Multimedia Systems 6: 113–124 (1998)

Multimedia Systems

Block-based manipulations on transform-compressed images and videos

Bo Shen, Ishwar K. Sethi

Vision and Neural Networks Laboratory, Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
e-mail: {bos, sethi}@cs.wayne.edu

Abstract. This paper addresses the problem of direct image and video manipulation in compressed domain. The capability to perform such manipulations has become attractive in recent years, as more and more visual information is being captured, stored, and moved in compressed form. Our solution to direct manipulation of compressed images and video is based on a set of block-level transforms, called the *inner block transforms* (IBTs). Each IBT requires a minimal computation effort in compressed domain to yield a regular geometric transformation on an image block. The paper shows how the IBTs can be used to perform many image and video manipulations. This is done by describing the application of IBTs to image subtitling, shearing, and arbitrary angle rotation. The paper also discusses the application of IBT to M-JPEG and MPEG video. The benefits of the proposed direct compressed domain manipulation approach include faster speed, preservation of image quality, and less computing resources.

Key words: Geometric transform – JPEG – M-JPEG – MPEG – Compressed domain – Image/video manipulation – Multimedia – Visual information system

1 Introduction

The last few years have seen a remarkable growth of information in digital form due to a combination of several technological factors that include the availability of high-speed electronic networks and affordable, yet powerful desktop computing. This, coupled with advances in scientific and consumer imaging technology (digital radiography, digital still cameras, desktop scanners, video capture boards) has created an environment where a large fraction of digital information is being created and disseminated in visual form (graphics, images, and video). This trend towards information in visual form is not surprising, considering that we use vision more than any other sense to acquire and communicate information.

Correspondence to: B. Shen

One of the salient characteristics of visual information is its sheer volume. For example, a color image of size 640× 480 pixels is made up of almost one 1,000,000 bytes. Such a sheer volume of data imposes severe requirements with respect to storage, manipulation, and delivery of visual information. As a consequence, it is becoming common to capture, move, and store visual information in compressed form. Since compressed images offer lower data rate, there is a growing interest in developing image-processing methods that operate directly on compressed data for common image manipulation and composition tasks such as translation, scaling, overlaying (opaque or semi-transparent) rotation, and image filtering. The motivation behind such methods is that the lower data rate of compressed images coupled with the elimination of decompression- compression cycle can be used to perform many image manipulation tasks in "real" time, thus providing a user with enhanced capabilities without expensive real-time image-processing hardware.

The methods for processing compressed images must be based on the underlying compression scheme. Since the Discrete Cosine Transform (DCT) is the basis of spatial redundancy removal in current commercial image compression standards such as JPEG, MPEG, and H.261, methods for processing of transformed-compressed images have been the focus of attention. Currently, there exist methods for some basic image manipulation operations in the DCT domain that show greater efficiency than their spatial domain counterparts. However, for compressed-image processing to be really useful, there is a need for methods that can perform much more sophisticated image manipulation, as well as image feature extraction. The present paper is an attempt to fill this need, where we describe a new set of methods, called inner block transformation (IBT) methods, and show how to perform geometric manipulations, for example, an image rotation by an arbitrary angle, directly on images in the transform-compressed domain. We also show the applications of IBT for manipulation of M-JPEG and MPEG videos directly in compressed domain.

The organization of the paper is as follows. In Sect. 2, the JPEG compression model is briefly introduced. Section 3 presents a brief review of existing compressed domain manipulation methods, and casts them into two categories: the

inner block algebra (IBA) approach and the inner block rearrangement/resampling (IBR) approach. Our solution to direct manipulation of compressed images and videos (the IBTs) is presented in Sect. 4. Section 5 discusses the application of IBTs to image subtitling, shearing, and arbitrary angle rotation. This is followed by the use of IBTs for M-JPEG and MPEG manipulation in Sect. 6. Finally, a summary of the work is provided in Sect. 7.

2 JPEG model of compression

In this section, we provide a brief review of the JPEG compression model, which is the most commonly used image compression model in multimedia systems research and applications. JPEG uses a lossy compression scheme based on an orthogonal transform, the DCT, to remove spatial redundancy in still images. Most of the current compressed-domain manipulation methods are based on the JPEG compression model and take advantages of the DCT properties.

The JPEG compression process is shown in Fig. 1. It consists of five steps. In the first step, the input image is subdivided into 8×8 blocks and the 2D forward DCT of each block is computed. The following equations are the mathematical definition of the DCT transform pair [8×8 forward DCT (FDCT) and 8×8 inverse DCT (IDCT)]:

$$F(u,v) = \frac{C_u C_v}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} f(i,j) \,, \tag{1}$$

$$f(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C_u C_v}{4} \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} F(u,v) \,, \tag{2}$$

where

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u, v = 0, \\ 1 & \text{otherwise.} \end{cases}$$

In the next step, each of the 64 DCT coefficients, $F(u,v)$, is uniformly quantized in conjunction with a 64-element quantization table, $Q_F(u,v)$, to discard information which is not visually significant. The output of the quantization, $F'(u,v)$, is thus computed as:

$$F'(u,v) = roundoff\left(\frac{F(u,v)}{Q_f(u,v)}\right) \,. \tag{3}$$

After quantization, all of the coefficients are ordered into the "zig-zag" sequence. This ordering helps to facilitate entropy coding by placing low-frequency coefficients (which are more likely to be nonzero) before the high-frequency coefficients. In entropy coding, run length coding (RLC), and Huffman coding [8] (or arithmetic coding) are performed to further compact the information, and finally generate compressed data stream. To uncompress or reconstruct an image from its compressed data stream, similar steps are applied in a reverse order, as shown in Fig. 1.

The encoded block after run length coding or the decoded block after Huffman decoding is called RLC block. The compressed-domain manipulation methods operate on RLC blocks. This is for two reasons. First, operating at RLC block level makes it easy to keep track of the structure of the original image. Second, operating at RLC block level needs only minimal decoding of the compressed data stream.

3 Existing compressed-domain manipulation methods

In this section, we provide a brief review of the existing compressed-domain manipulation methods. Although, as stated earlier, the RLC block level is the preferred place for manipulating compressed image data, we assume, for sake of a clear presentation, all the manipulation is done at the quantized block level. This allows us to view each block as an 8×8 matrix and treat different manipulations as matrix operations. Since the processes after quantization (zig-zagging and run length coding), are well-defined processes [16], one can always use additional control flow to implement them at the RLC block level. The following table shows the notations used in this paper.

$[F]$, $[G]$ $F(u,v)$, $G(u,v)$	(upper case) compressed blocks, in compressed domain
$[f]$, $[g]$ $f(i,j)$, $g(i,j)$	(lower case) uncompressed blocks, in spatial domain
Q_F	quantization table for image F
[ ]+#	add every element in [ ] by #
#[ ]	multiply every element in [ ] by #
[ ] + [ ]	add corresponding elements of two blocks
[ ] • [ ]	multiply corresponding elements of two blocks
[ ] ⊗ [ ]	convolve two blocks
[ ] [ ]	matrix multiplication
$A \Leftrightarrow B$	(spatial-domain) operation A is corresponding to (compressed-domain) operation B

3.1 Inner-block algebra (IBA)

The earliest work on direct manipulation of compressed image and video data expectedly dealt with point processing, consisting of operations such as contrast manipulation and image subtraction, where a pixel value in the output image at position p depends solely on the pixel value at the same position p in the input image. Examples of such works can be found in Chang and Messerschmitt [4], who developed some special functions for video compositing, and in Smith and Rowe [13], who developed a set of algorithms for basic point operations. When viewing the compressed-domain manipulation as a matrix operation, we can characterize the point-processing operations on compressed images and videos as IBA operations, since the information in the output block, i.e., the manipulated block, comes solely from information in the corresponding input block. We list these operations in Table 1.

