A Fast Two-Dimension 4× 4 Inverse Integer Transform Algorithm for Real-time H. 264 Decoder

Data *in* International journal of innovative computing, information & control: IJICIC · May 2015

CITATIONS
0

READS
30

3 authors, including:

Xh Ji
Shandong University of Finance and Economics
13 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Caiming Zhang
Shandong University
198 PUBLICATIONS **868** CITATIONS
SEE PROFILE

International Journal of Innovative Computing, Information and Control
Volume **5**, Number **3**, March **2009**

ICIC International ©2009 ISSN 1349-4198
pp. ICIC07-IS03-07

A FAST TWO-DIMENSION 4×4 INVERSE INTEGER TRANSFORM ALGORITHM FOR REAL-TIME H.264 DECODER

Xiuhua Ji[1,2], Caiming Zhang[1,2] and Kai Wang[1]

[1]School of Computer Science and Technology
Shandong Economic University
Jinan 250014, P. R. China
{jixiuhua; zhangcm}@sdie.edu.cn; Kai.Wang@usd.edu

[2]School of Computer Science and Technology
Shandong University
Jinan 250061, P. R. China

Received February 2008; revised July 2008

Abstract. *This paper presents a new fast two-dimensional* 4×4 *(2-D* 4×4*) inverse integer transform algorithm. The algorithm takes advantage of the symmetries of basic images and makes use of the characteristic of the transform data of practical videos. Compared to other fast algorithms, the number of addition and shift operations for the inverse transform can be reduced greatly. Numerical results on several standard video clips indicate that on average, for a* 4×4 *block, the new algorithm needs 12.7838 addition operations and 1.69536 shift operations, which are much less than the other algorithms need. Moreover, the new 2-D algorithm can be parallelized easily.*
Keywords: DCT, Basic image, Image compression, Integer transform

1. **Introduction.** Transform coding has been widely used in image and video coding standards, such as JPEG, MPEG-2, MPEG-4, and H.264. H.264 is the newest video coding standard. It has many new features to achieve significant improvement in coding efficiency and provides more "network friendly" video representation than other existing standards do [1-3]. However, it requires enormous number of computations. This limits the rapid application of H.264 standard.

Just like MPEG-2, H.264 applies the transform coding to the prediction residual. The transform used in h.264, however, is a 4×4 or 8×8 integer transform instead of the 8×8 Discrete Cosine Transform (DCT) that MPEG-2 uses. The integer transform is originated from DCT, but has lower complexity with little performance degradation. It only involves addition and shift operations, and no mismatch exists between the forward and inverse transforms [1,2]. This reduces the complexity significantly compared to DCT. In recent years, many researchers have designed and developed fast algorithms for the integer transforms and integer DCT [4-10]. The most influential fast algorithms for 2-D 4×4 inverse integer transform algorithm at present are in literatures [5,6]. The fast algorithm in [5] treats 2D 4×4 integer transform as a row-column application of the 1-D methods, i.e., the 1-D 4 dots integer transform is separately implemented in each of rows and columns in turn. The algorithm that Fan et al. introduced in [6] is based on sparse factorizations of the matrix. Table 1 shows the comparisons of computational complexity for the fast 2-D 4×4 integer transforms using different algorithms.

In this paper, we propose a novel 2-D fast algorithm for realization of the 4×4 inverse integer transform in H.264. The new fast algorithm is based on basic images of the transform and can exploit the data distribution of transform coefficients. As a result, the

computational complexity of the 2-D 4×4 inverse integer transform is reduced greatly. The decoding speed is improved. The rest of this paper is organized as follows. In Section 2, we review the 2-D 4×4 integer transform and its development from DCT. The expression of the 4×4 integer transform in basic images is derived, and then the symmetries of basic images are introduced and a new algorithm is developed. Section 3 reports the experimental results. Section 4 is the conclusions.

