Journal of Visual Languages and Computing (1999) 10, 607–623
Article No. jvlc.1999.0149, available online at http://www.idealibrary.com on IDEAL®

A Fast Algorithm for Reconstructing Motion-Compensated Blocks in Compressed Domain

HONGZHENG LI AND HONGCHI SHI

Department of Computer Engineering and Computer Science, University of Missouri-Columbia, Columbia, MO 65211, U.S.A., li@ece.missouri.edu, shi@cecs.missouri.edu

Accepted 18 August 1999

Many advanced video applications require processing compressed video signals. For compression systems using the discrete cosine transform (DCT) with motion compensation, we propose an algorithm with adaptive low-pass filtering (ALPF) to reconstruct blocks with motion vectors in the compressed domain. Compared with the previous work, this algorithm is faster and reduces blocky artifacts caused by quantization.
© 1999 Academic Press

Keywords: discrete cosine transform, motion compensation, compressed domain, low-pass filtering

1. Introduction

THERE ARE TWO FUNDAMENTAL ways to process compressed video. One way is by decompressing video frames to gather the information in the pixel domain. The other way involves exploiting the encoded information contained in the compressed representation. Processing video directly in the compressed domain has advantages: smaller data volume and less-expensive computation.

The DCT is the key compression technique used in the current image and video compression standards. Since most of the signal energy concentrates on a few DCT coefficients, we can process data more efficiently in the DCT domain. The DCT coefficients of the *I* frames in any MPEG video can be used directly, while the DCT coefficients of the motion-compensated blocks in the *P* and *B* frames have to be reconstructed from the four neighboring blocks in the reference frames.

In this paper, we devise a fast algorithm by using low-pass filtering to implement motion compensation directly in the DCT domain. The algorithm accelerates the processing time by avoiding the unnecessary computation and reduces the blocky artifacts due to quantization and using partial DCT information. In the following sections, we first introduce the basic techniques used in MPEG and related research work in this area. We then present the details of our ALPF algorithm. We finally discuss some experimental results.

1045-926X/99/120607+17 $30.00/0 © 1999 Academic Press

2. Background

2.1. MPEG Overview

An MPEG video sequence consists of three types of encoded frames: intraframes (*I* frames), predicted frames (*P* frames), and interpolated bi-directional frames (*B* frames) [1]. The *I* frames are coded independently, without reference to other frames. The *P* frames are encoded by motion prediction using past *I* or *P* frames. In the *B* frames, the reference frames for motion prediction are past and future frames.

The basic building block of an MPEG video is macroblock. A macroblock consists of a 16×16 array of luminance samples together with an 8×8 block of samples for each of two chrominance components. The 16×16 sample array is actually composed of four 8×8 blocks of samples. These 8×8 blocks are basic units for DCT compression.

Motion compensation is used for interframe compression. In *P* or *B* frames, if the motion-compensated prediction difference is larger than certain threshold for a macroblock, then the macroblock is actually intracoded. Otherwise, the difference itself is intracoded and transmitted along with the motion vectors which indicate locations of the reference areas.

The two dimensional (2D) discrete cosine transform (DCT) is used as the basis of compression of each *I* frame and the residue images from *P* frames and *B* frames. The image is first divided into 8×8 blocks, and each block is transformed by the DCT into the DCT coefficients. The forward DCT and inverse DCT (IDCT) are defined as follows [2]:

$$c(i,j) = \frac{1}{4}k(i)k(j)\left[\sum_{m=0}^{7}\sum_{n=0}^{7} f(m,n)\cos\frac{(2m+1)i\pi}{16}\cos\frac{(2n+1)j\pi}{16}\right]$$

$$f(i,j) = \frac{1}{4}\left[\sum_{m=0}^{7}\sum_{n=0}^{7} k(m)k(n)c(m,n)\cos\frac{(2i+1)m\pi}{16}\cos\frac{(2j+1)n\pi}{16}\right]$$

where $i, j = 0, 1, \ldots, 7$ and

$$k(i) = \begin{cases} \dfrac{1}{\sqrt{2}}, & i = 0 \\ 1 & \text{otherwise} \end{cases}$$

The DCT is a method of decomposing a block of data into a weighted sum of spatial frequencies. The DCT transform decorrelates the original signal, which generally results in the signal energy being redistributed among only a small set of transform coefficients.

2.2. Related Work

Video data is typically stored in a compressed format. If we directly analyze the compressed representation, we can avoid the costly overhead of decompressing and processing at the pixel level. In the recent years, there have been many efforts in

developing fast algorithms for processing compressed video streams in the DCT domain [3–17].

The reconstruction methods in those algorithms fall into two categories: exact reconstruction and approximate reconstruction. For video manipulation, we have to exactly reconstruct all the DCT coefficients in the blocks with motion vectors. For video retrieval and scene detection, due to the huge data volume, many researchers exploit the 'partial DCT information' technique to accelerate the processing. This technique only uses the low DCT frequencies which carry more important information.

Chang has implemented video manipulation functions such as overlapping, translation, scaling, and linear filtering in the compressed domain [3–6]. Merhav, Bhaskaran and Natarajan *et al.* have done similar work [9, 10, 12]. Shen has proposed an algorithm for direct extraction of features from compressed images [13, 14]. Although the costly overhead of decompressing is avoided in those algorithms, the computation to reconstruct blocks with motion vectors by using motion compensation is still expensive in the DCT domain.

