Direct Conversions Between DV Format DCT and Ordinary DCT

Neri Merhav
HP Laboratories Israel*
HPL-98-140
August, 1998

digital recording,
DVC format,
8-8 DCT,
2-4-8 DCT

A fast algorithm is developed for direct conversions between the 2-4-8 DCT mode associated with the digital video cassette (DVC) standard and the ordinary 8-8 DCT associated with other formats of compressed video. The algorithm avoids explicit transformations to and from the original pixel domain and thus saves a considerable amount of computations.

Internal Accession Date Only

*Hewlett-Packard Laboratories Israel, Technion City, Haifa 32000, Israel

1 Introduction

DV (formerly DVC) is a new digital recording format being backed by the main manufacturers such as Sony, Philips, Thomson, Hitachi, and Matsushita (Panasonic). It was the first digital recording format in the reach of consumer markets. Its main advantage is that it lends itself easily to video editing operations. DV uses a 5:1 DCT-based compression scheme at a fixed rate of 25Mbps and the signal-to-noise ratio (SNR) is typically around 54dB. Depending on whether the difference between two fields is small or large, the encoder adaptively decides whether to compress picture fields separately (small difference) or combine two fields into a single compression block (large difference) [1, p. 27]. Therefore, in a certain sense, DV coding can be thought of as a standard that lies in the midway between Motion JPEG and MPEG.

Unfortunately, however, the DV format is not related to any other compressed/uncompressed video format. As a first step towards the goal of building an interface between DV and other compressed video formats, such as MPEG-I, MPEG-II, H.261, H.263, etc. (for applications like transcoding), we develop and propose, in this document, a fast algorithm that converts the 2-4-8 DCT of the DV format (which is used for largely different fields) to the ordinary 8-8 DCT which is used by all the above-mentioned video compression standards.

Since the proposed algorithm operates directly in the DCT domain, without explicit transformation to and from the pixel domain, it saves a great deal of the computations without additional memory requirements.

2 Background and Problem Formulation

The DV standard contains two DCT modes, called 8-8 DCT mode and 2-4-8 DCT mode to improve the picture quality after the bit rate reduction. The 8-8 DCT mode should be selected when the difference between two fields is small, whereas the 2-4-8 DCT mode should be selected when the difference between two fields is large. The two DCT modes are defined as follows.

8-8 DCT: The 8-8 DCT is the ordinary, type-II two-dimensional DCT [2] that transforms a block $\{x(n,m)\}_{n,m=0}^{7}$ in the spatial domain into a matrix of frequency components $\{X(k,l)\}_{k,l=0}^{7}$ according to the following equation

$$X_{88}(k,l) = c(k)c(l)\sum_{n=0}^{7}\sum_{m=0}^{7} x(n,m)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi) \tag{1}$$

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

$$x(n,m) = \sum_{k=0}^{7}\sum_{l=0}^{7} c(k)c(l)X_{88}(k,l)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi). \tag{2}$$

In a matrix form, let $\boldsymbol{x} = \{x(n,m)\}_{n,m=0}^{7}$ and $\boldsymbol{X}_{88} = \{X_{88}(k,l)\}_{k,l=0}^{7}$. Define the 8-point DCT operator matrix $C_8 = \{c_8(k,n)\}_{k,n=0}^{7}$, where

$$c_8(k,n) = c(k)\cos(\frac{2n+1}{16}\cdot k\pi). \tag{3}$$

Then,

$$\boldsymbol{X}_{88} = C_8 \boldsymbol{x} C_8^t \tag{4}$$

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

$$\boldsymbol{x} = C_8^{-1} \boldsymbol{X}_{88} C_8^{-t} = C_8^t \boldsymbol{X} C_8 \tag{5}$$

where the second equality follows from the orthonormality of C_8.