TABLE 1. Comparisons of computational complexity for 2-D 4×4 integer transforms

Operations/Algorithms	Row-Column without Fast Algorithm	Row-Column Method with [5]	matrix factorization Method with [6]
Additions	96	64	64
Shifts	32	16	16

2. **The Proposed Algorithm.** The new fast algorithm is based on basic images of the 4×4 integer transform. It reduces the number of addition and shift operations greatly by making use of the symmetries of basic images and the characteristic of the transform coefficients of practical videos. In this section, we first introduce the 4×4 integer transform and the concept of basic images of the integer transform, and then we explain how to utilize the symmetries of basic images and the characteristic of the transform coefficients to reduce the computational complexity for the 4×4 inverse integer transform.

2.1. **The 2D 4×4 integer transform and its development from DCT.** The integer transform operates on 4×4 blocks of residual data after motion-compensated prediction or intra prediction [2]. It is based on DCT but with some fundamental differences. It is an integer transform so that all operations in it can be carried out with integer arithmetic, without losing any accuracy.

The 2-Dimensional 4×4 forward DCT is defined as

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{A}\boldsymbol{X}\boldsymbol{A}^T \\ &= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \boldsymbol{X} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \end{aligned} \tag{1}$$

where $a = \frac{1}{2}$, $b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$, $c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$, and $\boldsymbol{X}$ stands for the original input matrix. The matrix computation in (1) can be factorized to the following equivalent form [2]:

$$\boldsymbol{Y} = (\boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^T) \otimes \boldsymbol{E},$$

where

$$\boldsymbol{C} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix}, \quad \boldsymbol{E} = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

and the symbol $\otimes$ stands for the operation that each element of $(\boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^T)$ is multiplied by the scaling factor in the same position in matrix $\boldsymbol{E}$ (scalar multiplication rather than matrix multiplication). $\boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^T$ is the core part of the 2-D transform. To simplify the implementation of the transform, d is approximated by 0.5 to avoid multiplications in the

core forward transform. To ensure that the transform remains orthogonal, b also needs to be modified. The final $4\times$ 4 forward transform becomes

$$\boldsymbol{Y} = (\boldsymbol{C}_f \boldsymbol{X} \boldsymbol{C}_f^T) \otimes \boldsymbol{E}_f \tag{2}$$

Here,

$$\boldsymbol{C}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}, \quad \boldsymbol{E}_f = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix},$$

$a = \frac{1}{2}$, and $b = \sqrt{\frac{2}{5}}$. The output of the new transform in Equation (2) will no longer be the 4×4 DCT (1) because the factors d and b have changed. The transform (2) is an approximation to the 4×4 DCT (1). The core part of the transform is multiplication-free, i.e., it only requires addition and shift operations. The post-scaling operation "$\otimes\boldsymbol{E}_f$" requires one multiplication for every coefficient, which can be absorbed into the quantization process to reduce the total number of multiplications.

The inverse transform (defined in [1]) is given by

$$\boldsymbol{X'} = \boldsymbol{C}_t^T (\boldsymbol{Y} \otimes \boldsymbol{E}_t) \boldsymbol{C}_t = \boldsymbol{C}_t^T \cdot \boldsymbol{F} \cdot \boldsymbol{C}_t \tag{3}$$

where

$$\boldsymbol{C}_t = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}, \quad \boldsymbol{E}_t = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}, \text{ and } \boldsymbol{F} = \boldsymbol{Y} \otimes \boldsymbol{E}_t.$$

In fact, the pre-scaling operation "$\otimes\boldsymbol{E}_t$" in (3) can be merged into the inverse quantization process to reduce the total number of multiplications. Note the factors $\pm\frac{1}{2}$ in the matrices $\boldsymbol{C}_t$ and $\boldsymbol{C}_t^T$, the corresponding multiplication operation can be reduced to a right-shift operation without a significant loss of accuracy because the coefficients of $\boldsymbol{Y}$ are pre-scaled. In the following sections, we will focus on the derivation of the fast inverse integer transform ($\boldsymbol{C}_t^T \boldsymbol{F} \boldsymbol{C}_t$) algorithm in (3).