Many fast algorithms using partial DCT coefficients have been proposed for video retrieval and scene detection. Yeo's DC + 2AC algorithm uses three DCT coefficients to detect scene changes [15–17]. Merhav and Bhaskaran have a similar algorithm called 3-2-1 [11]. Dimitrova's DC + M algorithm and Kobla's work exploit the DC coefficients and motion vectors as features for video retrieval and indexing [7, 8]. An advantage of their algorithms is that they are fast due to the fact that very few DCT coefficients are used. The disadvantage is that the number of DCT coefficients is fixed and the information, in most cases, is lost too much.

In our Computational Intelligence Laboratory, we have a research project on networked video databases. To find good compact representations for long video sequences, Joshi and Auephanwiriyakul use fuzzy clustering to extract representative frames (Rframes) [18, 19]. The fuzzy clustering algorithm uses luminance features of video frames to measure the distance and determine the centers of clusters. This algorithm can handle gradual scene changes which may cause the majority of scene change detection algorithms to fail. We are interested in hierarchical clusters and the use of the features in the compressed domain. Under this circumstance, we devise a fast algorithm with ALPF to reconstruct the motion-compensated blocks in the DCT domain and provide the DCT coefficients of video frames for the fuzzy clustering algorithm. In our ALPF algorithm, by controlling the bandwidths of the low-pass filters, we can provide a coarse-to-fine, multi-scale searching strategy for building hierarchical clusters.

3. Reconstructing Motion-Compensated Blocks

A block P with motion vector consists of two parts: a reference block P_{ref} and an error block P_{err}. We have $P = P_{ref} + P_{err}$, where P_{err} only contains the difference between P and P_{ref} and is encoded independently. In order to obtain P, we need to reconstruct P_{ref} from the reference frames.

The motion vector of a macroblock indicates the location of reference area in the four reference macroblocks. The same motion vector is used for every pixel in the macroblock. Thus, each 8×8 block P in the macroblock has a corresponding reference 8×8 block P_{ref} located in the four neighboring 8×8 blocks P_1, P_2, P_3, and P_4, as shown in

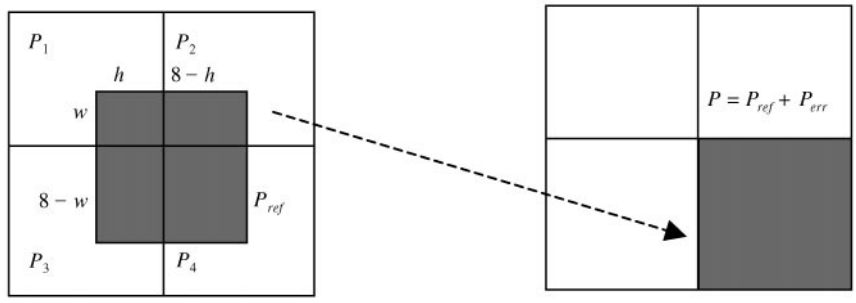


Figure 1. Reconstruction of a motion-compensated block

Figure 1. The partial areas (subblocks) of blocks P_1, P_2, P_3, and P_4 form the reference block P_{ref}.

In the spatial domain, an 8×8 block P can be represented as

$$P = \begin{pmatrix} p(0,0) & \dots & p(0,7) \\ & \ddots & \\ p(7,0) & \dots & p(7,7) \end{pmatrix}$$

where $p(i, j)$ denotes the pixel located at the ith row and jth column in block P.

Suppose C is an 8×8 matrix with elements defined by $c(i, j) = (k(i)/2)\cos((2j + 1)i\pi/16)$, $i, j = 0, \dots, 7$, with $k(i) = 1$ except $k(0) = 1/\sqrt{2}$. The forward DCT and inverse DCT can be represented as

$$DCT(P) = C \cdot P \cdot C^{\mathrm{t}}$$

$$P = C^{-1} \cdot DCT(P) \cdot C^{-\mathrm{t}}$$

where C^{t} and C^{-1} denote the transposition matrix and inverse matrix of C, respectively.

As we know, a reference block P_{ref} is the sum of the partial areas extracted from the four original neighboring blocks from which P_{ref} is derived, as shown in Figure 1. Chang uses matrix multiplication to implement the subblock extraction [3]. His approach can be described as

$$P_{ref} = \sum_{i=1}^{4} L_i \cdot P_i \cdot R_i$$

where L_i and R_i are matrices in the format of

$$\begin{pmatrix} 0 & [I]_{n \times n} \\ 0 & 0 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & 0 \\ [I]_{n \times n} & 0 \end{pmatrix}.$$

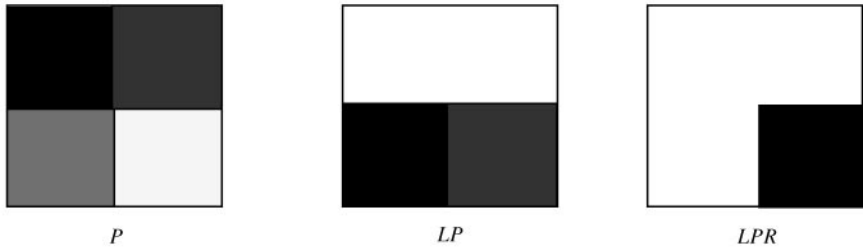


Figure 2. Using matrices L and R to extract and translate a subblock

The matrix $[I]_{n\times n}$ is the identity matrix of size $n \times n$, $1 \le n < 8$.

In the spatial domain, the multiplication P_i with matrix L_i or R_i extracts the first several rows or columns and translates them into the corresponding positions. An example is given in Figure 2.

