A Transform Domain Approach To Spatial Domain Image Scaling

Neri Merhav* Vasudev Bhaskaran†

*N. Merhav(merhav@hp.technion.ac.il) is with the HP Israel Science Center, Technion City, Haifa 32000, Israel.

†V. Bhaskaran(bhaskara@hpl.hp.com) is with the Visual Computing Department, HP Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A.

1. Introduction

Many video compression methods, like MPEG and H.261, use transform domain techniques, in particular, the discrete cosine transform (DCT). Certain applications require real time manipulation of digital video in order to implement image composition and special effects, e.g., scaling (zooming in or out), modifying contrast and brightness, translating, filtering, masking, rotation, motion compensation, etc. There are two major difficulties encountered in this class of tasks: the computational complexity of image compression and decompression, and the high rates of the data to be manipulated. These difficulties rule out the possibility of running, on currently existing workstations, the traditional algorithms that first decompress the data, then perform one of these manipulations in the decompressed domain, and finally, compress again if necessary. For this reason there has been a great effort in recent years to develop fast algorithms that perform these tasks directly in the transform domain (see e.g., [2], [3], [4] and references therein) and thereby avoid the need of decompression, or at least its computational bottle neck - the inverse DCT (IDCT) which requires 38.7% of the execution time on a PA-RISC workstation [1].

As an example, consider a video conferencing session of several parties, where each one of them can see everybody else in a separate window on his screen. Every user would like to have the flexibility to resize windows, move them from one location on the screen to another, and so on. Since each workstation is capable of handling one video stream only, the server must compose the streams from all parties to a single stream whose architecture depends on the user's requests. If one user wishes, say, to scale down by a factor of 2 a window corresponding to another user and move it to a different place on the screen, this might affect the entire image. The traditional and expensive approach would be that all compressed video streams are first decompressed at the server, then the desired change is translated into a suitable arithmetic operation on the decompressed video streams with the appropriate composition into a single stream, and finally, the composite stream is compressed again and sent to the user. A great deal of the computational load is in the DCT and IDCT operations and this drives us to seek fast algorithms that perform the desired modification directly in the DCT domain.

In this work, we focus on speeding up the operations of scaling down [1] by factors of 2,3 and 4, as compared to the traditional approach. Since the scaling transformation is linear, the overall effect in the DCT domain is linear as well and hence the basic operation can be represented as multiplication by a fixed matrix. Fast multiplication by this matrix is possible if it can be factorized into a product of sparse matrices whose entries are mostly 0, 1 and -1. We will demonstrate that this can be done efficiently by taking advantage of the factorizations of the DCT and IDCT operation matrices that correspond to the fast 8-point DCT/IDCT due to Arai, Agui, and Nakajima [5] (see also [6]).

[1] In this document, the term "scaling down by a factor x" means that if the input image resolution is $M \times N$ pixels, then after scaling it is $(M/x) \times (N/x)$.

The resulting schemes for scaling save about 37% of the computations for a scaling factor of 2, 39% for scaling by 3, and 50% for a factor of 4. Here the term "computation" corresponds to the basic arithmetic operation of the PA-RISC processor which is either "shift", "add", or "shift and add" (SH1ADD, SH2ADD, and SH3ADD). These are 'worst-case' estimates in the sense that nothing is assumed on sparseness in the DCT domain. Typically, in a considerably large percentage of the DCT blocks all the DCT coefficients are zero except for the upper left 4×4 quarter that corresponds to low frequencies in both vertical and horizontal directions. If this fact is taken into account, then computation reductions can reach about 80%.

Another advantage of the proposed method is that it improves the precision of the computations as compared to the traditional approach. The reason for this will become apparent later on when we describe the method in detail. The degree of improvement in precision varies between 1.5-3dB.

2. Preliminaries and Problem Description

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

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

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

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

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

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

Then,

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

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

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

where the second equality follows from the orthonormality of S.