2.2. **Expressing the 4×4 integer transform in basic images.** Let $\boldsymbol{G}_u$ ($u = 0, 1, 2, 3$) be the vector which locates in the uth row of $\boldsymbol{C}_t$, then Equation (3) can be written as

$$\boldsymbol{X}^t = \begin{bmatrix} \boldsymbol{G}_0^T & \boldsymbol{G}_1^T & \boldsymbol{G}_2^T & \boldsymbol{G}_3^T \end{bmatrix} \cdot \boldsymbol{F} \cdot \begin{bmatrix} \boldsymbol{G}_0 \\ \boldsymbol{G}_1 \\ \boldsymbol{G}_2 \\ \boldsymbol{G}_3 \end{bmatrix} \tag{4}$$

where $\boldsymbol{G}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$, $\boldsymbol{G}_1 = \begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$, $\boldsymbol{G}_2 = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}$, and $\boldsymbol{G}_3 = \begin{bmatrix} \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$.

We can rewrite the matrix $\boldsymbol{F}$ in the following form

$$\boldsymbol{F} = \begin{bmatrix} F(0,0) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & F(0,1) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \cdots + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & F(3,3) \end{bmatrix} \tag{5}$$

Substitute Equation (5) into Equation (4) and then we get

$$\begin{aligned} \boldsymbol{X}^t &= \sum_{u=0}^{3}\sum_{v=0}^{3}\left(F(u,v)\times \boldsymbol{G}_u^T \times \boldsymbol{G}_v\right) \\ &= \sum_{u=0}^{3}\sum_{v=0}^{3}\left(F(u,v)\times \boldsymbol{B}_{uv}\right) \end{aligned} \tag{6}$$

Here, $\boldsymbol{B}_{uv} = \boldsymbol{G}_u^T \times \boldsymbol{G}_v$ is defined as the basic image of the transform coefficient $F(u,v)$ $(u,v=0,1,2,3)$. Equation (6) indicates that the matrix $\boldsymbol{X}^t$ can be gotten by adding all the basic images linearly with the transform coefficients $F(u,v)$ $(u,v=0,1,2,3)$ as weights. The proposed algorithm in this paper is based on Equation (6).

2.3. **Symmetry of basic image $\boldsymbol{B}_{uv}$.** The proposed algorithm makes use of the symmetries of basic images to reduce operations for computing (6). All the basic images $\boldsymbol{B}_{uv} = \boldsymbol{G}_u^T \times \boldsymbol{G}_v$ $(u,v=0,1,2,3)$ are defined as follows

$$\boldsymbol{B}_{20} = \begin{bmatrix} \boxed{1} & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{B}_{02} = \begin{bmatrix} \boxed{1} & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \boldsymbol{B}_{00} = \begin{bmatrix} \boxed{1} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\boldsymbol{B}_{22} = \begin{bmatrix} \boxed{1} & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \boldsymbol{B}_{10} = \begin{bmatrix} \boxed{1} & 1 & 1 & 1 \\ \boxed{1/2} & 1/2 & 1/2 & 1/2 \\ -1/2 & -1/2 & -1/2 & -1/2 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

$$\boldsymbol{B}_{03} = \begin{bmatrix} \boxed{1/2 & -1} & 1 & -1/2 \\ 1/2 & -1 & 1 & -1/2 \\ 1/2 & -1 & 1 & -1/2 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}, \quad \boldsymbol{B}_{30} = \begin{bmatrix} \boxed{1/2} & 1/2 & 1/2 & 1/2 \\ \boxed{-1} & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ -1/2 & -1/2 & -1/2 & -1/2 \end{bmatrix}$$

$$\boldsymbol{B}_{01} = \begin{bmatrix} \boxed{1 & 1/2} & -1/2 & -1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & 1/2 & -1/2 & -1 \end{bmatrix}, \quad \boldsymbol{B}_{32} = \begin{bmatrix} \boxed{1/2} & -1/2 & -1/2 & 1/2 \\ \boxed{-1} & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1/2 & 1/2 & 1/2 & -1/2 \end{bmatrix}$$