Because of the large number of zeros in the quantized block, the data manipulation rate is heavily reduced. The speedup for operations IBA1 to IBA3 is quite obvious. Convolution in compressed domain was derived in [13] by mathematically combining decompression, manipulation, and recompression processes to obtain a single equivalent local linear operation, where we can easily take the advantage of the energy compaction property in quantized DCT blocks. A similar approach was taken by Smith [14] to extend point

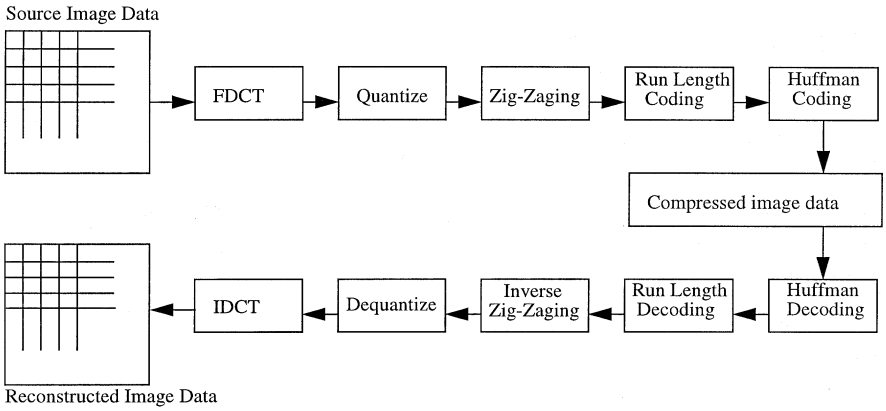


Fig. 1. JPEG compression model

Table 1. Inner-block algebra operations

	Spatial operation	Math. expression	DCT coefficients manipulation
IBA1	Scalar addition	$[f]+\beta \Leftrightarrow [F]+\begin{bmatrix} \frac{8\beta}{Q(0,0)} & 0 \\ 0 & 0 \end{bmatrix}$	only (0, 0) coefficient is affected
IBA2	Scalar multiplication	$\alpha[f] \Leftrightarrow \alpha[F]$	multiplies only on nonzero coefficients
IBA3	Pixel addition	$[f]+[g] \Leftrightarrow [F]+[G]$	additions only on nonzero coefficients
IBA4	Pixel multiplication	$[f] \bullet [g] \Leftrightarrow [F] \otimes [G]$	convolution in compressed domain is much more efficient, taking the advantage of the packing of data in quantized DCT blocks

processing to global processing of operations, where the value of a pixel in the output image is an arbitrary linear combination of pixels in the input image.

As an example of IBA application, consider the compositing operation where foreground F is combined with background B with a factor of α to generate an output R. In spatial domain, this operation can be expressed [18] as $R = \alpha \times F + (1-\alpha) \times B$. The operation in block DCT domain can be conveniently performed as $[R] = \alpha[F] + (1-\alpha)[B]$. The operation is just a combination of some of the above-defined image algebra operations; it can be done in DCT domain efficiently with tremendous speedup [13]. Similar compressed-domain algorithms for subtitling and dissolving applications can also be developed based on the above IBA operations with computational speedups of 50 or more over the corresponding processing of the uncompressed data [3, 13].

This set of methods can also be used for color transformation in compressed domain. As long as the transformation is linear, it can be derived in compressed domain using a combination of the IBA operations. As RGB images are transformed to the YUV domain first, then compressed using DCT, one can obtain RGB directly from DCT blocks in the YUV domain using IBAs.

3.2 Inner-block rearrangement/resampling (IBR)

Many image manipulation operations are local or neighborhood operations where the pixel value at position p in the output image depends on neighboring pixels of p in the input image. We characterize methods to perform such operations in the compressed domain as IBR methods. These methods are based on the fact that DCT is a unitary orthogonal transform and is distributive to matrix multiplication [10]. It is also distributive to matrix addition, which is actually

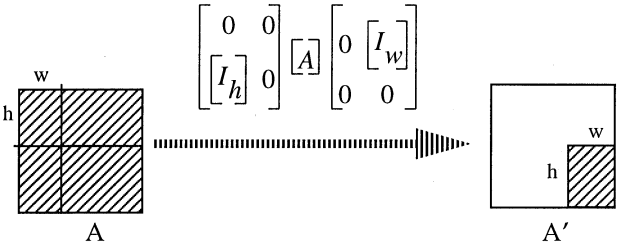


Fig. 2. Rearrangement of sub-block

the case of pixel addition (IBA3). We group these two distributive properties of JPEG/DCT in Table 2 and name these properties by the roles they play in image manipulation. We refer to the process of subblock extraction and shifting as rearrangement, and the process of filtering and sampling as resampling.

One purpose of matrix multiplication is to extract or rearrange subblock structure. This is shown in Fig. 2, where $[I_h]$ and $[I_w]$ are identity matrices of size h and w, respectively. As shown in the figure, we can extract the upper left subblock and move it to the bottom right corner by multiplying matrix A (source block) with certain pre- and post-matrices, called *manipulation matrices*. Using the distributive property, we can perform such operations directly in the DCT domain. This allows us to break the block structure, which is a must for any sophisticated manipulation.

We can use IBR1 to break the block structure and use IBR2 to construct a new block from several blocks. In this sense, it is an inter-block operation, but for each operation alone, either to extract/rearrange a subblock from the original block or to add subblock onto the result block, it is still an inner block operation. A combination of these inner block operations gives us the possibility for global manipulation of images in compressed domain.

Table 2. Inner-block resampling operations

	Spatial operation	Math. expression	DCT coefficients manipulation
IBR1	sub-block extraction, rearrangement or resampling	$[f][g] \Leftrightarrow [F][G]$	DCT domain matrix multiplication
IBR2	Reconstruction	$[f] + [g] \Leftrightarrow [F] + [G]$	additions only on nonzero coefficients

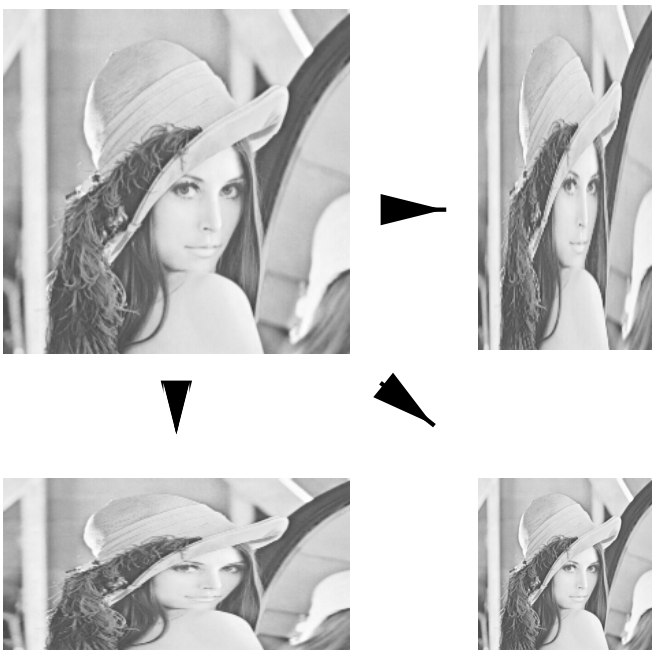


Fig. 3. Resampling in DCT domain

As an example of IBR operations, consider the shrink-by-2 operation. It is easy to see that this operation can be done in DCT domain using IBR1 operation, i.e., matrix multiplication with the help of the following two 8×8 matrices:

$$\begin{bmatrix} 0.5 & 0.5 & & & & & & 0 \\ & & 0.5 & 0.5 & & & & \\ & & & & 0.5 & 0.5 & & \\ & & & & & & 0.5 & 0.5 \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ 0 & & & & & & & \end{bmatrix}$$

$$\begin{bmatrix} 0.5 & & & & & & & 0 \\ 0.5 & & & & & & & \\ & 0.5 & & & & & & \\ & 0.5 & & & & & & \\ & & 0.5 & & & & & \\ & & 0.5 & & & & & \\ & & & 0.5 & & & & \\ 0 & & & 0.5 & & & & \end{bmatrix} .$$

