple frames are used in a group of pictures during encoding, inverse motion compensation using around 6 DCT coefficients such as the $3 - 2 - 1$ case may be preferable to the 3 DCT coefficients that is used in the $DC + 2AC$ case.

We have also compared the performance of our method against the proposed DC only scheme of Yeo and Liu [2] and [3]. For a two-frame IP sequence, our approach yields better quality images and has a 0.5dB better PSNR. Furthermore, in our work, we have considered the effects of error propagation due to image reconstruction from partial DCT information; in [2] and [3], there is no discussion of error propagation effects in their scheme and we believe that if error propagation effects were taken into account in the work in [2] and [3], the conclusions regarding the effectiveness of a DC only reconstruction scheme may be different than that reported by Yeo and Liu.

3.4.4 Computation Complexity

¿From an image quality viewpoint, the experimental results we have described so far suggests that the $DC + 2AC$ case is barely acceptable for many applications and in fact the $3 - 2 - 1$ case might be a better choice. The computation complexity associated with each choice of the partial DCT information setting (full, 4x4, 2x2, etc.) is shown in Table 1.

DCT Information	Domain	Computation Cost						
		$\hat{\boldsymbol{x}}_m$	$\hat{\boldsymbol{x}}_a$	$\boldsymbol{e}_m$	$\boldsymbol{e}_a$	$(\hat{\boldsymbol{x}} + \boldsymbol{e})_m$	$(\hat{\boldsymbol{x}} + \boldsymbol{e})_a$	Total
8×8	DCT							2364.5[4]
	spatial(1)	320	1856	80	464	400	2384	4384
	spatial(2)	320	1856	80	464	400	2384	4384
4×4	DCT							1091.2[4]
	spatial(1)	240	984	60	246	300	1230	1830
	spatial(2)	320	1856	80	464	400	2384	4384
$3 - 2 - 1$	DCT	72.83	186.95	40	165	112.83	351.95	577.61
	spatial(1)	128	632	40	165	168	861	1197
	spatial(2)	320	1856	80	464	400	2384	4384
2×2	DCT	72	160	30	120	102	344	548
	spatial(1)	120	480	30	120	150	664	964
	spatial(2)	320	1856	80	464	400	2384	4384
$DC + 2AC$	DCT	41.95	93.01	28	108	69.95	265.01	480.01
	spatial(1)	112	432	28	108	140	604	884
	spatial(2)	320	1856	80	464	400	2384	4384
DC only	DCT							17
	spatial(1)	4		1		5	4	14
	spatial(2)	320	1856	80	464	400	2384	4384

Table 1: Operations count for DCT-domain and spatial-domain inverse motion compensation using full(8×8) or partial DCT information. The subscript m refers to multiplication count and the subscript a refers to add count. The column labelled **Total** reflects the operations count when an approximate multiplication-free method is used in the inverse DCT calculations.

Note that the spatial approach requires calculating four inverse DCTs for the blocks $\boldsymbol{X}_i$, $i = 1, .., 4$ before performing inverse motion compensation. A naive approach in the spatial domain (referred to as **spatial(2)** in Table 1) will not make any sparseness assumptions on the contents of $\boldsymbol{X}_i$, $i = 1, .., 4$. If sparseness of $\boldsymbol{X}_i$, $i = 1, .., 4$ is exploited during the inverse DCT calculations for the spatial-domain approach (referred to as **spatial(1)** in Table 1) the computation complexity relative to the naive approach is substantially reduced. In all but the DC only case, the DCT domain approach for inverse motion-compensation has lower complexity compared with the spatial domain approaches.

¿From an operations count viewpoint, the 3-2-1 case has nearly eight-fold reduction in computation complexity relative to the naive spatial-domain inverse motion-compensation method for the full 8×8 DCT case. The $DC + 2AC$ case has nine-fold reduction in computation complexity compared with the full 8×8 DCT case; however, the resulting image quality may be acceptable only for a limited set of applications. Since the image quality resulting from the $3-2-1$ case is better than the $DC+2AC$ case and the computation savings using $DC+2AC$ is not significant, the $3-2-1$ case may be preferred for most applications. The results of Table 1 are depicted in Fig. 11.

The simulation results presented in Fig. 9, Fig. 10 and the operations count in Fig. 11 can be used to make a careful tradeoff between the computation complexity and the desired image quality for the specific application.

4 Noninteger Motion Vectors

As explained earlier in Section 1, MPEG encoding can generate noninteger motion vectors, more precisely, vectors whose components are integer multiples of 1/2. If both components are nonintegers, this mean that each pixel in the reference macroblock is the average of the four nearest neighbor pixels of integer-valued coordinates. If only one component of the motion vector is noninteger, then the average of only two nearest neighbor pixels is taken.