$$\boldsymbol{B}_{23} = \begin{bmatrix} \boxed{1/2 & -1} & 1 & -1/2 \\ -1/2 & 1 & -1 & 1/2 \\ -1/2 & 1 & -1 & 1/2 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}, \quad \boldsymbol{B}_{11} = \begin{bmatrix} \boxed{1 & 1/2} & -1/2 & -1 \\ \boxed{1/2 & 1/4} & -1/4 & -1/2 \\ -1/2 & -1/4 & 1/4 & 1/2 \\ -1 & -1/2 & 1/2 & 1 \end{bmatrix}$$

$$\boldsymbol{B}_{21} = \begin{bmatrix} \boxed{1 & 1/2} & -1/2 & -1 \\ -1 & -1/2 & 1/2 & 1 \\ -1 & -1/2 & 1/2 & 1 \\ 1 & 1/2 & -1/2 & -1 \end{bmatrix}, \quad \boldsymbol{B}_{31} = \begin{bmatrix} \boxed{1/2 & 1/4} & -1/4 & -1/2 \\ \boxed{-1 & -1/2} & 1/2 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ -1/2 & -1/4 & 1/4 & 1/2 \end{bmatrix}$$

$$\boldsymbol{B}_{12} = \begin{bmatrix} \boxed{1} & -1 & -1 & 1 \\ \boxed{1/2} & -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & 1/2 & -1/2 \\ -1 & 1 & 1 & -1 \end{bmatrix}, \quad \boldsymbol{B}_{33} = \begin{bmatrix} \boxed{1/4 & -1/2} & -1/2 & -1/4 \\ \boxed{-1/2 & 1} & -1 & 1/2 \\ 1/2 & -1 & 1 & -1/2 \\ -1/4 & 1/2 & -1/2 & 1/4 \end{bmatrix}$$

$$\boldsymbol{B}_{13} = \begin{bmatrix} \boxed{1/2 & -1} & 1 & -1/2 \\ \boxed{1/4 & -1/2} & 1/2 & -1/4 \\ -1/4 & 1/2 & -1/2 & 1/4 \\ -1/2 & 1 & -1 & 1/2 \end{bmatrix}$$

By analyzing all the basic images, we observed the following properties.

Property 2.1. *The small matrix in the dashed rectangle in each basic image* $\boldsymbol{B}_{uv}$ *is called the smallest unit of* $\boldsymbol{B}_{uv}$*, written as* $\overline{\boldsymbol{B}}_{uv}$*. Each basic image can be gotten by extending its smallest unit according to the odd or even symmetries in the horizontal and vertical directions.*

Property 2.2. $\boldsymbol{B}_{01}$ *and* $\boldsymbol{B}_{03}$ *have the same symmetry and they can be put into one group* ($\boldsymbol{B}_{01}$, $\boldsymbol{B}_{03}$)*. So are the groups* ($\boldsymbol{B}_{10}$, $\boldsymbol{B}_{30}$), ($\boldsymbol{B}_{12}$, $\boldsymbol{B}_{32}$), ($\boldsymbol{B}_{21}$, $\boldsymbol{B}_{23}$), *and* ($\boldsymbol{B}_{11}$, $\boldsymbol{B}_{33}$, $\boldsymbol{B}_{13}$, $\boldsymbol{B}_{31}$).

2.4. **Computing the 4×4 inverse integer transform.** In order to reduce computational complexity, we compute Equation (6) in the following way.

Step 1: Utilizing the symmetries of the basic images $\boldsymbol{B}(u, v)$ $(u, v = 0, 1, 2, 3)$, the 16 matrices $F(u, v) \times \boldsymbol{B}_{uv}$ $(u, v = 0, 1, 2, 3)$ are classified into 8 groups. The terms in each group are added in turn. The details are as follows.