2-4-8 DCT: The 2-4-8 DCT combines two fields in one block in the following way. Let us assume that the spatial domain 8×8 block $\boldsymbol{x} = \{x(n,m)\}_{n,m=0}^{7}$ consists of two interlaced fields, where the odd rows (starting to count from zero) are from the first field and the even rows are from the second field. Then, the 2-4-8 DCT is defined as follows. For $k = 0, 1, 2, 3$ and $l = 0, 1, ..., 7$ define

$$X_{248}(k,l) = c(k)c(l) \sum_{n=0}^{3} \sum_{m=0}^{7} [x(2n,m) + x(2n+1,m)] \cos(\frac{2n+1}{8} \cdot k\pi) \cos(\frac{2m+1}{16} \cdot l\pi) \tag{6}$$

and

$$X_{248}(k+4,l) = c(k)c(l) \sum_{n=0}^{3} \sum_{m=0}^{7} [x(2n,m) - x(2n+1,m)] \cos(\frac{2n+1}{8} \cdot k\pi) \cos(\frac{2m+1}{16} \cdot l\pi). \tag{7}$$

The inverse transform is given by

$$\begin{aligned} x(2n,m) = \quad & \frac{1}{2} \sum_{k=0}^{3} \sum_{l=0}^{7} c(k)c(l)[X_{248}(k,l) + X_{248}(k+4,l)] \times \\ & \cos(\frac{2n+1}{8} \cdot k\pi) \cos(\frac{2m+1}{16} \cdot l\pi) \end{aligned} \tag{8}$$

and

$$\begin{aligned} x(2n+1,m) = \quad & \frac{1}{2} \sum_{k=0}^{3} \sum_{l=0}^{7} c(k)c(l)[X_{248}(k,l) - X_{248}(k+4,l)] \times \\ & \cos(\frac{2n+1}{8} \cdot k\pi) \cos(\frac{2m+1}{16} \cdot l\pi), \end{aligned} \tag{9}$$

where $n = 0, 1, 2, 3$ and $m = 0, 1, ..., 7$. In a matrix form, let

$$F = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \tag{10}$$

and

$$C_{24} = \begin{pmatrix} C_4 & 0 \\ 0 & C_4 \end{pmatrix} \tag{11}$$

where $C_4 = \{c_4(k,n)\}_{k,n=0}^{3}$ is the 4-point DCT operator matrix whose entries are given by

$$c_4(k,n) = c(k)\cos(\frac{2n+1}{8} \cdot k\pi), \qquad k,n = 0,1,2,3. \tag{12}$$

We can now rewrite eqs. (8) and (9) as

$$\boldsymbol{X}_{248} = C_{24} F \boldsymbol{x} C_8^t \tag{13}$$

and

$$\boldsymbol{x} = F^{-1} C_{24}^{-1} \boldsymbol{X}_{248} C_8, \tag{14}$$

respectively, where $\boldsymbol{X}_{248} = \{X_{248}(k,l)\}_{k,l=0}^{7}$.

Our goal is to devise fast algorithms that convert from $\boldsymbol{X}_{248}$ to $\boldsymbol{X}_{88}$ and vice versa. Our algorithms will be based on sparse matrix factorizations of C_4 and C_8. The best known factorization of C_8 to sparse matrices corresponds to the Winograd 8-point DCT which was derived from the 16-point Winograd FFT [3] (see also [4]). According to this factorization, C_8 is represented as a product:

$$C_8 = DPB_1B_2MA_1A_2A_3 \tag{15}$$

where D is a diagonal matrix given by

$$D = \mathrm{diag}\{0.3536, 0.2549, 0.2706, 0.3007, 0.3536, 0.4500, 0.6533, 1.2814\}, \tag{16}$$

P is a permutation matrix given by

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{17}$$

and the remaining matrices are defined as follows:

$$B_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix} \tag{18}$$

$$B_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix} \tag{19}$$

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.9239 & 0 & -0.3827 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.3827 & 0 & 0.9239 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{20}$$

$$A_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{21}$$

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{22}$$

$$A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \tag{23}$$

In the next section, we will present a parallel sparse matrix factorization of C_4 that corresponds to the 8-point Winograd FFT. We will then show how to use these factorizations to derive a fast conversion algorithm from 8-8 DCT to 2-4-8 DCT and vice versa.

3 A 4-Point Winograd DCT

Let us concentrate on 1D transforms, where it should be understood that 2D transforms are obtained in a row-column separable fashhion.