Figure 3 shows some special effect manipulations of an image in compressed domain, which require resampling. Generally, a manipulation requiring uniform and integer scaling, i.e., space-invariant filtering, is easy to implement in DCT domain using the resampling matrix. Since each block can use the same resampling matrix in space-invariant filtering, these kinds of manipulations require little overhead in DCT domain. On the other hand, sophisticated manipulations such as projective mapping based on space-variant filtering are difficult to perform in DCT domain due to a large overhead needed for rearrangement and resampling of blocks.

Based on the above, Chang and Messerschmitt [3, 4] have developed a set of algorithms to manipulate images directly in the compressed domain. Some of the interesting algorithms developed by them include the translation of images by arbitrary amounts, linear filtering and scaling. Merhav and Bhaskaran [11] also found that this matrix-multiplication type of manipulation can be incorporated with many fast DCT (or IDCT) methods developed based on matrix decomposition [1, 5]. Tremendous speedup can be thus obtained, even without considering the sparseness of DCT block. Rotation and shearing can also be done in form of matrix multiplication. To perform arbitrary angle rotation, however, the extraction and rearrangement of subblocks requires many different manipulation matrices, which in DCT form are difficult to manage without the help of the following set of properties of JPEG/DCT discussed in the following section.

4 A new set of algorithms for geometric transformation

The IBA operations do not allow those manipulations where pixels are moved around. While IBR operations allow such manipulations, the cost is usually prohibitive due to expensive matrix multiplications. In this section, we present a set of block operations that allow moving of pixels directly in DCT domain in an efficient manner, involving only column/row exchanges or sign reversals of DCT coefficients. However, the movement of pixels is limited to a set of regular moves. Later on, we show how the proposed block operations can be used to perform a variety of sophisticated manipulations. In the following discussion, we assume the quantization table of the output image G (in DCT domain) is the same as that of the input image F (in DCT domain), that is:

$$Q_G(u, v) = Q_F(u, v) \; . \tag{4}$$

4.1 Regular geometric transformation

We define a *regular geometric transformation* as one of the following seven operations: column flip (U-Flip), row flip (V-Flip), rotate 180 (R-180), rotate 270 (R-270), rotate 90 (R-90), diagonal flip (D-Flip) and opposite diagonal flip (OD-Flip). We assume image size is always a multiple of the block size 8, so we can concentrate on operations within a block. The U-Flip refers to flipping an image in the center around its vertical axis. In spatial domain it can be mathematically expressed as

$$g(i,j) = f(7-i,j) \qquad i,j = 0,1,\ldots 7\ , \tag{5}$$

where $f(i,j)$ and $g(i,j)$, respectively, are input and output blocks. The D-Flip means flipping an image around its principle diagonal. In spatial domain it is given as

$$g(i,j) = f(j,i) \qquad i,j = 0,1,\ldots 7\ . \tag{6}$$

Other regular geometric transforms can be defined in a similar way. The important point of note is that pixels in the output block come from the same input block; only their positions are changed.

Now, we derive their compressed-domain counterparts of regular geometric transformations. For U-Flip, the input block is $F(u,v)$ and the output block is $G(u,v)$. From Eqs. 1, 3, 5, 4, and 2 the output block in compressed domain can be computed as follows:

$$\begin{aligned} G(u,v) &= \frac{C_u C_v}{4Q_G(u,v)} \sum_i \sum_j \cos\left(\frac{2i+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) g(i,j) \\ &= \frac{C_u C_v}{4Q_G(u,v)} \sum_i \sum_j \cos\left(\frac{2i+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) f(7-i,j) \end{aligned}$$

let $\quad i = 7 - i'$
because $\quad i = 0 \sim 7$, so $\quad i' = 7 \sim 0$

$$\begin{aligned} G(u,v) &= \frac{C_u C_v}{4Q_G(u,v)} \sum_{i'} \sum_j \cos\left(\frac{2(7-i'+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) f(i',j) \\ &= \frac{C_u C_v}{4Q_G(u,v)} \sum_{i'} \sum_j \cos\left(u\pi - \frac{2i'+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) f(i',j) \\ &= \cos(u\pi)\frac{C_u C_v}{4Q_G(u,v)} \sum_{i'} \sum_j \cos\left(\frac{2i'+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) f(i',j) \\ &= \cos(u\pi)F(u,v) \end{aligned}$$

So, in compressed domain, the output block can be directly computed from the input block as

$$G(u,v) = \cos(u\pi)F(u,v)\ . \tag{7}$$

Now, let us consider the case of D-Flip. In this case, using Eqs. 1, 3, 6, 4, and 2, the output block in compressed domain can be computed as follows:

$$\begin{aligned} G(u,v) &= \frac{C_u C_v}{4Q_G(u,v)} \sum_i \sum_j \cos\left(\frac{2i+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) g(i,j) \\ &= \frac{C_u C_v}{4Q_G(u,v)} \sum_i \sum_j \cos\left(\frac{2i+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2j+1}{16}v\pi\right) f(j,i) \end{aligned}$$

let $\quad i' = j, j' = i$

$$\begin{aligned} G(u,v) &= \frac{C_u C_v}{4Q_G(u,v)} \sum_{j'} \sum_{i'} \cos\left(\frac{2j'+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2i'+1}{16}v\pi\right) f(i',j') \\ &= \frac{Q_F(v,u)}{Q_G(u,v)} \frac{C_u C_v}{4Q_F(v,u)} \sum_{j'} \sum_{i'} \cos\left(\frac{2j'+1}{16}u\pi\right) \\ &\quad \cos\left(\frac{2i'+1}{16}v\pi\right) f(i',j') \\ &= \frac{Q_F(v,u)}{Q_G(u,v)} F(v,u) \end{aligned}$$

So, in compressed domain, the output block can be directly computed from the input block as

$$G(u,v) = \frac{Q_F(v,u)}{Q_G(u,v)} F(v,u)\ . \tag{8}$$

The above expression says that, if we want to perform D-flip in compressed domain, we should exchange the coefficients on each row with those in the corresponding column, and multiply each coefficient with $Q_F(v,u)/Q_G(u,v)$. Q_F and Q_G can be selected from one of the following tables, which were provided as examples in [12] for luminance and chrominance quantization tables:

16, 11, 10, 16, 24, 40, 51, 61	17, 18, 24, 47, 99, 99, 99, 99
12, 12, 14, 19, 26, 58, 60, 55	18, 21, 26, 66, 99, 99, 99, 99
14, 13, 16, 24, 40, 57, 69, 56	24, 26, 56, 99, 99, 99, 99, 99
14, 17, 22, 29, 81, 87, 80, 62	47, 66, 99, 99, 99, 99, 99, 99
18, 22, 37, 56, 68, 109, 103, 77	99, 99, 99, 99, 99, 99, 99, 99
24, 35, 55, 64, 81, 104, 113, 92	99, 99, 99, 99, 99, 99, 99, 99
49, 64, 78, 87, 103, 121, 120, 101	99, 99, 99, 99, 99, 99, 99, 99
72, 92, 95, 98, 112, 100, 103, 99	99, 99, 99, 99, 99, 99, 99, 99
Luminance quantization table	Chrominance quantization table

The chrominance quantization table is diagonally symmetric, i.e., $Q(u,v) = Q(v,u)$. Thus, using Eq. 4, we can simplify Eq. 8 to:

$$G(u,v) = F(v,u)\ . \tag{9}$$

In the luminance quantization table, although there are differences between elements in columns and those in corresponding rows, experiments show that the transform effects are almost the same by using Eq. 8 or Eq. 9.