According to the symmetries of $\boldsymbol{B}_{00}$ and $\boldsymbol{B}_{02}$, the computation of $\boldsymbol{Z}_1 = F(0,0) \times \boldsymbol{B}_{00} + F(0,2) \times \boldsymbol{B}_{02}$ needs 2 additions and the size of the smallest unit of the result matrix $\boldsymbol{Z}_1$ is (1×2). Similarly, that of

$\boldsymbol{Z}_2 = F(2,0) \times \boldsymbol{B}_{20} + F(2,2) \times \boldsymbol{B}_{22}$ needs 2 additions;
$\boldsymbol{Z}_3 = F(0,1) \times \boldsymbol{B}_{01} + F(0,3) \times \boldsymbol{B}_{03}$ needs 2 additions and 2 right-shifts;
$\boldsymbol{Z}_4 = F(2,1) \times \boldsymbol{B}_{21} + F(2,3) \times \boldsymbol{B}_{23}$ needs 2 additions and 2 right-shifts;
$\boldsymbol{Z}_5 = F(1,0) \times \boldsymbol{B}_{10} + F(3,0) \times \boldsymbol{B}_{30}$ needs 2 additions and 2 right-shifts;
$\boldsymbol{Z}_6 = F(1,2) \times \boldsymbol{B}_{12} + F(1,2) \times \boldsymbol{B}_{12}$ needs 2 additions and 2 right-shifts;
$\boldsymbol{Z}_7 = F(1,1) \times \boldsymbol{B}_{11} + F(3,3) \times \boldsymbol{B}_{33}$ needs 3 additions and 3 right-shifts;
$\boldsymbol{Z}_8 = F(1,3) \times \boldsymbol{B}_{13} + F(3,1) \times \boldsymbol{B}_{31}$ needs 3 additions and 3 right-shifts;

Step 2: The $\boldsymbol{Z}_1 \sim \boldsymbol{Z}_8$ matrices are added in turn according to the computing order of Equation (7) and the result matrix is $\boldsymbol{X}^t$.

$$\boldsymbol{X}^t = [(\boldsymbol{Z}_1 + \boldsymbol{Z}_2) + (\boldsymbol{Z}_3 + \boldsymbol{Z}_4)] + [(\boldsymbol{Z}_5 + \boldsymbol{Z}_6) + (\boldsymbol{Z}_7 + \boldsymbol{Z}_8)] \tag{7}$$

The size of the smallest unit of $\boldsymbol{Z}_1$ or $\boldsymbol{Z}_2$ is (1×2) or (2×1) respectively so that the computation of $\boldsymbol{W}_1 = (\boldsymbol{Z}_1 + \boldsymbol{Z}_2)$ needs 4 additions and the size of the smallest unit of the matrix $\boldsymbol{W}_1$ is (2×2). Similarly, that of $\boldsymbol{W}_2 = (\boldsymbol{Z}_3 + \boldsymbol{Z}_4)$, $\boldsymbol{W}_4 = \boldsymbol{Z}_5 + \boldsymbol{Z}_6$ or $\boldsymbol{W_5} = \boldsymbol{Z}_7 + \boldsymbol{Z}_8$ needs 4 additions individually; that of $\boldsymbol{W}_3 = (\boldsymbol{W}_1 + \boldsymbol{W}_2)$ or $\boldsymbol{W}_6 = (\boldsymbol{W}_4 + \boldsymbol{W}_5)$ needs 8 additions; and that of $\boldsymbol{X}^t = (\boldsymbol{W}_3 + \boldsymbol{W}_6)$ needs 16 additions.

Based on the discussion above, we can see that the total number of computations of Equation (6) is 66 additions and 14 right-shifts. In addition, the matrix computations in the proposed algorithm can be implemented easily on any parallel computing architecture, which makes the algorithm suitable for parallel computing.