In this section, we show a simple way to modify the inverse motion compensation algorithm developed earlier so as to deal with noninteger motion vectors. If the motion vectors are such that the intersection of the reference block (see Fig. 1) with $\hat{\boldsymbol{x}}_1$ has parameters $h+1/2$ and $w+1/2$ (h and w integers), then $\hat{\boldsymbol{x}}$ is given by the average of four contributions corresponding to the pairs (h,w), $(h+1,w)$, $(h,w+1)$ and $(h+1,w+1)$. Specifically, eq. (9) is rewritten as

$$\begin{aligned}
\hat{\boldsymbol{x}} &= \frac{1}{4}[U_h S^t \boldsymbol{X}_1 S L_w + U_h S^t \boldsymbol{X}_2 S U_{8-w} + L_{8-h} S^t \boldsymbol{X}_3 S L_w + L_{8-h} S^t \boldsymbol{X}_4 S U_{8-w} + \\
&\quad U_{h+1} S^t \boldsymbol{X}_1 S L_w + U_{h+1} S^t \boldsymbol{X}_2 S U_{8-w} + L_{7-h} S^t \boldsymbol{X}_3 S L_w + L_{7-h} S^t \boldsymbol{X}_4 S U_{8-w} + \\
&\quad U_h S^t \boldsymbol{X}_1 S L_{w+1} + U_h S^t \boldsymbol{X}_2 S U_{7-w} + L_{8-h} S^t \boldsymbol{X}_3 S L_{w+1} + L_{8-h} S^t \boldsymbol{X}_4 S U_{7-w} + \\
&\quad U_{h+1} S^t \boldsymbol{X}_1 S L_{w+1} + U_{h+1} S^t \boldsymbol{X}_2 S U_{7-w} + L_{7-h} S^t \boldsymbol{X}_3 S L_{w+1} + L_{7-h} S^t \boldsymbol{X}_4 S U_{7-w} \\
&= \frac{1}{4}[(U_h + U_{h+1}) S^t \boldsymbol{X}_1 S (L_w + L_{w+1}) + (U_h + U_{h+1}) S^t \boldsymbol{X}_2 S (U_{8-w} + U_{7-w}) + \\
&\quad (L_{8-h} + L_{7-h}) S^t \boldsymbol{X}_3 S (L_w + L_{w+1}) + (L_{8-h} + L_{7-h}) S^t \boldsymbol{X}_4 S (U_{8-w} + U_{7-w})] \\
&= \frac{1}{4}[(I_8 + U_7) U_{h+1} S^t \boldsymbol{X}_1 S L_{w+1} (I_8 + L_7) + (I_8 + U_7) U_{h+1} S^t \boldsymbol{X}_2 S U_{8-w} (I_8 + U_7) + \\
&\quad (I_8 + L_7) L_{8-h} S^t \boldsymbol{X}_3 S L_{w+1} (I_8 + L_7) + (I_8 + L_7) L_{8-h} S^t \boldsymbol{X}_4 S U_{8-h} (I_8 + U_7)] \\
&= \frac{1}{4}\{(I_8 + U_7) J_{h+1}^t B_2^t B_1^t P^t D [\boldsymbol{X}_1 D P B_1 B_2 J_{w+1} (I_8 + L_7) + \\
&\quad \boldsymbol{X}_2 D P B_1 B_2 K_{8-w} (I_8 + U_7)] + \\
&\quad (I_8 + L_7) K_{8-h}^t B_2^t B_1^t P^t D [\boldsymbol{X}_3 D P B_1 B_2 J_{w+1} (I_8 + L_7) + \\
&\quad \boldsymbol{X}_4 D P B_1 B_2 K_{8-w} (I_8 + U_7)]\}
\end{aligned} \tag{28}$$

or, equivalently,

$$\hat{\boldsymbol{x}} = \frac{1}{4}\{[(I_8 + U_7) J_{h+1}^t B_2^t B_1^t P^t D \boldsymbol{X}_1 +$$

$$(I_8 + L_7)K^t_{8-h}B^t_2B^t_1P^tD\boldsymbol{X}_3]DPB_1B_2J_{w+1}(I_8 + L_7) +$$
$$[(I_8 + U_7)J^t_{h+1}B^t_2B^t_1P^tD\boldsymbol{X}_2 +$$
$$(I_8 + L_7)K^t_{8-h}B^t_2B^t_1P^tD\boldsymbol{X}_4]DPB_1B_2K_{8-w}(I_8 + U_7)\}, \tag{29}$$

where the choice between the two forms depends again on the overall number of computations associated with its implementation for the given pair (h, w). Note that the computational compexity associated with the implementation of each one of these equations is only slightly larger than the algorithm in [4] as multiplication by $(I_8 + U_7)$ or $(I_8 + L_7)$ involves at most only 7 additions per vector, and no extra multiplications at all.