4.2 Inner-block transform (IBT)

All regular geometric transformations can be derived in the way we presented above. In Table 3, we summarize the seven regular geometric transformations as IBTs. In Table 3, $\cos(n\pi)$ only has two values: 1 or -1. All operations required in compressed domain are, therefore, nothing but

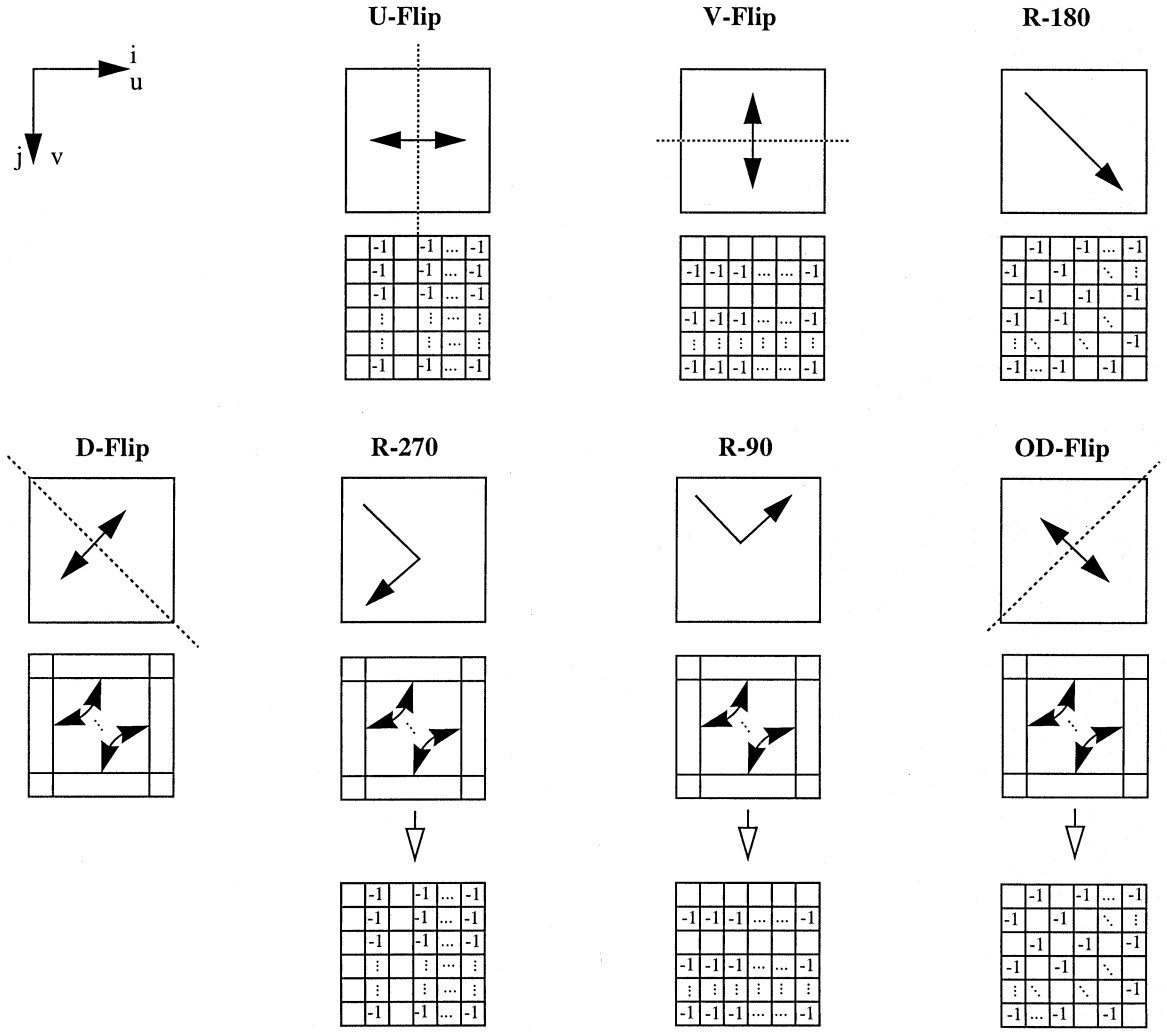


Fig. 4. IBT operations

coefficient exchanges and sign reversals. It is very easy to implement them with almost no overhead. When input and output quantization tables are not identical, we need one additional multiplication of $Q_G(u,v)/Q_F(u,v)$ on sign-reversed coefficients. For operations IBT4 to IBT7, if the user-specified quantization table is not diagonal-symmetric, the right side of the expressions should be scaled by the factor of $Q_G(u,v)/Q_F(u,v)$. Even so, only one multiplication is required for each nonzero coefficient. Assuming seven nonzero coefficients in a typical DCT block, this implies that only seven multiplications are required instead of 108 multiplications required by the brute force method using the fastest DCT algorithm [5].

Figure 4 summarizes the different IBTs in pictorial form. The blocks marked with operation names represent the spatial-domain operations; underneath them, correspondingly, are the DCT domain counterparts. A cell with "- 1" means that the DCT coefficient at that position undergoes a sign reversal. An empty cell means no operation on that coefficient. Note that, irrespective of the operation, the DC coefficient is never changed, as an IBT involves only moving pixels within the same block.

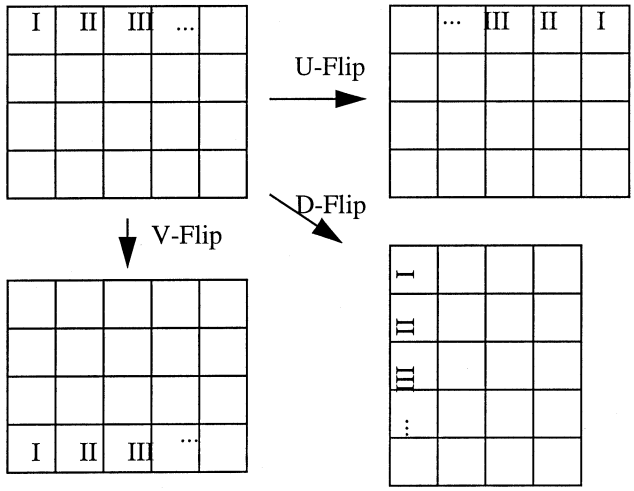


Fig. 5. Block rearrangement, each cell is a block

The actual application of an IBT operation on an entire encoded image should take into account the block rearrangement also. Figure 5 shows examples of block rearrangements for different IBT operations. This block rearrangement in