In the frequency domain, the DCT coefficients of P_{ref} can be computed by

$$DCT(P_{ref}) = C\left(\sum_{i=1}^{4} L_i \cdot P_i \cdot R_i\right) C^{\mathrm{t}} = \sum_{i=1}^{4} C \cdot L_i \cdot P_i \cdot R \cdot C^{\mathrm{t}}$$

It is easy to verify that $C \cdot C^{\mathrm{t}} = C^{\mathrm{t}} \cdot C = I$. Thus, we have

$$\begin{aligned} DCT(P_{ref}) &= \sum_{i=1}^{4} (C \cdot L_i \cdot C^{\mathrm{t}})(C \cdot P_i \cdot C^{\mathrm{t}})(C \cdot R_i \cdot C^{\mathrm{t}}) \\ &= \sum_{i=1}^{4} DCT(L_i) \cdot DCT(P_i) \cdot DCT(R_i) \end{aligned}$$

The DCT coefficient matrices $DCT(L_i)$ and $DCT(R_i)$ can be pre-computed and stored in the memory.

4. Adaptive Low-Pass Filtering for DCT Domain Motion Compensation

Although the DCT coefficients of P_{ref} can be directly computed in the DCT domain using Chang's algorithm, the computation is expensive. Since $DCT(L_i)$, $DCT(P_i)$, and $DCT(R_i)$ are $N \times N$ matrix with size $N = 8$, it needs $2N^3 = 1024$ multiplication to calculate each $DCT(L_i P_i R_i)$, $i = 1, 2, 3, 4$, and $8N^3 = 4096$ multiplication for $DCT(P_{ref})$.

Usually, $DCT(P_i)$ is a sparse matrix and most of its elements for high frequencies are zero. Assume the sparse matrix $DCT(P_i)$ can be represented as

$$\begin{pmatrix} [A]_{m\times m} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[A]_{m\times m}$ denotes a $m\times m$ matrix with m^2 non-zero elements. The number of multiplication for computing $DCT(L_iP_iR_i)$ can be reduced from $2N^3$ to $Nm(m+N)$.

In this section, we develop an algorithm which further reduces the computation time by using adaptive low-pass filtering.

4.1. Spatial Continuity in Reference Block

In the spatial domain, $L_iP_iR_i$ represents an 8×8 block in which only the subblock extracted from P_i contains effective signals. We consider the extracted subblock as an effective area. The rest is blank. Since the signals change abruptly on the boundary between the effective and blank areas, there exists subblock boundary inside block $L_iP_iR_i$. Figure 3 shows the 1D case, where signals S_1 and S_2 change abruptly at t_0. Due to the subblock effect, almost all the DCT coefficients in matrix $DCT(L_iP_iR_i)$ are non-zero and many high-frequency components are caused by the abrupt signal changes in $L_iP_iR_i$.

We have observed that the effective areas in $L_iP_iR_i$, $i=1,\ldots,4$, are spatially neighboring and there should be no subblock boundaries inside P_{ref}. This implies that the signals change continuously across the subblock boundaries in P_{ref} and the number of non-zero elements in $DCT(P_{ref})$ is much smaller than that in $DCT(L_iP_iR_i)$, $i=1,\ldots,4$. As shown in Figure 3, the 1D signal S_1+S_2 changes smoothly at t_0.

We can divide each coefficient matrix $DCT(L_iP_iR_i)$ into two coefficient matrices: F_i and E_i. The matrix F_i is the actual contribution to P_{ref} and usually contains low-frequency components, while E_i is due to the subblock boundary effect. Thus, we have

$$DCT(P_{ref}) = \sum_{i=1}^{4}(F_i+E_i) = \sum_{i=1}^{4}F_i + \sum_{i=1}^{4}E_i$$

Although many elements in the matrices E_i, $i=1,\ldots,4$, are non-zero, the sum of the four matrices is an all-zero matrix, $\sum_{i=1}^{4}E_i = [0]_{8\times 8}$. This implies there is no need to calculate E_i, $i=1,\ldots,4$. Since most of the non-zero elements in E_i represent the high-frequency components due to the subblock boundary, we can apply a low-pass filter to $DCT(P_{ref})$ to avoid the unnecessary computation.

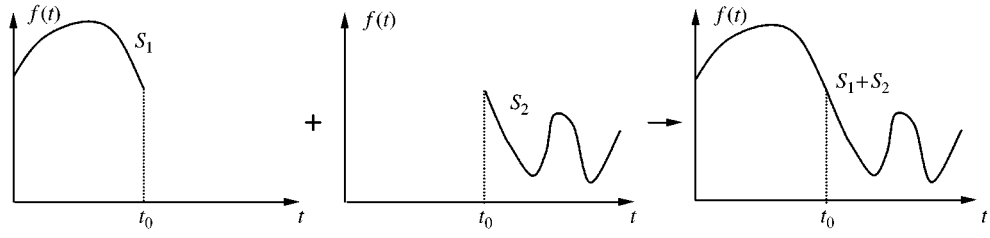


Figure 3. Reconstruction of a 1D signal

4.2. Low-Pass Filtering

The matrix $DCT(P_i) = [d(m, n)]_{8\times 8}$ is usually sparse. Assume $d(x_i, y_i)$ is the non-zero highest effective frequency element in $DCT(P_i)$ and $k_i = \max(x_i, y_i)$. That is, $d(m, n) = 0$, if $m, n \geq k_i$. The $DCT(P_i)$ can be rewritten as

$$DCT(P_i) = \begin{pmatrix} [A_i]_{k_i\times k_i} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[A_i]_{k_i\times k_i}$ is a $k_i \times k_i$ submatrix of $DCT(P_i)$ containing all non-zero elements in $DCT(P_i)$, $i = 1, \ldots, 4$.