The 4-point Winograd DCT will be based on the 8-point Winograd FFT, which works as follows (see also [5, p. 163]). Given an input vector $x = (x_0, x_1, ..., x_7)$, the 8-point DFT $X = (X_0, X_1, ..., X_7)$, is computed in the following way:

1. Compute $t = (t_1, t_2, ..., t_8)$ according to

$$\begin{aligned} t_1 &= x_0 + x_4 \quad t_2 = x_2 + x_6 \quad t_3 = x_1 + x_5 \quad t_4 = x_1 - x_5 \\ t_5 &= x_3 + x_7 \quad t_6 = x_3 - x_7 \quad t_7 = t_1 + t_2 \quad t_8 = t_3 + t_5 \end{aligned}$$

2. Define $u = \pi/4$, $i = \sqrt{-1}$ and compute $m = (m_0, m_1, ..., m_7)$ according to

$$\begin{aligned} m_0 &= t_7 + t_8 \quad m_1 = t_7 - t_8 \\ m_2 &= t_1 - t_2 \quad m_3 = x_0 - x_4 \\ m_4 &= (t_4 - t_6)\cos(u) \quad m_5 = i(t_3 - t_5) \\ m_6 &= i(x_2 - x_6) \quad m_7 = i(t_4 + t_6)\sin(u) \end{aligned}$$

3. Compute $s = (s_1, ..., s_4)$ according to

$$\begin{aligned} s_1 &= m_3 + m_4 \quad s_2 = m_3 - m_4 \\ s_3 &= m_6 + m_7 \quad s_4 = m_6 - m_7 \end{aligned}$$

4. Finally, compute the output DFT according to

$$\begin{aligned} X_0 &= m_0 \quad X_1 = s_1 + s_3 \quad X_2 = m_2 + m_5 \quad X_3 = s_2 - s_4 \\ X_4 &= m_1 \quad X_5 = s_2 + s_4 \quad X_6 = m_2 - m_5 \quad X_7 = s_1 - s_3 \end{aligned}$$

It is well-known (see, e.g., [4]) that an N-point DCT can be obtained from the real part of a $(2N)$-point DFT if the input vector is defined by the symmetric extension

$$(x_0, x_1, ..., x_{N-1}, x_{N-1}, x_{N-2}, ..., x_0).$$

A nice property of the Winograd FFT is that there is complete separation between the real part and the imaginary part. Since the DCT is a real transform, the imaginary part can be therefore ignored altogether.

In our case, $N = 4$. If we use the above algorithm, taking advantage of the facts that (i) $x_i = x_{7-i}$, $i = 0, 1, 2, 3$, and (ii) the imaginary part can be ignored, we end up with a further simplified algorithm for computing the 4-point DCT.

The resulting Winograd DCT algorithm corresponds to the following factorization of C_4:

$$C_4 = D_1 G H L \tag{24}$$

where

$$D_1 = \mathrm{diag}\left\{ \frac{c(i)}{2\cos(\pi i/8)}, i = 0,1,2,3 \right\} \tag{25}$$

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \tag{26}$$

$$H = \begin{pmatrix} 2 & 2 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0.7071 & 0.7071 \end{pmatrix} \tag{27}$$

and

$$L = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}. \tag{28}$$

For later use, we will also define

$$D_2 = \begin{pmatrix} D_1 & 0 \\ 0 & D_1 \end{pmatrix}, \tag{29}$$

$$G_2 = \begin{pmatrix} G & 0 \\ 0 & G \end{pmatrix}, \tag{30}$$

$$H_2 = \begin{pmatrix} H & 0 \\ 0 & H \end{pmatrix}, \tag{31}$$

and

$$L_2 = \begin{pmatrix} L & 0 \\ 0 & L \end{pmatrix}. \tag{32}$$

Clearly, since $C_4 = D_1 G H L$, we also have $C_{24} = D_2 G_2 H_2 L_2$.

4 Derivation of the Algorithm

We are now ready to develop the conversion algorithm. First, observe that according to eqs. (13), (14), and the subsequent derivations, we have

$$\begin{aligned} \boldsymbol{X}_{88} &= C_8 \boldsymbol{x} C_8^t \\ &= C_8 (F^{-1} C_{24}^{-1} \boldsymbol{X}_{248} C_8) C_8^t \\ &= C_8 F^{-1} C_{24}^{-1} \boldsymbol{X}_{248} \\ &= DPB_1 B_2 M A_1 A_2 A_3 F^{-1} L_2^{-1} H_2^{-1} G_2^{-1} D_2^{-1} \boldsymbol{X}_{248}. \end{aligned} \tag{33}$$