Table 3. Inner-block transform operations

	Spatial operation	Mathematical expression	DCT coefficients manipulation
IBT0	No change	$G(u,v) = F(u,v)$	no operation
IBT1	Column Flip (U-Flip)	$G(u,v) = \cos(u\pi)F(u,v)$	sign reversal on coefficients of every other row
IBT2	Row Flip (V-Flip)	$G(u,v) = \cos(v\pi)F(u,v)$	sign reversal on coefficients of every other column
IBT3	Rotate 180o (R-180)	$G(u,v) = \cos(u\pi)\cos(v\pi)F(u,v)$	sign reversal on coefficients of every other element
IBT4	Diagonal Flip (D-Flip)	$G(u,v) = F(v,u)$	exchange coefficients of rows with corresponding columns
IBT5	Rotate 270o (R-270)	$G(u,v) = \cos(u\pi)F(v,u)$	exchange row and column, then sign reversal of coefficients on every other row
IBT6	Rotate 90o (R-90)	$G(u,v) = \cos(v\pi)F(v,u)$	exchange row and column, then sign reversal of coefficients on every other column
IBT7	Opposite Diagonal Flip (OD-Flip)	$G(u,v) = \cos(u\pi)\cos(v\pi)F(v,u)$	exchange row and column, then sign reversal of coefficients on every other element

compressed domain requires memory bookkeeping; similar bookkeeping is also needed in spatial domain. Since DCT blocks are kept in compressed form, specifically in run-length-encoded form, the memory overhead, however, is less than that of spatial-domain bookkeeping. On the other hand, some IBT operations, specifically IBT1 to IBT3, only require sign reversals on some nonzero coefficients. This means no loss of precision, which is almost certain to take place if the same operation is to be performed in spatial domain along with a decompress/recompress cycle. This loss of precision can lead to the accumulation of quantization error when the manipulations are to be performed several times, as shown in Fig. 6, where the results of applying three IBT operations repeatedly (ten times) on a JPEG image are shown. The top row of Fig. 6 shows the results as obtained by applying three IBT operations in compressed domain. The bottom row shows the results of same operations applied in spatial domain along with decompression and recompression during each application. The loss in quality of the output is apparent when the IBT operations are not performed in compressed domain. In addition to preserving image quality, the compressed-domain IBT method provides a faster processing speed and requires less memory overhead. These are important advantages in favor of direct processing of compressed images.

5 IBT application examples

In this section, we present three examples of IBT applications for image manipulations directly in compressed domain.

5.1 Subtitling

While previous work in direct compressed-domain manipulation has shown how subtitling can be done, the present application example of IBT to subtitling is different in the respect that it allows a subtitle or logo to be affixed in different orientations. This is a straightforward application of the IBT operations discussed previously. Figure 7a shows a logo whose copy in compressed form is used in the IBT6 operation to generate the result of Fig. 7b. The original flower image (color) and logo image (gray) are both compressed with 75% of quality factor, which gives compression rations of 17.6 and 3.4, respectively. The composited image is generated in compressed domain, also using the same quantization factor. The horizontal line in the logo image shown in Fig. 7a is only of height one pixel, it can be seen that the compressed-domain composition gives reasonably good results even for fine details. It should be noted that, using IBA operations only, we would require a separate logo image for each orientation.

5.2 Shearing

Shearing is an important component of many special-effects manipulations. Here, we show how horizontal shearing can be performed using IBT operations. Shearing involves shifting pixels. Because of the block structure in DCT domain, the compressed domain shearing has to perform shifting on a block-by-block basis. To see how to perform the subblock rearrangement and why we need IBT for management of manipulation matrices, let us consider the example shown in Fig. 8, which illustrates a 45-degree horizontal skew of a 3x3 block.

In this case, the structure of the block has to be rearranged in the way shown in the first row of Fig. 8 (each cell is one pixel). The source block A is shifted horizontally 45 degrees to generate two result blocks: B1 and B2. This operation can be decomposed into the following operations: (1) The extraction of the first row of A, since the first row of result block B1 is identical to that of A. This can be done by pre-matrix multiplication of A; (2) The second row of B1 is the result of extracting the second row of A, then moving it one position to the right. This operation can be done by pre- and post-matrix multiplication of A; (3) The third row of B1 is the result of extracting the third row of A, then moving two positions to the right. Again, this operation can be done by multiplying a pre-matrix and a post-matrix with A. B1 is then constructed by combining the results from step (1)–(3). Similarly, B2 is composed of the results from the rearrangements of the second and third row of A in step (4) and (5), respectively. The steps are depicted in Fig. 8, along with corresponding matrix multiplications.

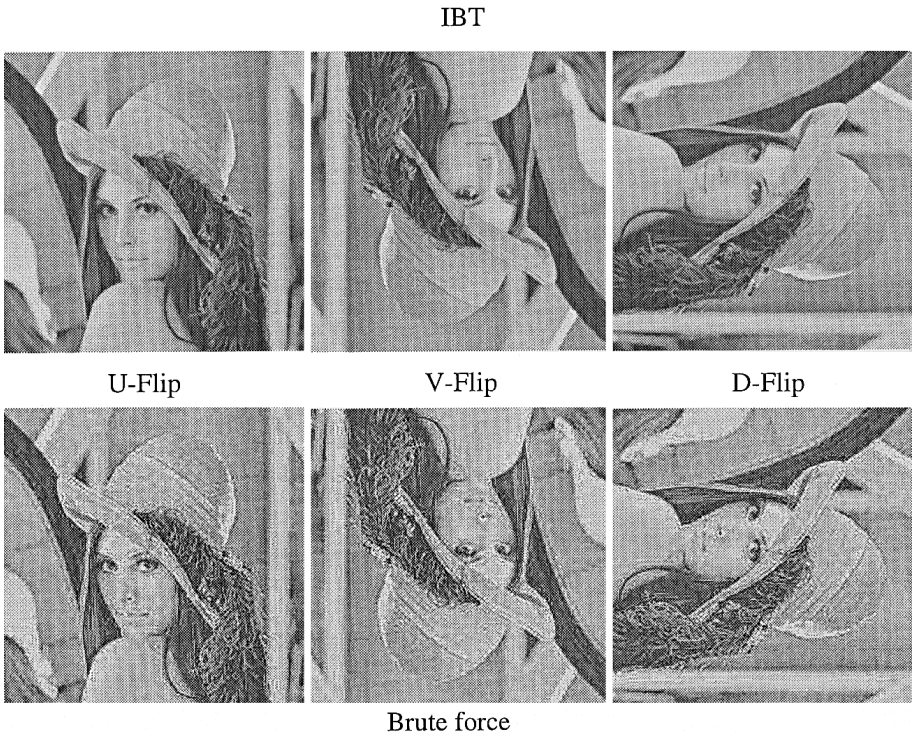


Fig. 6. Effects of U-Flip, V-Flip, and D-Flip (Original image is shown in Fig. 3)