Now, suppose we are given four adjacent 8×8 spatial domain data blocks $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, and $\boldsymbol{x}_4$ that together form a 16×16 square, where $\boldsymbol{x}_1$ corresponds to northwest, $\boldsymbol{x}_2$ to northeast, $\boldsymbol{x}_3$ to southwest and $\boldsymbol{x}_4$ to southeast. Scaling down (decimation) by a factor of 2 in each dimension means that every nonoverlapping group of 4 pixels forming a small 2×2 block is replaced by one pixel whose intensity is the average of the 4 original pixels. As a result, the original blocks $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$ are replaced by a single 8×8 output block $\boldsymbol{x}$ corresponding to the decimation of $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$. Our task is to calculate efficiently $\boldsymbol{X}$, the DCT of $\boldsymbol{x}$, directly from the given DCT's of the original blocks $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$, and $\boldsymbol{X}_4$.

The problems of scaling down by a factor of 3 and 4 are defined similarly, where the number of input blocks is 9 and 16, respectively, and the blocks $\boldsymbol{x}_1, \boldsymbol{x}_2, ...$ are indexed in a raster scan order. Similarly, as in the case of a factor of 2, here every nonoverlapping group of 3×3 pixels for the case of scaling by 3, and 4×4 pixels for the case of scaling by 4, are replaced by their average, and the output is one DCT block that will be denoted always by $\boldsymbol{X}$. Note, that the case of a factor 3 is somewhat more involved because some of the 3×3 groups to be averaged are not entirely within the same 8×8 block.

3. The Basic Idea

For the sake of simplicity, let us confine attention first to the one dimensional case and decimation by 2. The two dimensional case will be a repeated application for every row and then for every column of each block. In this case, we are given two 8-dimensional vectors $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ of DCT coeficients corresponding to adjacent time domain vectors of length 8, $\boldsymbol{x}_1 = S^{-1}\boldsymbol{X}_1$ and $\boldsymbol{x}_2 = S^{-1}\boldsymbol{X}_2$, and we wish to calculate $\boldsymbol{X}$, the DCT of the 8-dimensional vector $\boldsymbol{x}$, whose each component is the average of the two appropriate adjacent components in $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$.

It is convenient to describe the decimation operation in a matrix form as follows.

$$\boldsymbol{x} = \frac{1}{2}(Q_1\boldsymbol{x}_1 + Q_2\boldsymbol{x}_2) \tag{6}$$

where

$$Q_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and

$$Q_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Therefore,

$$\boldsymbol{X} = \frac{1}{2}(SQ_1S^{-1}\boldsymbol{X}_1 + SQ_2S^{-1}\boldsymbol{X}_2). \tag{7}$$

We shall now focus on efficient factorizations of the matrices $U_1 = SQ_1S^{-1}$ and $U_2 = SQ_2S^{-1}$. To this end, we shall use a factorization of S that corresponds to the fastest existing algorithm for 8-point DCT due to Arai, Agui, and Nakajima [5] (see also [6]). According to this factorization S is represented as follows.

$$S = DPB_1B_2MA_1A_2A_3 \tag{8}$$

where D is a diagonal matrix given by

$$D = \mathrm{diag}\{0.3536, 0.2549, 0.2706, 0.3007, 0.3536, 0.4500, 0.6533, 1.2814\}, \tag{9}$$

P is a permutation matrix given by

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

and the remaining matrices are defined as follows:

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

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

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

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

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

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

Thus, for $i = 1, 2$ we have

$$U_i = SQ_iS^{-1} = DPB_1B_2MA_1A_2A_3Q_iA_3^{-1}A_2^{-1}A_1^{-1}M^{-1}B_2^{-1}B_1^{-1}P^{-1}D^{-1} \tag{10}$$