Precomputation of

$$R \triangleq MA_1A_2A_3F^{-1}L_2^{-1}H_2^{-1} \tag{34}$$

results in a fairly sparse matrix given by

$$R = \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2a \\ 0 & a & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & a \\ 0 & 0 & -b & b & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & -a/2 & 0 & 0 \\ 0 & 0 & c & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.125 & 0.25 & 0 & 0 \end{pmatrix}, \tag{35}$$

where $a = 0.7071$, $b = 1.3066$, and $c = 0.5412$. The proposed conversion algorithm calculates $\boldsymbol{X}_{88}$ from $\boldsymbol{X}_{248}$ according to

$$\boldsymbol{X}_{88} = DPB_1B_2RG_2^{-1}D_2^{-1}\boldsymbol{X}_{248}. \tag{36}$$

where

$$G_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & -0.5 \end{pmatrix}. \tag{37}$$

This means that the input 2-4-8 DCT is first premultiplied by D_2^{-1}, then the result is pre-multiplied by G_2^{-1}, and so on.

Assuming that the multiplication by D_2^{-1} can be absorbed in the dequantization of the 2-4-8 DCT coefficients (as well as compensation for the weight factors [1, p. 28, subsection 7.5.2]), and that multiplication by D can be absrobed in the re-quantization of the 8-8 DCT coefficients, the only phase, in this algorithm, that contains nontrivial multiplications is the pre-multiplication by the matrix R. This is because multiplications by powers of two are implementable by simple shifts.

Given a column vector $u = (u_1, ..., u_8)^t$, the vector $v = (v_1, ..., v_8)^t$ defined as $v = Ru$ can be efficiently computed by the following steps:

$$\begin{aligned} v_1 &= u_1/2 \\ w &= (2a)u_8 \\ v_2 &= 2u_7 - w \\ v_3 &= au_2 \\ v_4 &= (u_2 + w)/2 \end{aligned}$$

$$
\begin{aligned}
v_5 &= b(u_4 - u_3) \\
v_6 &= (u_4 - au_6)/2 \\
v_7 &= c(u_3 + u_4) \\
v_8 &= (u_3 + (u_6 + u_5/2)/2)/2
\end{aligned}
$$

Thus, pre-multiplication by R can implemented by 5 multiplications, 3 shift&adds, 4 additions, and 4 shifts. Since the matrix P is computationally costless, B_1 and B_2 require 4 additions per vector each one, and G_2^{-1} is associated with 4 additions and 4 shifts per vector, the total complexity associated with this algorithm is 5 multiplications, 3 shift&adds, 16 additions, and 8 shifts per each column vector. For the sake of comparison, the direct approach (of explicitely undoing the column 2-4-8 DCT and doing the 8-8 DCT instead) would require 7 multiplications, and 55 additions/shift&adds. This means that if, for example, each multiplication takes 3 cycles on the average, then about 50% of the computations have been saved. If, on the other hand, the processor performs multiplication in one cycle, the saving factor is even higher. For a complete DCT block, all the above numbers should be multiplied by 8.

As a final remark, it should be pointed out that since both C_8 and C_{24} are orthonormal, namely, $C_8^{-1} = C_8^t$ and $C_{24}^{-1} = C_{24}^t$, there are three more ways to write the equation relating $\boldsymbol{X}_{248}$ to $\boldsymbol{X}_{88}$ (eq. (33)). These are:

$$\boldsymbol{X}_{88} = DPB_1B_2MA_1A_2A_3F^{-1}L_2^tH_2^tG_2^tD_2\boldsymbol{X}_{248}, \tag{38}$$

$$\boldsymbol{X}_{88} = D^{-1}P^{-t}B_1^{-t}B_2^{-t}M^{-t}A_1^{-t}A_2^{-t}A_3^{-t}F^{-1}L_2^tH_2^tG_2^tD_2\boldsymbol{X}_{248}, \tag{39}$$

and

$$\boldsymbol{X}_{88} = D^{-1}P^{-t}B_1^{-t}B_2^{-t}M^{-t}A_1^{-t}A_2^{-t}A_3^{-t}F^{-1}L_2^{-1}H_2^{-1}G_2^{-1}D_2^{-1}\boldsymbol{X}_{248}. \tag{40}$$

Accordingly, the matrix R would then be redefined as

$$MA_1A_2A_3F^{-1}L_2^tH_2^t,$$

or

$$M^{-t}A_1^{-t}A_2^{-t}A_3^{-t}F^{-1}L_2^tH_2^t,$$

or

$$M^{-t}A_1^{-t}A_2^{-t}A_3^{-t}F^{-1}L_2^{-1}H_2^{-1},$$

respectively. However, the first representation that we have used (eq. (33)) yields the sparsest R.