Observing that all the signals in P_{ref} are obtained from P_i, $i = 1, \ldots, 4$, we have the assumption that the highest effective frequency component in $DCT(P_{ref}) = [d(m, n)]_{8\times 8}$ is $d(k_f, k_f)$, where $k_f = \max(k_1, k_2, k_3, k_4)$.

We can use a low-pass filter

$$T_{k_f} = \begin{pmatrix} [I]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[I]_{k_f\times k_f}$ is the identity matrix of size $k_f \times k_f$, to extract the effective coefficients in $DCT(P_{ref})$:

$$DCT(P_{ref}) = T_{k_f}\left(\sum_{i=1}^{4} DCT(L_i)\cdot DCT(P_i)\cdot DCT(R_i)\right)T_{k_f}$$

$$= \begin{pmatrix} [A]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}$$

Note that

$$T_{k_f}\cdot DCT(P_i)\cdot T_{k_f} = \begin{pmatrix} [I]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} [A_i]_{k_i\times k_i} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} [I]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} [A_i]_{k_i\times k_i} & 0 \\ 0 & 0 \end{pmatrix} = DCT(P_i)$$

where $k_i \leq k_f$. Thus we have

$$DCT(P_{ref}) = \sum_{i=1}^{4} (T_{k_f}\cdot DCT(L_i)\cdot T_{k_f})\, DCT(P_i)\, (T_{k_f}\cdot DCT(R_i)\cdot T_{k_f})$$

$$= \sum_{i=1}^{4} \begin{pmatrix} [B_i]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} [A_i]_{k_i\times k_i} & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} [C_i]_{k_f\times k_f} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[B_i]_{k_f \times k_f}$ and $[C_i]_{k_f \times k_f}$ are $k_f \times k_f$ top-left submatrices of matrices $DCT(L_i)$ and $DCT(R_i)$, respectively. That is,

$$DCT(L_i) = \begin{pmatrix} [B_i]_{k_f \times k_f} & \bullet \\ \bullet & \bullet \end{pmatrix} \quad \text{and} \quad DCT(R_i) = \begin{pmatrix} [C_i]_{k_f \times k_f} & \bullet \\ \bullet & \bullet \end{pmatrix}$$

The bandwidth of the low-pass filter is determined by k_f. For convenience of presentation, we call k_f the bandwidth of the low-pass filter. In our low-pass filtering algorithm, k_f is not fixed. For exact reconstruction, k_f is determined by k_i. If $DCT(P_i)$ contains more high-frequency components, a large k_f is used to retain more information in $DCT(P_{ref})$. On the other hand, if $DCT(P_i)$ contains less high-frequency components, a small k_f is automatically selected to speed up the computation.

4.3. Partial DCT Information Extraction

In video retrieval and video scene detection, many researchers exploit the 'partial DCT information' technique to accelerate the processing [7, 8, 11, 15–17]. A disadvantage of their work is that the number of used DCT coefficients is fixed. If the number is too small, the information is lost too much. On the other hand, if the number is too large, the computation is too expensive.

Considering the trade-off between speed and accuracy, we can choose different number of DCT coefficients by applying the adaptive low-pass filter. This approach is useful in the multi-scale search strategy.

We define k_b as the maximum bandwidth of the low-pass filter. Only the elements $d(m, n)$, $m, n \leq k_b$, in $DCT(P_i)$ are used for partial DCT information extraction. That is

$$\begin{aligned} DCT(P_i)_{partial} &= \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} DCT(P_i) \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [A_i]_{k_i \times k_i} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \end{aligned}$$

where $[I]_{k_b \times k_b}$ is the identity matrix of size $k_b \times k_b$.

Given $b_i = \min(k_i, k_b)$, $i = 1, \ldots, 4$, we have

$$DCT(P_i)_{partial} = \begin{pmatrix} [A_i]_{b_i \times b_i} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[A_i]_{b_i \times b_i}$ is a $b_i \times b_i$ submatrices extracted from matrices $DCT(P_i)$. They are related as follows:

$$DCT(P_i) = \begin{pmatrix} [A_i]_{b_i \times b_i} & \bullet \\ \bullet & \bullet \end{pmatrix}$$

We can approximate $DCT(P_{ref})$ by

$$DCT(P_{ref}) = \sum_{i=1}^{4} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [B_i]_{k_f \times k_f} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [A_i]_{b_i \times b_i} & 0 \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} [C_i]_{k_f \times k_f} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix}$$

Note that

$$\begin{pmatrix} [A_i]_{b_i \times b_i} & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [A_i]_{b_i \times b_i} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix}$$

where $b_i \le k_b$. Defining $k_n = \min(k_f, k_b)$, we have

$$\begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [B_i]_{k_f \times k_f} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} [B_i]_{k_n \times k_n} & 0 \\ 0 & 0 \end{pmatrix}$$

and

$$\begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [C_i]_{k_f \times k_f} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [I]_{k_b \times k_b} & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} [C_i]_{k_n \times k_n} & 0 \\ 0 & 0 \end{pmatrix}$$

where $[B_i]_{k_n \times k_n}$ and $[C_i]_{k_n \times k_n}$ are $k_n \times k_n$ top-left submatrices of matrices $DCT(L_i)$ and $DCT(R_i)$, respectively. That is,

$$DCT(L_i) = \begin{pmatrix} [B_i]_{k_n \times k_n} & \bullet \\ \bullet & \bullet \end{pmatrix} \quad \text{and} \quad DCT(R_i) = \begin{pmatrix} [C_i]_{k_n \times k_n} & \bullet \\ \bullet & \bullet \end{pmatrix}$$