In a similar manner, if w is integer and only h is noninteger, then the corresponding expression for $\hat{\boldsymbol{x}}$ will involve only premultiplications by (I_8+U_7) and (I_8+L_7), and if only w is noninteger then only post-multiplications by these matrices are needed.

7 References

[1] M. M. Yeung, B.-L. Yeo, W. Wolf, and B. Liu, "Video browsing clustering and scene transitions on compressed sequences," preprint 1995.

[2] B.-L. Yeo and B. Liu, "Reconstruction of spatially subsampled sequences from MPEG compressed video," submitted to *IEEE Trans. on Image Processing.*

[3] B.-L. Yeo and B. Liu, "Rapid scene analysis on compressed video," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 5, no. 6, pp. 533-544, Dec. 1995.

[4] N. Merhav and V. Bhaskaran, "A Fast Algorithm for DCT-Domain Inverse Motion Compensation," *Proc. ICASSP '96*, pp. IV.2307-2310, Atlanta, May 1996.

[5] Coding of Moving and Associated Audio. Committee Draft of Standard ISO11172: ISO/MPEG 90/176, December 1990.

[6] Video Codec for Audio Visual Services at px64 Kbits/s. CCITT Recommendation H.261, 1990.

[7] D. le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Commun. of the ACM*, Vol. 34, No. 4, pp. 47-58, April 1991.

[8] S.-F. Chang and D. G. Messerschmitt, "A New Approach to Decoding and Compositing Motion-Compensated DCT Based Images," *Proc. ICASSP '93*, pp. V.421-V.424, Minneapolis, April 1993.

[9] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Selected Areas in Communications*, Vol. 13, no. 1, pp. 1-11, January 1995.

[10] Y. Arai, T. Agui, and M. Nakajima, "A Fast DCT-SQ Scheme for Images," *Trans. of the IEICE*, E 71(11):1095, November 1988.

[11] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[12] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, second edition, 1991.

[13] V. Bhaskaran, and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, Chapter 15, Kluwer Academic Publishers, , Sept. 1995.

Figure 1: DCT domain based inverse motion compensation.

Figure 2: Partial DCT information. **x** refers to locations in the 8×8 DCT matrix wherein the DCT coefficients can take on nonzero values. DCT coefficients are assumed to be zero-valued in locations not marked with an **x**.

Figure 3: Probability distribution for h, w.

Figure 4: Video encoder structure used in the simulations.

Figure 5: Test video frames used in a two frame simulation. Frame tensif.65 is coded as an intra (I) frame. Frame tensif.66 is coded as a predictive (P) frame and its motion-compensated prediction residual is shown in Frame tensif65_66.diff. For motion-compensated prediction residual, a reconstructed version tensif_65.rec of Frame tensif.65 is used.

Figure 6: DCT domain inverse motion compensated reconstruction using full or partial DCT information for the two frame IP sequence shown in Fig. 5. The original version of the reconstructed frame is tensif.66 shown in Fig. 5.

Figure 7: DCT domain inverse motion compensated reconstruction for a 16 frame IBBP...P sequence and using full or partial DCT information. The results shown are for the 14^{th} frame of the sequence. The original version of this frame is tensif.66 shown in Fig. 5.

Figure 8: DCT domain inverse motion compensated reconstruction for the two frame IP sequence using the approximate multiplication-free method. The original version of the reconstructed frame is tensif.66 shown in Fig. 5.

Figure 9: DCT domain inverse motion compensated reconstruction for the 16 frame IBBP...P sequence using the approximate multiplication-free method. The results shown are for the 14^{th} frame. The original version of this frame is tensif.66 in Fig. 5. The DCT domain reconstruction is shown for the case of full and partial DCT information.

Figure 10: PSNR versus full or partial DCT information for the DCT domain based inverse motion compensation method. In this figure, **Exact** referes to the use of an exact representation for matrix G whereas **Quantized** refers to the use of the approximate multiplication-free method. Also **I,P** refers to the use of a two frame IP sequence and the PSNR is for the 2^{nd} frame. **I,P,..P** refers to the use of a 16 frame IBBP...P sequence and in this case the PSNR for the 14^{th} frame is shown.

Figure 11: Computation complexity for DCT-domain and spatial-domain inverse motion compensation using full or partial DCT information. Spatial(2) refers to a naive spatial-domain approach whereas Spatial(1) refers to a spatial-domain approach wherein the DCT sparseness is exploited in the inverse DCT computation.