The proposed decimation algorithm is based on the observation that the products

$$F_i = MA_1A_2A_3Q_iA_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \qquad i = 1, 2 \tag{11}$$

are fairly sparse matrices, and most of the corresponding elements are the same, sometimes with a different sign. This means that their sum $F_+ = F_1 + F_2$ and their difference $F_- = F_1 - F_2$ are even sparser. These matrices are given as follows.

$$F_+ = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.8281 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.7071 & 0 & 0.7071 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2930 & 0 & -0.7071 & 0 \\ 0 & 0 & 0 & 0 & -0.3750 & 0 & 0.9219 & 0 \end{pmatrix}$$

$$F_- = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7617 & 0 & 1.8477 & 0 \\ 0 & 0 & 0 & 0 & -0.7617 & 0 & 1.8477 & 0 \\ 0.5391 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.7071 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1.2969 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5000 & 0 & 0.7071 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Finally, we use the following simple relation:

$$\begin{aligned} \boldsymbol{X} &= \frac{1}{2}(U_1\boldsymbol{X}_1 + U_2\boldsymbol{X}_2) \\ &= \frac{1}{4}[(U_1 + U_2)(\boldsymbol{X}_1 + \boldsymbol{X}_2) + (U_1 - U_2)(\boldsymbol{X}_1 - \boldsymbol{X}_2)] \\ &= \frac{1}{4}DPB_1B_2[F_+B_2^{-1}B_1^{-1}P^{-1}D^{-1}(\boldsymbol{X}_1 + \boldsymbol{X}_2) + \\ &\quad F_-B_2^{-1}B_1^{-1}P^{-1}D^{-1}(\boldsymbol{X}_1 - \boldsymbol{X}_2)] \end{aligned} \tag{12}$$

Let us count the number of basic arithmetic operations on the PA-RISC processor that are needed to implement the right-most side of (12) and compare it to the spatial domain approach. As explained in the introduction, here the term “operation” corresponds to the elementary arithmetic computation of the PA-RISC processor which is either “shift”, “add”, or “shift and add” (SH1ADD, SH2ADD, and SH3ADD). For example, the computation

$z = 1.375x + 1.125y$ is implemented as follows: First, we compute $u = x + 0.5x$ (SH1ADD), then $v = x + 0.25u$ (SH2ADD), afterwards $w = v + y$ (ADD), and finally, $z = w + 0.125y$ (SH3ADD). Thus, overall 4 basic operations are needed.

When counting the operations, we wil use the fact that multiplications by D and D^{-1} can be ignored because these can be absorbed in the MPEG quantizer and dequantizer, respectively. The matrices P and P^{-1} cause only changes in the order of the components so they can be ignored as well. Thus we are left with the following (in parentheses, we detail also the number of additions/subtractions and the number of nontrivial multiplications):

Creating $\boldsymbol{X}_1 + \boldsymbol{X}_2$ and $\boldsymbol{X}_1 - \boldsymbol{X}_2$: 16 operations (16 additions).
Two multiplications by B_1^{-1}: 8 operations (8 additions).
Two multiplications by B_2^{-1}: 8 operations (8 additions).
Multiplication by F_+: 23 operations (5 multiplications + 5 additions).
Multiplication by F_-: 28 operations (6 multiplications + 4 additions).
Adding the products: 8 operations (8 additions).
Multiplication by B_2: 4 operations (4 additions).
Multiplication by B_1: 4 operations (4 additions).
Total: 115 operations (11 multiplications + 57 additions).

In the spatial domain approach, on the other hand, we have the following:

Two IDCT's: 114 operations (10 multiplications + 60 additions).
Decimation in time domain: 8 operations (8 additions).
DCT: 42 operations (5 multiplications + 30 additions).
Total: 161 operations (15 multiplications + 98 additions).

It turns out, as can be seen, that the proposed approach saves about 30% of the operations in the one-dimensional case. We shall see later on, in the two-dimensional case, that by using the same ideas, we obtain even greater reductions in complexity.