Finally, we approximate $DCT(P_{ref})$ as follows:

$$DCT(P_{ref}) = \sum_{i=1}^{4} \begin{pmatrix} [B]_{k_n \times k_n} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [A_i]_{b_i \times b_i} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} [C_i]_{k_n \times k_n} & 0 \\ 0 & 0 \end{pmatrix}$$

4.4. Computational Complexity

We use the number of multiplication to estimate the computation time. In the case of exact reconstruction, with low-pass filtering, the number of multiplication for computing $DCT(P_{ref})$ is

$$\sum_{i=1}^{4} k_f k_i (k_f + k_i)$$

The speedup S is given by

$$S = \frac{\sum_{i=1}^{4} Nk_i(N + k_i)}{\sum_{i=1}^{4} k_f k_i(k_f + k_i)}$$

Assuming $k_1 = 2$, $k_2 = k_3 = 3$, $k_4 = 4$, and $k_f = \max(k_1, k_2, k_3, k_4) = 4$, we have

$$S = \frac{160 + 264 + 264 + 384}{48 + 84 + 84 + 128} = \frac{1072}{344} = 3{\cdot}12$$

In the case of approximate reconstruction, if k_b is the maximum bandwidth of the low-pass filter, the number of multiplication for computing $DCT(P_{ref})$ is

$$\sum_{i=1}^{4} k_n b_i(k_n + b_i)$$

where $b_i = \min(k_i, k_b)$, $i = 1, \ldots, 4$, and $k_n = \min(k_f, k_b)$. The speedup S is given by

$$S = \frac{\sum_{i=1}^{4} Nb_i(N + b_i)}{\sum_{i=1}^{4} k_n b_i(k_n + b_i)}$$

Assume $k_1 = 2$, $k_2 = k_3 = 3$, $k_4 = 4$, and $k_f = \max(k_1, k_2, k_3, k_4) = 4$, and $k_b = 3$. The speedup is

$$S = \frac{160 + 264 + 264 + 264}{30 + 54 + 54 + 54} = \frac{952}{192} = 4{\cdot}96$$

5. Experimental Results

We use Chang's algorithm as the original algorithm and compare our algorithm with adaptive low-pass filtering (ALPF algorithm) with the original algorithm. All the testing programs are written in C and run on a SUN SPARCstation LX.

Two data sets are used in our experiments. In the first data set, we have five 256×256 gray-level images: 'Lena,' 'Boat,' 'Earth,' 'Peppers,' and 'Zelda,' shown in the first row of Figure 4. In the experiment, each image is divided into $32 \times 32 = 1024$ blocks of size 8×8. After DCT, quantization, and dequantization, we obtain the integral DCT coefficients for each block. Every four neighboring blocks are selected as a basic unit for reconstruction. We give the same motion vector to all 31×31 units and reconstruct 31×31 motion compensated blocks. A 248×248 reconstructed gray-level image is obtained by applying the IDCT to the 31×31 reconstructed DCT coefficient blocks. By comparing 248×248 reconstructed image to the corresponding area in the original 256×256 image, we evaluate the accuracy level of the two algorithms.

Table 1. Test results for different input images by using $8 \times 8 = 64$ DCT coefficients

	Original algorithm		ALPF algorithm		
Image	Running time (s)	SNR	Running time (s)	SNR	Speedup
Lena	0·933	29·59	0·566	29·65	1·65
Boat	0·933	28·39	0·699	28·29	1·33
Earth	1·083	27·50	0·716	27·55	1·51
Peppers	0·866	30·48	0·533	30·48	1·62
Zelda	0·849	32·96	0·399	33·06	2·13

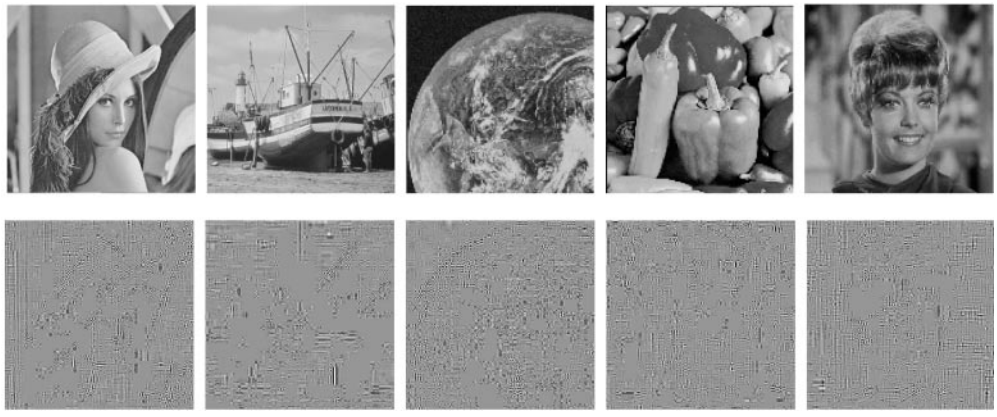


Figure 4. Test images (top row) and the differences of images reconstructed by the two algorithms (bottom row)

The second data set consists of six MPEG sequences: 'EricClapton,' 'DebbieHarry,' 'Jackson,' 'Energizer,' 'RedsNightmare', and 'UnderSiege.' These video sequences are extracted from some music show, commercial advertisement, graphic animation, and movie videos. We use these real video sequences to evaluate the running time for exact reconstruction implemented by the two algorithms. In Figures 7–9, we show the frames reconstructed from the MPEG videos.