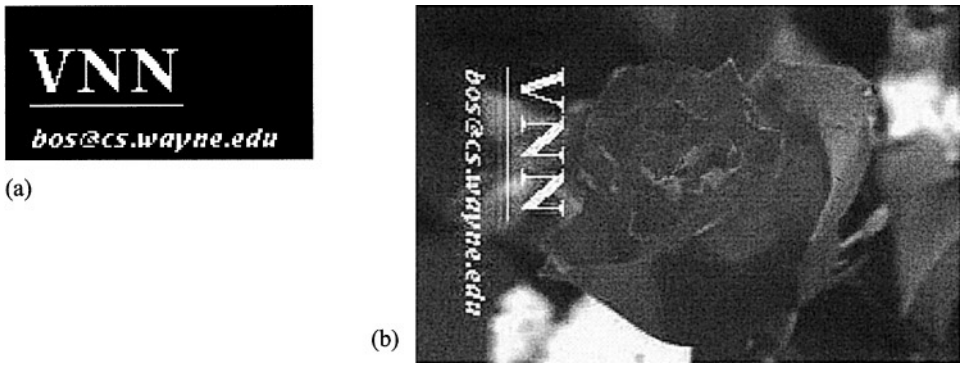


Fig. 7a,b. Special effect subtitling

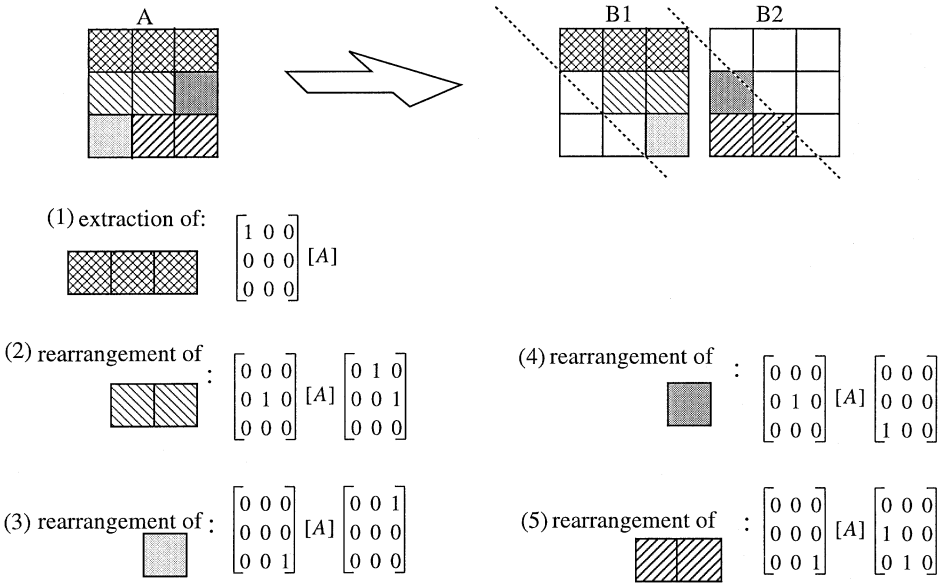


Fig. 8. 45° horizontal skew of a 3×3 block

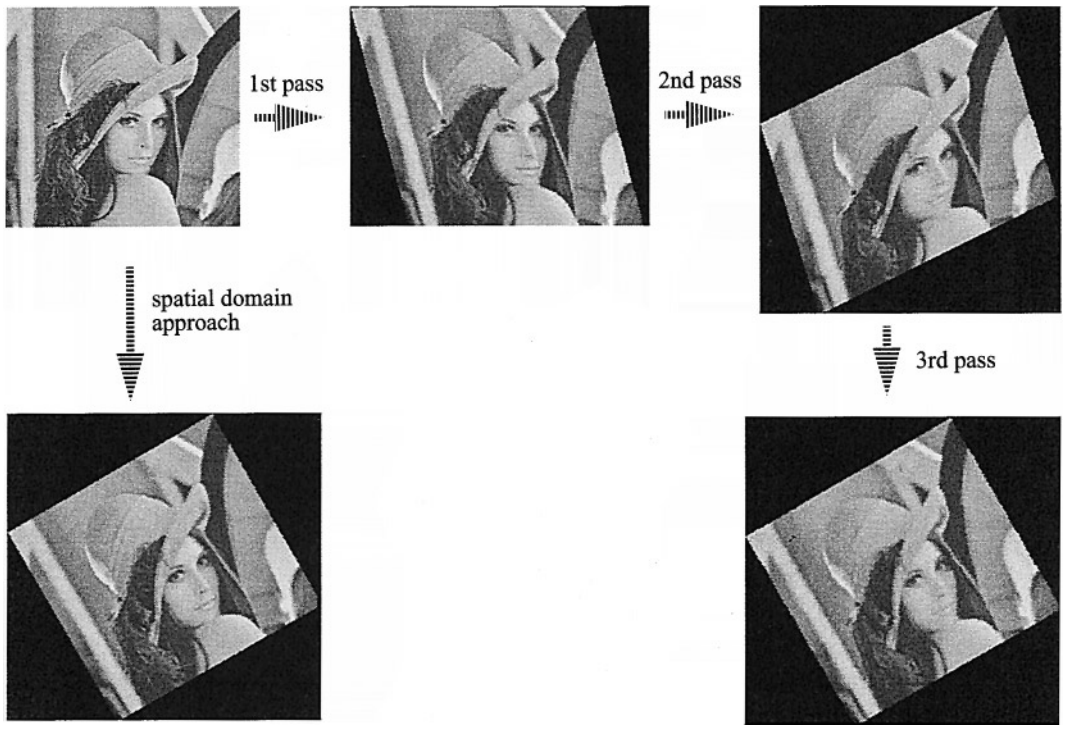


Fig. 9. Three-pass rotation in DCT domain

In the above process, we need a total of seven distinct pre/post-matrices. The whole process can be performed in DCT domain in the same fashion, except that all matrices involved are in DCT form. This means we have to keep in memory all the manipulation matrices in DCT form. But one may notice that the pre-matrix in step (1) is actually the 180°-rotated version of the pre-matrix in (3), U-Flipped version of the post-matrix in (4) and V-Flipped version of the post-matrix in step (3). The post-matrix in step (2) is the 180°-rotated version of the post-matrix in step (5), and so on. We know that flips and 180-degree rotation can be done very easily by reversing signs of certain coefficients in DCT domain. In this example, only three distinct manipulation matrices are needed. All the others can be generated using IBT with almost no overhead.

Since the standard JPEG/DCT block size is 8×8, we need eight distinct manipulation matrices for extracting each row, and 14 distinct manipulation matrices for shifting (7 for left, 7 for right). If we use IBT, only four extracting matrices and six shifting matrices are needed; the remaining matrices can be generated on the fly using IBTs. These matrices will satisfy the requirements of all shearing processes in DCT domain, either horizontal or vertical and for arbitrary amount. Of course, in the general case, when the displacement of each row or column is not an integer, we would also need proper interpolation to avoid an aliasing effect.

5.3 Arbitrary rotation

The capability to perform arbitrary rotation of an image directly in compressed domain is important, as rotation is an often used image manipulation step. Generally, rotation can be implemented by direct mapping. For each output pixel, the mapping function can decide which input pixels contribute to it and by how much. The problem is that the output pixel can take intensities from as many as six input pixels. Furthermore, the output pixel coverage of adjacent input pixels is nonperiodic. This calls for intensive computation for intersection test and creates difficulties in developing a clean algorithm structure. It is now a common understanding that rotation and many other sophisticated geometric transformations can be decomposed to separable passes, with each pass requiring only shifting and resampling in one direction. In other words, a combination of shearing processes can yield the rotated image.

Many multi-pass rotation algorithms have been developed in spatial domain [2, 6, 15, 17]. We describe here an implementation for performing arbitrary angle rotation in compressed domain based on a three-pass rotation algorithm [15]. In the three-pass rotation algorithm, the rotation is decomposed into three passes:

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\tan\frac{\theta}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & \sin\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\tan\frac{\theta}{2} & 1 \end{bmatrix}. \quad (10)$$

The algorithm first skews the image along the horizontal direction by displacing each row. The result is then skewed along the vertical direction. Finally, an additional skew in the horizontal direction yields the rotated image. Again, the displacement is generally not integral and proper interpolation has to be performed in each pass.