5 The Inverse Conversion

In view of the last comment in the previous section, there are also four ways to write the inverse transformation from $\boldsymbol{X}_{88}$ to $\boldsymbol{X}_{248}$:

$$\boldsymbol{X}_{248} = D_2G_2H_2L_2FA_3^{-1}A_2^{-1}A_1^{-1}M^{-1}B_2^{-1}B_1^{-1}P^{-1}D^{-1}\boldsymbol{X}_{88}, \tag{41}$$

$$\boldsymbol{X}_{248} = D_2G_2H_2L_2FA_3^tA_2^tA_1^tM^tB_2^tB_1^tP^tD\boldsymbol{X}_{88}, \tag{42}$$

$$\boldsymbol{X}_{248} = D_2^{-1}G_2^{-t}H_2^{-t}L_2^{-t}FA_3^{-1}A_2^{-1}A_1^{-1}M^{-1}B_2^{-1}B_1^{-1}P^{-1}D^{-1}\boldsymbol{X}_{88}, \tag{43}$$

and

$$\boldsymbol{X}_{248} = D_2^{-1} G_2^{-t} H_2^{-t} L_2^{-t} F A_3^t A_2^t A_1^t M^t B_2^t B_1^t P^t D \boldsymbol{X}_{88}, \tag{44}$$

Correspondingly, we define $\tilde{R}$ as either

$$H_2 L_2 F A_3^{-1} A_2^{-1} A_1^{-1} M^{-1},$$

or

$$H_2 L_2 F A_3^t A_2^t A_1^t M^t,$$

or

$$H_2^{-t} L_2^{-t} F A_3^{-1} A_2^{-1} A_1^{-1} M^{-1},$$

or

$$H_2^{-t} L_2^{-t} F A_3^t A_2^t A_1^t M^t.$$

The latter possbility turns out to be best in terms of the computational complexity of the associated algorithm.

The inverse algorithm then computes $\boldsymbol{X}_{248}$ from $\boldsymbol{X}_{88}$ according to

$$\boldsymbol{X}_{248} = D_2^{-1} G_2^{-t} \tilde{R} B_2^t B_1^t P^t D \boldsymbol{X}_{88} \tag{45}$$

where

$$\tilde{R} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2a & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2b & 0 & 2c & 1 \\ 0 & 0 & 0 & 0 & 2b & 1 & 2c & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 \\ 0 & 0 & 0 & 0 & 0 & -a & 0 & 0.5 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4a & 0 & 2a & 0 & 0 & 0 & 0 \end{pmatrix} \tag{46}$$

where a, b, and c are as in eq. (35). The computational complexity of this algorithm is approximately the same as that of the forward algorithm because the computation of $v = \tilde{R}u$ can be done in 5 multiplications, 5 additions, 2 shift&adds, and 2 shifts, using the following procedure:

$$\begin{aligned} v_1 &= u_1 \\ v_2 &= (2a)u_3 + u_4 \\ w_1 &= (2b)u_5 \\ w_2 &= (2c)u_7 \\ v_3 &= -w_1 + w_2 + u_8 \\ v_4 &= w_1 + w_2 + u_6 \\ v_5 &= u_8/4 \\ v_6 &= -au_6 + u_8/2 \\ v_7 &= 4u_2 \\ v_8 &= (2a)(u_4 - 2u_2). \end{aligned}$$

6 References

[1] HD Digital VCR Conference, *Specifications of Consumer-Use Digital VCRs Using 6.3mm Magnetic Tape*, Part 2, December 1994.

[2] K. R. Rao, and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press 1990.

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

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

[5] H. S. Silverman, "An introduction to programming the Winograd Fourier transform algorithm," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP–25, no. 2, pp. 152–165, April 1977.