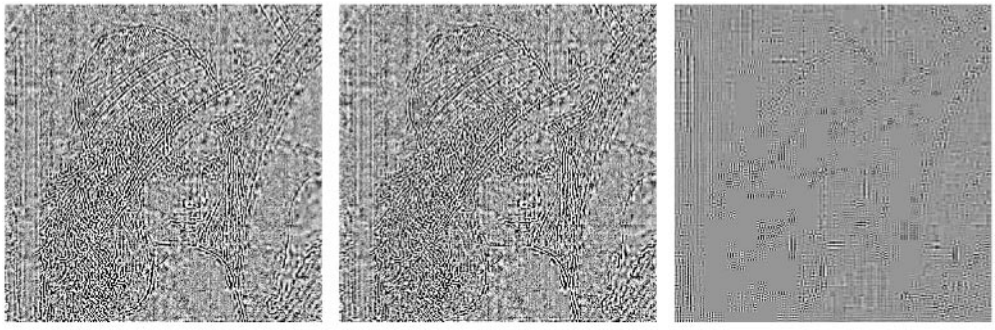
Using the first data set, we also compare the performance of the ALPF, DC + 2AC, and 3-2-1 algorithms for approximate extraction. The results are listed in Table 2.

5.1. Test Results for Grey-Level Images

Table 1 gives the performance of the original algorithm and the ALPF algorithm for exact reconstruction. The speedup is an average value for reconstructing 31×31 blocks in each image. The signal-to-noise ratio (SNR) reflects the distortion level between the reconstructed image and original image. Since the adaptive low-pass filter can reduce the blocky artifacts caused by quantization, three of five images have better SNR by using the ALPF algorithm.

Table 2. Approximate extraction by different algorithms

	DC + 2AC		ALPF ($k_b = 2$)		3-2-1		ALPF ($k_b = 3$)	
Image	Time (s)	SNR	Time (s)	SNR	Time (s)	SNR	Time (s)	SNR
Lena	0·063	21·86	0·075	23·09	0·144	24·37	0·155	25·14
Boat	0·062	20·77	0·073	21·38	0·131	22·37	0·146	22·78
Earth	0·066	20·08	0·076	21·12	0·150	22·19	0·168	22·91
Peppers	0·062	21·66	0·076	22·99	0·143	24·70	0·158	25·64
Zelda	0·061	25·90	0·074	27·15	0·136	28·54	0·154	29·21



(a) (b) (c)

Figure 5. Error images for reconstructing 'Lena' by the original algorithm (a) and the ALPF algorithm (b) and their difference image (c)

The error image is the difference between the original image and the corresponding reconstructed image, i.e., $I_{err} = I_{orig} - I_{recon} + 128$, and all the error images are enhanced to give better visual effect. Figure 5(a) is the error image between the original image 'Lena' and the image reconstructed by the original algorithm. Figure 5(b) gives the error image between the original image and the image reconstructed by the ALPF algorithm. Figure 5(c) shows the difference image, $I_{diff} = I_{recon1} - I_{recon2} + 128$, between the images reconstructed by the original and ALPF algorithms.

Note that in Figure 5(c) there are a few 'smooth' blocks in which all pixels have the same value 128. This implies that in these areas the original algorithm and the ALPF algorithm obtain the same DCT coefficients. Since the bandwidth of the low-pass filter is adaptive, in the area with more high-frequency components a larger bandwidth is automatically selected to retain all the important information. The 'smooth' blocks in the error image contain abruptly changing signals in the original image. Thus, all the DCT coefficients are reconstructed in these blocks. The same results are illustrated in the second row of Figure 4 for the five 248×248 error images from the original algorithm and the ALPF algorithm.

For approximate extraction, we compare our ALPF algorithm with Yeo's DC + 2AC algorithm [15] and Merhav's 3-2-1 algorithm [11]. There are three DCT coefficients used in the DC + 2AC algorithm and six coefficients used in the 3-2-1 algorithm. In our

Table 3. Approximate extraction for 'Lena'

Maximum bandwidth k_b	Original algorithm		ALPF algorithm		
	Running time (s)	SNR	Running time (s)	SNR	Speedup
8	0·933	29·59	0·566	29·65	1·65
7	0·916	29·45	0·499	29·48	1·84
6	0·899	29·26	0·433	29·27	2·08
5	0·883	28·78	0·383	28·60	2·31
4	0·816	27·69	0·266	27·38	3·07
3	0·718	25·89	0·155	25·14	4·63
2	0·568	23·43	0·075	23·09	7·57
1	0·416	19·73	0·033	18·71	12·61

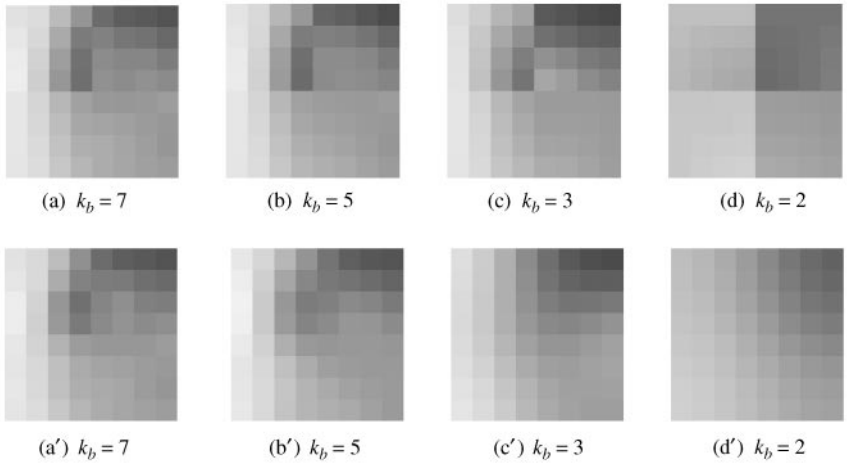


Figure 6. A block reconstructed using approximate extraction by the original algorithm (top row) and the ALPF algorithm (bottom row)

ALPF algorithm, we use $2^2 = 4$ and $3^2 = 9$ DCT coefficients when choosing $k_b = 2$ and $k_b = 3$, respectively. In Table 2, from left to right, both the running time and SNR increase as the number of used coefficients increases. This implies that there is trade-off between the speedup and reconstructed image quality.