Figure 9 shows the effects after each pass, and also compares the final effects of the compressed-domain rotation versus that of spatial-domain approach. There is no perceptual difference between the two results, although both versions suffer minor blurring compared to the original image. This degradation is introduced by a roundoff integer operation. If we have fast floating-point hardware, we can perform

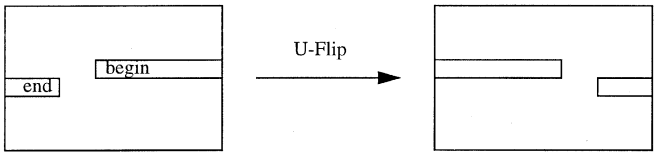


Fig. 10. U-Flip on one slice generated two slices

the operation in real numbers to obtain better results. This algorithm shows a way to adopt the concept of separable pass processing into compressed domain, so that simplified, systematic algorithms can be designed for special- effects manipulation directly in compressed domain.

6 Manipulation of compressed video

While the previous examples dealt with encoded images, we now present an application example of IBT to compressed video. There are currently two dominant types of DCT-based video compression models: M-JPEG (Motion-JPEG) and MPEG (Moving Picture Expert Group). A M-JPEG video is composed of a sequence of JPEG-compressed images, and thus has no temporal redundancy feature. MPEG considers both spatial redundancy and temporal redundancy reduction by using DCT and motion-compensated prediction/ interpolation [7]. Since each frame of a M-JPEG video is a standard JPEG image, it is obvious that all inner block operations (IBOs) described earlier can be applied immediately on M-JPEG-compressed video to generate, for example, flipped or special-angle-rotated video in compressed form. However, for MPEG-compressed video, we need to consider two unique factors: the manipulation of motion vectors (MVs) and motion-compensated prediction error (MCPE).

An MPEG sequence consists of three different types of encoded frames: intraframes (I frames), predicted frames (P frames), and interpolated bi-directional frames (B frames). The I frames are coded independently of other frames in a video to provide random access to the MPEG bitstream. Each I frame is encoded in a JPEG-like scheme, and therefore all IBOs are immediately applicable.

The P frames are encoded by motion prediction using past frames, either an I frame or a P frame. The encoding of P frames as well as B frames is done at macroblock level. Each macroblock is of size 16×16 and consists of four blocks of size 8×8. If the motion-compensated prediction error e is larger than certain threshold for a macroblock in a P or B frame, then the macroblock is actually intracoded, i.e., just like blocks in an I frame. Otherwise, e itself is intracoded and transmitted along with MV information. In order to see how IBOs can be applied to such macroblocks, let us consider the following reconstruction relationship:

$$b_c = IDCT\,(B_r) + \overline{mv} + IDCT(E)\ ,$$

where b_c is the reconstructed block, B_r is the reference block, $\overline{mv}$ is the motion vector and E is the MCPE in DCT form. Let us use the IBT1 (Column Flip) operation as an example. Applying this operation on the reconstructed block b_c in spatial domain, we have

$$\begin{aligned} ibt1\,(b_c) &= ibt1\,(IDCT\,(B_r) + \overline{mv} + IDCT(E)) \\ &= ibt1\,(IDCT\,(B_r)) + ibt1\,(\overline{mv}) \\ &\quad + ibt1(IDCT(E))\ . \end{aligned} \tag{11}$$

Since B_r and E are all DCT blocks, we already know that

$$ibt1(IDCT(B_r)) = IBT1(B_r)\ ,$$
$$ibt1(IDCT(E)) = IBT1(E)\ .$$

We then only have to define ibt1($\overline{mv}$) in Eq. 11 to complete the application of an IBT operation in compressed-video domain. A motion vector, {x, y} contains two components, horizontal displacement and vertical displacement. IBTs can be easily defined for MVs, since all the operations are done along the two orthogonal directions. Thus,

Column flip	ibt1({x, y}) = {-x, y}
Row flip	ibt2({x, y}) = {x, -y}
Rotate 180	ibt3({x, y}) = {-x, -y}
Diagonal flip	ibt4({x, y}) = {y, x}
Rotate 270	ibt5({x, y}) = {y, -x}
Rotate 90	ibt6({x, y}) = {-y, x}
Opposite-D flip	ibt7({x, y}) = {-y, -x}

Note that motion information is coded differentially with respect to the motion information present in the previous adjacent macroblock; this means that an MV is represented as $\{\delta_x,\ \delta_y\}$ with

$$x = X + \delta_x\ , y = Y + \delta_y\ ,$$

where {X, Y} is the MV of the reference macroblock. It can be easily seen that it is not even necessary to decode the predictively encoded MV. The same IBT operations applied on $\{\delta_x,\ \delta_y\}$ would give out the same results. However, since MPEG resets MV predication at the beginning of each slice or when the previous macroblock is intracoded (or skipped in P frames), the algorithm has to keep track of the correct reference macroblock in order to apply the MV adjustment directly on differentially encoded MVs.

For B frames where macroblocks can be estimated from macroblocks in both past and future reference frames, the decoding includes motion prediction using both forward and backward MVs. In this case, the IBT operations on the macroblock include IBTs on the prediction error block, as well as adjustments on both forward and backward MVs, as we discussed above.

In the actual implementation, we have to sometimes adjust or even insert new syntax symbols in the MPEG stream, in order to guarantee that the output stream is syntactically compliant with the MPEG specification. We now discuss three out of many situations in which syntax manipulations are required by direct MPEG video editing.

Similar to the block rearrangement required when performing IBTs on JPEG images, the location of each macroblock, along with its associated side information such as macroblock type, MV and coded block pattern, have to be rearranged. And furthermore, individual blocks within each macroblock also undergo rearrangement as per the selected IBT. This leads to the necessity of adjustment on some syntax side information. For instance, MPEG allows certain blocks, indicated by the coded block pattern, within a macroblock to be “Not Coded”. After an IBT operation, say column flip, this not-coded block is relocated to a different

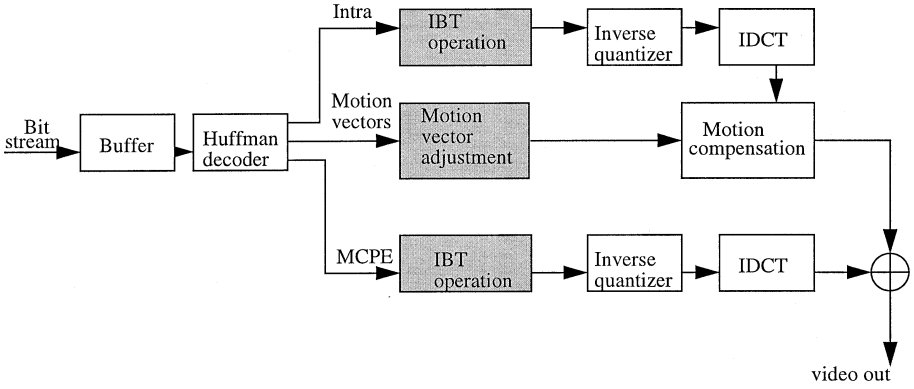


Fig. 11. Flipped or special angle-rotated video play-back decoder

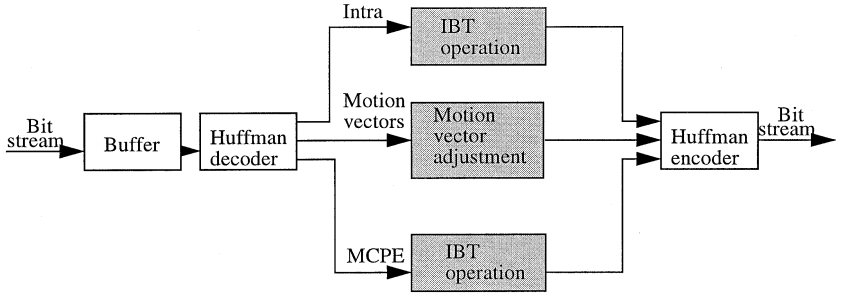


Fig. 12. Video editing from compressed bitstream to compressed bitstream

place, therefore, the algorithm must adjust the syntax symbol of the coded block pattern accordingly.