As a byproduct of the proposed approach, it should be noted that arithmetic precision is gained. Since in the direct approach, we actually multiply by each one of the matrices on right hand side of (11) one at a time, then roundoff errors, associated with finite word length representations of the elements of these matrices, accumulate in each step. On the other hand, in the proposed approach, we can precompute F_i once and for all to any desired degree of precision, and then round off each element of these matrices to the allowed precision. The latter has, of course, better precision. More details will be provided in Section 5.

4. The Two Dimensional Case

Let us now return to the two dimensional case. A 2D-DCT is just 1D-DCT applied to every column and every row of the spatial domain block. Therefore, the very same ideas can be applied to the two dimensional case as well. In this section we describe in detail the computation schemes for scaling down by a factor of 2, 3, and 4.

4.1 Scaling by 2

Similarly as in the one dimensional case, we have in the spatial domain

$$\boldsymbol{x} = \frac{1}{4}(Q_1 \boldsymbol{x}_1 Q_1^t + Q_1 \boldsymbol{x}_2 Q_2^t + Q_2 \boldsymbol{x}_3 Q_1^t + Q_2 \boldsymbol{x}_4 Q_2^t) \tag{13}$$

and therefore, in the frequency domain,

$$\boldsymbol{X} = \frac{1}{4}(U_1 \boldsymbol{X}_1 U_1^t + U_1 \boldsymbol{X}_2 U_2^t + U_2 \boldsymbol{X}_3 U_1^t + U_2 \boldsymbol{X}_4 U_2^t) \tag{14}$$

Again, we would like to express the right-hand side of (14) in terms of $U_+ = U_1 + U_2 = DPB_1 B_2 F_+ B_2^{-1} B_1^{-1} P^{-1} D^{-1}$ and $U_- = U_1 + U_2 = DPB_1 B_2 F_- B_2^{-1} B_1^{-1} P^{-1} D^{-1}$. To this end, let us define

$$\boldsymbol{X}_{+++} = \boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3 + \boldsymbol{X}_4, \tag{15}$$

$$\boldsymbol{X}_{+--} = \boldsymbol{X}_1 + \boldsymbol{X}_2 - \boldsymbol{X}_3 - \boldsymbol{X}_4, \tag{16}$$

$$\boldsymbol{X}_{-+-} = \boldsymbol{X}_1 - \boldsymbol{X}_2 + \boldsymbol{X}_3 - \boldsymbol{X}_4, \tag{17}$$

and

$$\boldsymbol{X}_{--+} = \boldsymbol{X}_1 - \boldsymbol{X}_2 - \boldsymbol{X}_3 + \boldsymbol{X}_4. \tag{18}$$

Note that to create all these linear combinations, we need only 8 (and not 12) additions/subtractions per frequency component: We first compute $\boldsymbol{X}_1 \pm \boldsymbol{X}_2$ and $\boldsymbol{X}_3 \pm \boldsymbol{X}_4$ and then $(\boldsymbol{X}_1 + \boldsymbol{X}_2) \pm (\boldsymbol{X}_3 + \boldsymbol{X}_4)$ and $(\boldsymbol{X}_1 - \boldsymbol{X}_2) \pm (\boldsymbol{X}_3 - \boldsymbol{X}_4)$. Now, eq. (14) can be rewritten as

$$\begin{aligned}
\boldsymbol{X} &= \frac{1}{16}(U_+ \boldsymbol{X}_{+++} U_+^t + U_- \boldsymbol{X}_{+--} U_+^t + U_+ \boldsymbol{X}_{-+-} U_-^t + U_- \boldsymbol{X}_{--+} U_-^t) \\
&= \frac{1}{16} DPB_1 B_2 \cdot \\
&\quad [(F_+ B_2^{-1} B_1^{-1} P^{-1} D^{-1} \boldsymbol{X}_{+++} + F_- B_2^{-1} B_1^{-1} P^{-1} D^{-1} \boldsymbol{X}_{+--}) B_1^{-t} B_2^{-t} F_+^t + \\
&\quad (F_+ B_2^{-1} B_1^{-1} P^{-1} D^{-1} \boldsymbol{X}_{-+-} + F_- B_2^{-1} B_1^{-1} P^{-1} D^{-1} \boldsymbol{X}_{--+}) B_1^{-t} B_2^{-t} F_-^t] \cdot \\
&\quad B_2^t B_1^t P^t D^t.
\end{aligned} \tag{19}$$