The main drawback of the DC + 2AC and 3-2-1 algorithms is that the number of used coefficients is fixed. In our ALPF, the number of coefficients can change from $8^2 = 64$ to $1^2 = 1$ by selecting different k_b's. Table 3 gives the running time for different k_b's. As k_b decreases from 8 to 1, the speedup increases from 1·65 to 12·6.

Figure 6 shows one block reconstructed with different maximum bandwidth k_b. The images in the top row are results of the original algorithm. We can see the blocky effects inside the block when k_b becomes small. The bottom row shows the images reconstructed by the ALPF algorithm. The low-pass filter eliminates the blocky effects, and the images look smoother.

Table 4. MPEG videos used in experiment

Video sequence	Frame size	Number of frames	Percentage of *P*, *B* Frames (%)	Number of macroblocks	Percentage of MC Blocks (%)
DebbieHarry	160×128	100	83·0	8000	66·7
EricClapton	160×128	151	82·7	12080	82·1
Jackson	160×128	155	82·5	12400	80·1
Energizer	192×144	488	83·2	52704	80·0
RedNightmare	320×240	1210	96·6	363000	92·0
UnderSiege	352×240	731	83·2	241230	80·7

5.2. Test Results for MPEG Video Sequences

In an MPEG video, the macroblocks in the *I* frames are intracoded and their DCT coefficients can be obtained directly from the data stream. Thus, the total running time for reconstructing the motion-compensated (MC) blocks is determined by the number of the macroblocks with motion vectors in the *P* and *B* frames. Generally, the running time can be roughly estimated by the frame size, number of frames, and percentage of *P*, *B* frames, which are listed in Table 4. Since some macroblocks in the *P* and *B* frames do not have motion vectors and are intracoded independently, percentage of the motion-compensated macroblocks is generally smaller than the percentage of *P*, *B* frames.

Reconstruction of a macroblock is completed by reconstructing four 8×8 blocks in the luminance plane, one 8×8 block in the Cr chrominance plane, and one 8×8 block in the Cb chrominance plane. The basic unit for reconstruction is an 8×8 DCT coefficient block. Given the motion vector of a macroblock, we can easily locate the reference blocks for each 8×8 block inside the macroblock.

We use the number of multiplication for reconstruction to approximate the running time of the algorithms. For our ALPF algorithm, the running time is not only related to the total number of the motion-compensated macroblocks but also related to the frequency distribution inside the reference blocks. In Section 4.4, we mention that the number of multiplication for reconstructing one 8×8 block is given by $\sum_{i=1}^{4} k_f k_i (k_f + k_i)$, where $k_f = \max_i(k_i)$ and k_i is the bound of non-zero elements in one block. We call k_i effective block size in our ALPF algorithm (Figures 7, 8).

In Table 5, we give the average effective block size, number of multiplication, and the speedup. On average, the ALPF algorithm is two times faster than the original algorithm. Note that the speedup is related to the average effective block size. Lower average effective block size results in higher speedup. In Figure 9, we see that the scenes

Figure 7. Frames reconstructed from music show 'DebbieHarry' video sequence

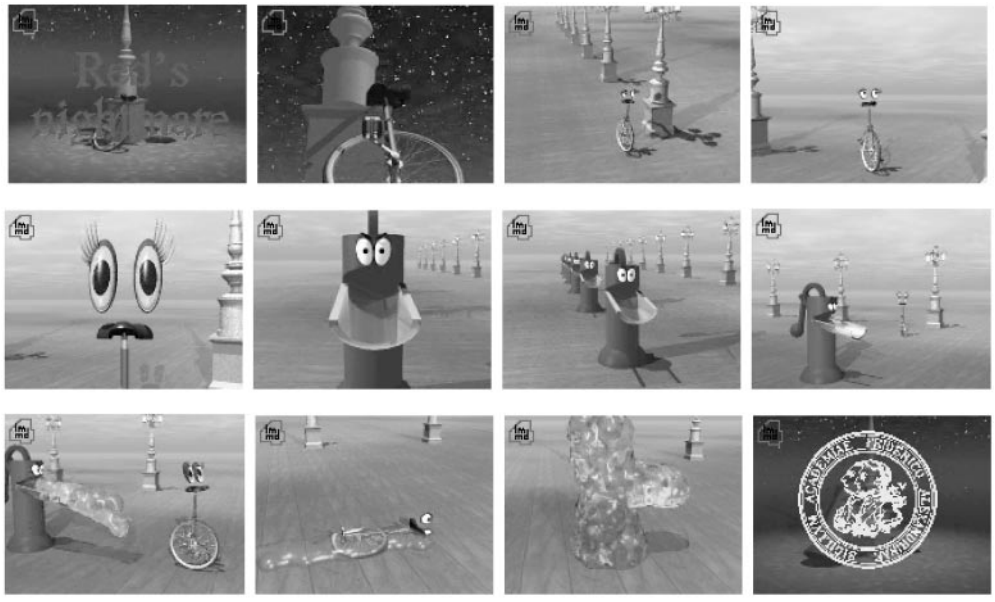


Figure 8. Frames reconstructed from graphic animation 'RedsNightmare' video sequence

Table 5. Test results for different MPEG video sequences

Video sequence	Average effective block size	Numbers of multiplication ($\times 10^6$)		Speedup
		Original algorithm	ALPF algorithm	
DebbieHarry	6·35	157·59	111·04	1·42
EricClapton	5.09	221.06	103.64	2.13
Jackson	5·11	199·63	102·04	1·96
Energizer	6·19	935·03	594·65	1·57
RedNightmare	5·10	6521·71	3623·24	1·81
UnderSiege	3·39	4548·09	980·17	4·64

in the movie 'UnderSiege' change smoothly and the signal concentrates on low frequencies, which results in a small average effective block size, 3·39. Using the ALPF algorithm, we can avoid more unnecessary computation for high-frequency coefficients for the 'UnderSiege' video sequence than that for others, achieving the highest speedup 4·64.