According to the MPEG standard, a macroblock can be skipped, that is, not coded at all. In B frames, a skipped macroblock M is defined as a macroblock with zero prediction error and the same MVs as that of the previous adjacent macroblock, that is, the macroblock to its left. Obviously, after column flipping, the relative order of the macroblocks has been changed (the macroblock originally to the right of M is now to M's left). It should be noted that the two adjacent macroblocks of M might have different MVs. Thus, if we still code M as a skipped macroblock, it will be decoded incorrectly when the output stream is played back. Our algorithm will solve this problem by coding M as forward and/or backward predicated with zero predication error. This means that we actually insert new syntax symbols which do not exist for M in the original MPEG stream.

Another problem lies in the slice layer of the MPEG standard. Once again, consider column flipping where the macroblocks inside each macroblock slice is flipped, and thus the last macroblock in the original slice becomes the first macroblock in the output slice. Since both MPEG-1 and MPEG-2 allow slices to start at any macroblock, then in the situation shown in Fig. 10, two separated slices have to be generated from one slice in the input MPEG bitstream. More slices certainly impose more overhead because of the coding of slice headers. For most MPEG sequences, however, we do not have to worry about the above situation, since slices start from the left edge of a frame and end at the right edge (specified in MPEG2 Test Model 3 [9]).

Based on the above discussion, Fig. 11 summarizes a scheme for flipped or special-angle-rotated video playback. Without the shaded blocks, it is a typical motion-compensation-based video decoder like MPEG. Figure 12 shows a scheme for another application – generating flipped or special-angle-rotated video from MPEG stream to MPEG stream with minimum processes in between. The operations in the shaded blocks require very little overhead. For instance, for U-Flipped playback, the operations required in MV adjustment only include sign reversal of horizontal MVs, as we discussed above. Both of these schemes have been implemented and speedups of at least 1000% over spatial processing coupled with decompression and recompression have been obtained.

Other video manipulation operations such as fade in/out or cross-dissolving are local operations which do not require moving pixels around. IBA and IBR operations can be used directly on a frame-by-frame basis to achieve the computation saving for video processing.

7 Summary and conclusion

In this paper, the problem of manipulation of images in compressed form was considered and a new set of methods, the IBT methods, were presented. Unlike the previously developed manipulation methods for compressed images, IBT methods are suitable for geometric transformations which are the basis of sophisticated special effects that are often needed in advertising, entertainment, and education. This capability of IBT methods in the present paper was demonstrated by several examples.

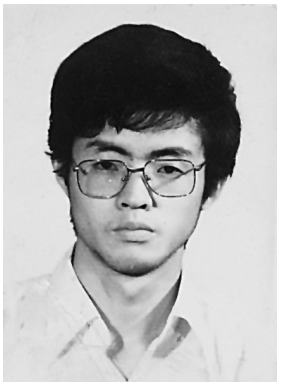
The paper also described the use of IBT methods for direct M-JPEG and MPEG video manipulations, along with motion field and syntax adjustment imposed by this kind of manipulation. Since our method completely eliminates the processes of IDCT and motion compensation, and DCT and motion estimation by avoiding decompression-recompression cycles, it provides tremendous speedups for digital video manipulation in situations where video after manipulation

needs to be stored or forwarded in compressed form. Also, our video manipulation method is completely independent of the actual method used in DCT/IDCT and motion estimation/compensation.

In addition to gains in speed, the direct manipulation of compressed images and videos through IBT allows image and video quality to be maintained. Another advantage of the IBT methods is the relatively smaller memory requirement compared to other direct processing methods. We are currently investigating methods for further improvements in the use of IBT to the applications discussed above, as well as to new manipulation and feature extraction tasks.

References

1. Arai Y, Agui T, Nakajima M (1988) A Fast DCT-SQ Scheme for Images. Trans. IEICE E71(11):1095
2. Catmull E, Smith AR (1980) 3-D Transformations of Images in Scanline Order. Comput Graphics 14(3): 279–285
3. Chang S-F, Messerschmitt DG (1995) Manipulation and Compositing of MC-DCT Compressed Video. IEEE JSAC Special Issue on Intell Signal Process 13(1): 1–11
4. Chang S-F, Messerschmitt DG (1992) Video Compositing in the DCT domain. Proc. IEEE Workshop on Visual Signal Processing and Communications, Raleigh, NC, pp 138–143
5. Feig E, Winograd S (1992) Fast Algorithms for the Discrete Cosine Transform. IEEE Trans Signal Process 40(9): 2174–2193
6. Foley JD, Van Dam A, Feiner SK, Hughes JF (1990) Computer Graphics: Principles and Practice, 2nd Ed., Addison-Wesley, Reading, Mass.
7. Gall DL (1991) MPEG: A Video Compression Standard for Multimedia Applications. Commun ACM 34(4):47–58
8. Haffman DA (1962) A method for the construction of minimum redundancy codes. Proc. IRE 40: 1098–1101
9. ISO-IEC/JTS1/SC29/WG11 (1992) Coded Representation of Picture and Audio Information, Test Model 3
10. Jain AK (1989) Fundamentals of Digital Image Processing. Prentice-Hall, Englewood Cliffs, NJ
11. Merhav N, Bhaskaran V (1996) A Transform Domain Approach to Spatial Domain Image Scaling. Proc. ICASSP Conf. Atlanta, June 1996, pp 2307–2310
12. Pennebaker WB (1992) ISO Draft International Standard 10918 Part 1, Requirements and Guidelines. JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, New York
13. Smith BC, Rowe L (1993) Algorithms for Manipulating Compressed Images. IEEE Comput Graphics Appl 13(9):34–42
14. Smith BC (1994) Fast Software Processing of Motion JPEG Video. Proc. of the Second ACM International Conference on Multimedia, San Francisco, Calif., ACM Press, pp 77–88
15. Tanaka A, et al. (1986) A Rotation Method for Raster Image Using Skew Transformation. Proc. IEEE Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, pp 272–277
16. Wallace GK (1991) The JPEG Still Picture Compression Standard. Commun ACM 34(4):31–44
17. Weiman CFR (1980) Continuous Anti-Aliased Rotation and Zoom of Raster Images. Comput Graphics 14(3): 286–293
18. Porter T, Duff T (1984) Compositing Digital Images. Comput Graphics (Proc. SIGGRAPH 84) 18(3): 253–259



BO SHEN received his B.S. degree in computer science from Nanjing Aeronautical Institute, Nanjing, China in 1988. He then served as a software engineer at local computer/software companies in the following 4 years. He joined Wayne State University, Detroit, Michigan in 1993 and is currently a Ph.D. candidate in the Department of Computer Science. He worked at Hewlett-Packard Laboratories as a research intern during summer, 1996. His research interests include image/video processing and content-based retrieval, multimedia systems, computer graphics and computer vision.

ISHWAR K. SETHI received the BTech (Hons.), MTech, and PhD degrees in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1969, 1971, and 1977, respectively. He is currently a professor of computer science at Wayne State University, Detroit, Michigan. Prior to joining Wayne State University in 1982, he was on the faculty at the Indian Institute of Technology. His current research intersts are in the areas of artificial neural networks, computer vision, pattern recognition, and multimedia systems. He is co-editor of the book Artificial Neural Networks and Statistical Pattern Recognition (North-Holland, 1991). He currently serves on the editorial boards of Pattern Recognition, Pattern Recognition Letters, and Machine Vision and Applications. Dr. Sethi is the co-chair of the IS&T/SPIE Conference on Storage and Retrieval for Image and Video Databases.