2.5. **Further reducing the operations by using the characteristic of actual data.** In actual video coding process, most of the quantized transform coefficients $F(u, v)$ are zero. By using JM10.1 Reference Software, we can get the average number of non-zero quantized transform coefficients per non-all-zero 4×4 block. Table 2 shows the statistical results on a few standard video clips (Figure 1). Of the 16 coefficients, the average number of non-zero transform coefficients is 2.660267, the rest are zero. That implies, in computing Equation (6), many of the terms $F(u, v) \times \boldsymbol{B}_{uv}$ $(u, v = 0, 1, 2, 3)$ are equal to a zero-matrix. In our algorithm, we only let those non-zero terms $(F(u, v) \times \boldsymbol{B}_{uv})$ perform the computation. Therefore, the number of the operations of Equation (6) can be reduced significantly.

Although the algorithms in literatures [5,6] can also utilize the characteristic of actual data (that is, only let those non-zero transform coefficients participate in computation) to reduce addition and shift operations, they are less efficient than the proposed algorithm.

The reason is that an operation which judges whether a datum is zero needs to be done for each participant datum in the algorithms of the literatures [5,6], while a judging-zero operation only needs to be done for each participant matrix in the proposed algorithm. Note that the number of participant matrices is much less than the number of participant data. The algorithms in the literatures [5,6] need much more judging-zero operations than the proposed algorithm in order to utilize the characteristic of actual data.

TABLE 2. The average numbers of non-zero quantized transform coefficients per non-all-zero 4×4 block in the standard video clips (QP=28)

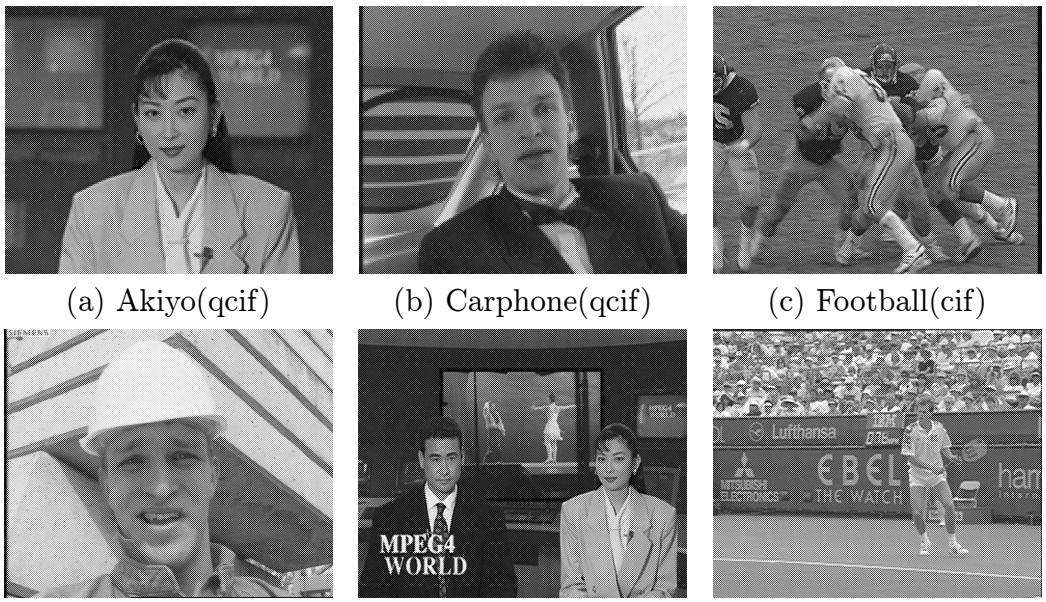
Standard Video clips	Num. of non-all-zero 4×4 blocks	Average num. of nonzero coeffs per non-all-zero 4×4 block
Akiyo(qcif)	916	2.80240
Carphone(qcif)	1268	2.75763
Football(cif)	6453	2.45948
Foreman(cif)	3565	2.49453
News(cif)	3784	2.81950
Stefan(cif)	6417	2.62806

3. **Experiments.** The original JM10.1 Reference Software uses the fast algorithm of [5] for the 2D 4×4 inverse integer transform. In order to demonstrate the efficiency of our new algorithm, we replace the fast algorithm of [5] with our new algorithm. In this experiment, we removed all the all-zero blocks and did the inverse integer transform on the non-all-zero 4×4 blocks. The comparison of computational complexity of two algorithms (see Table 3) shows that the average number of addition operations in the proposed algorithm per non-all-zero 4×4 block is 12.7838; and the average number of shift operations in the proposed algorithm per non-all-zero 4×4 block is 1.69536. Compared to 64 addition operations and 16 shift operations of the fast algorithms in [5,6], the proposed algorithm increases the decoding speed drastically.