We have colleagues observe the video frames reconstructed by the two algorithms and judge the quality of the reconstructed frames. The observers cannot tell the difference between the normal size frames reconstructed by the two algorithms. Provided by the enlarged frames, they note that the frames reconstructed by the ALPF algorithm not only retains all the details in the areas with more high-frequency components, but also gives smoother and better look in the areas with less high-frequency components.

Figure 9. Frames reconstructed from movie 'UnderSiege' video sequence

6. Conclusion

In this paper, we have presented an algorithm with adaptive low-pass filtering to reconstruct blocks with motion vectors in the compressed domain. The experiments show that this algorithm is faster by avoiding the unnecessary computation and improve the quality of reconstructed blocks by reducing blocky artifacts due to quantization. Our ALPF algorithm can be applied in various MPEG video-processing applications such as key-frame extraction, content-based retrieval, multi-scale searching, and scene detection. Currently, our ALPF algorithm only deals with the blocks with motion vectors up to one pixel accuracy. In the future work, we attempt to extend our algorithm to reconstruction of blocks with half-pixel accuracy motion vectors.

Acknowledgements

This work was in part supported by the University of Missouri Research Board under grant RB-98-068. The authors wish to thank the four anonymous reviewers and the guest editors for their insightful comments and suggestions.

References

1. J. L. Mitchel (1997) *MPEG Video Compression Standard.* Chapman & Hall, New York.
2. K. R. Rao & P. Yip (1990) *Discrete Cosine Transform: Algorithms, Advantages, and Applications.* Academic Press, San Diego, CA.
3. S. F. Chang (1993) Compositing and manipulation of video signals for multimedia network video services. Ph.D. Dissertation, University of California, Berkeley.
4. S. F. Chang (1995) Compressed-domain techniques for image/video indexing and manipulation. In: *Proceedings of the IEEE International Conference on Image Processing* (*ICIP'95*), *Special Session on Digital Image/Video Libraries and Video-on-Demand*, Washington, DC.
5. S. F. Chang & D. G. Messerschmitt (1993) A new approach to decoding and compositing motion compensated DCT based images. In: *Proceedings of ICASSP '93*, Minneapolis, MN, pp. 421–424.
6. S. F. Chang & D. G. Messerschmitt (1995) Manipulation and compositing of MC-DCT compressed video. *IEEE JSAC Special Issue on Intelligent Signal Processing*, 13, 1–11.
7. N. Dimitrova & M. Abdel-Mottaleb (1997) Content-based video retrieval by example video clip. In: *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Database*, Vol. 3022, pp. 59–69.
8. V. Kobla & D. Doermann (1997) Compressed domain video indexing techniques using DCT and motion vector information in MPEG video. In: *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Database*, Vol. 3022, pp. 200–210.
9. N. Merhav & V. Bhaskaran (1996) A fast algorithm for DCT domain filtering. Hewlett–Packard Laboratories Technical Report, HPL-95-56, May.
10. N. Merhav & V. Bhaskaran (1996) A fast algorithm for DCT-domain image downsampling. In: *Proceedings of ICASSP'96*, Atlanta, GA.
11. N. Merhav & V. Bhaskaran (1996) A fast algorithm for DCT-domain inverse motion compensation. In: *Proceedings of ICASSP'96*, Atlanta, GA, pp. 2307–2310.
12. B. K. Natarajan & V. Bhaskaran (1995) A fast approximate algorithm for scaling down digital images in the DCT domain. In: *Proceedings of the IEEE International Conference on Image Processing* (*ICIP'95*), Washington, DC, October.
13. B. Shen & I. K. Sethi (1996) Convolution-based edge detection for image/video in block-DCT domain. *Journal of Visual Communications and Image Representation* 7, 411–423.
14. B. Shen & I. K. Sethi (1996) Direct feature extraction from compressed image. In: *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Database IV*, Vol. 2670.
15. B. L. Yeo (1996) Efficient processing of compressed images and video. Ph.D. Dissertation, Princeton University.
16. B. L. Yeo & B. Liu (1995) On the extraction of DC sequences from MPEG compressed video. In: *Proceedings of the IEEE International Conference on Image Processing*, Vol. 2, pp. 260–263.
17. B. L. Yeo & B. Liu (1995) Rapid scene analysis on compressed videos. *IEEE Transactions on Circuits and Systems For Video Technology*, December.
18. S. Auephanwiriyakul & R. Krishnapuram (1998) Fuzzy shot clustering to support networked video databases. In: *Proceedings of the IEEE World Congress on Computational Intelligence FUZZ-IEEE'98*, Anchorage, Alaska, pp. 1400–1405.
19. A. Joshi, S. Auephanwiriyakul & R. Krishnapuram (1998) On fuzzy clustering and content based access to networked video databases. In: *Proceedings of the 8th IEEE Workshop on Research Issue in Data Engineering*, February, pp. 42–49.