TABLE 3. Comparisons of computational complexity for 2-D 4×4 inverse integer transform (QP=28)

Standard Video clips	Proposed fast algorithm		Fast algorithm in [5-6]	
	Average num. of Additions per non-all-zero block	Average num. of shifts per non-all-zero block	Num. of additionsr per non-all-zero block	Num. of shifts per non-all-zero block
Akiyo(qcif)	13.6779	1.81987	64	16
Carphone(qcif)	12.9953	1.80284	64	16
Football(cif)	12.3477	1.49930	64	16
Foreman(cif)	11.5257	1.59523	64	16
News(cif)	13.3153	1.78436	64	16
Stefan(cif)	12.8409	1.67056	64	16

4. **Conclusions.** In actual video coding process, most of the quantized transform coefficients are zero. This paper proposes a new direct fast 2-D 4×4 inverse integer transform algorithm. The algorithm is developed by utilizing the symmetries of basic images and the characteristic of actual data. The proposed algorithm only allows those non-zero terms $(F(u,v) \times \boldsymbol{B}_{uv})$ perform the computation of the inverse integer transform. Therefore, the proposed algorithm can make use of the characteristic of actual data more efficiently in comparison with the algorithms in literatures [5,6]. As a result, it can achieve a higher speed in real-time H.264/AVC video decoding.



(a) Akiyo(qcif) (b) Carphone(qcif) (c) Football(cif)

(d) Foreman(cif) (e) News(cif) (f) Stefan(cif)

FIGURE 1. Standard video clips

Acknowledgment. This work is supported by the national natural science foundations of China (No.60573114 and 60633030), and the national key basic research 973 program of China (2006CB303102). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] Joint video team (JVT) of ISO/IEC MPEG and ITU-T VCEG, *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec.H.264/ISO/IEC 14496-10 AVC)*, JVT-G050, 2003.

[2] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia*, John Wiley & Sons, 2003.

[3] H. T. Yu, F. Pan and Z. P. Li, Content adaptive rate control for H.264, *International Journal of Innovative Computing, Information and Control*, vol.1, no.4, pp.685-700, 2005.

[4] M. Wien, Variable block-size transforms for H.264/AVC, *IEEE Trans. on Circuits and Systems for Video Technology*, vol.13, no.7, pp.604-613, 2003.

[5] H. S. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky, Low-complexity transform and quantization in H.264/AVC, *IEEE Trans. on Circuits and Systems for Video Technology*, vol.13, no.7, pp.598-603, 2003.

[6] C. P. Fan, Fast 2-dimensional 4×4 forward integer transform implementation for H.264/AVC, *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol.53, no.3, pp.174-177, 2006.

[7] Y. Wang, Y. Zhou and H. Yang, Early detection method of all-zero integer transform coefficients, *IEEE Trans. on Consumer Electronics*, vol.50, no.3, pp.923-928, 2004.

[8] R. Kordasiewicz and S. Shirani, Hardware implementation of the optimized transform and quantization blocks of H.264, *Proc. of the Canadian Conf. on Electrical and Computer Engineering*, Niagra Falls, Canada, pp.943-946, 2004.

[9] Y. Zeng, L. Cheng, G. Bi and A. C. Kot, Integer DCTs and fast algorithms, *IEEE Trans. on Signal Process*, vol.49, no.11, pp.2774-2782, 2001.

[10] Y. H. Moon, G. Y. Kim and J. H. Kim, An improved early detection algorithm for all-zero blocks in H.264 video encoding, *IEEE Trans. on Circuits and Systems for Video Technology*, vol.15, no.8, pp.1053-1057, 2005.