If we count the the number of operations associated with the implementation of the right-most side of eq. (19) (similarly as in the previous section), we find that the total is 2824. The traditional approach, on the other hand, requires 4512 operations. This means that 37.4% of the operations are saved.

Additional savings in computations can be made by taking advantage of the fact that in typical images most of the DCT blocks $\boldsymbol{X}_i$ have only a few nonzero coefficients, normally, the low frequency coefficients. A reasonable possibility might be to use a mechanism that operates in two steps. In the first step, DCT blocks are classified as being *lowpass* or *non-lowpass*, where the former is defined as a block where, say, only the upper left 4×4 subblock is nonzero. The second step uses either the computation scheme described above for nonlowpass blocks, or a faster scheme that utilizes the lowpass assumption for the precomputation of the above matrix multiplications. It turns out that if $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ are all lowpass blocks, then the reduction in computations is about 80%. The same idea is of course applicable to scalings by 4 and by 3 described below.

4.2 Scaling by 4

One approach for scaling by 4, is to scale twice by 2. However, it turns out that by using the same methods, we can develop a more efficient scheme for scaling directly by 4.

Scaling by 4, involves the following decimation matrices:

$$Q_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q_4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q_5 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q_6 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

and

$$\begin{aligned} \boldsymbol{x} = & \frac{1}{16}(Q_3\boldsymbol{x}_1Q_3^t + Q_3\boldsymbol{x}_2Q_4^t + Q_3\boldsymbol{x}_3Q_5^t + Q_3\boldsymbol{x}_4Q_6^t + \\ & Q_4\boldsymbol{x}_5Q_3^t + Q_4\boldsymbol{x}_6Q_4^t + Q_4\boldsymbol{x}_7Q_5^t + Q_4\boldsymbol{x}_8Q_6^t + \\ & Q_5\boldsymbol{x}_9Q_3^t + Q_5\boldsymbol{x}_{10}Q_4^t + Q_5\boldsymbol{x}_{11}Q_5^t + Q_5\boldsymbol{x}_{12}Q_6^t + \\ & Q_6\boldsymbol{x}_{13}Q_3^t + Q_6\boldsymbol{x}_{14}Q_4^t + Q_6\boldsymbol{x}_{15}Q_5^t + Q_6\boldsymbol{x}_{16}Q_6^t) \end{aligned} \tag{20}$$

Next, define

$$H_1 = MA_1A_2A_3Q_3A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{21}$$
$$H_2 = MA_1A_2A_3Q_4A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{22}$$
$$H_3 = MA_1A_2A_3Q_5A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{23}$$
$$H_4 = MA_1A_2A_3Q_6A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{24}$$

and

$$H_{+++} = H_1 + H_2 + H_3 + H_4 \tag{25}$$
$$H_{--+} = H_1 - H_2 - H_3 + H_4 \tag{26}$$
$$H_{-+-} = H_1 - H_2 + H_3 - H_4 \tag{27}$$
$$H_{+--} = H_1 + H_2 - H_3 - H_4 \tag{28}$$

We shall also define the following linear combinations on the input data

$$\boldsymbol{X}_i^{+++} = \boldsymbol{X}_i + \boldsymbol{X}_{i+4} + \boldsymbol{X}_{i+8} + \boldsymbol{X}_{i+12} \tag{29}$$
$$\boldsymbol{X}_i^{--+} = \boldsymbol{X}_i - \boldsymbol{X}_{i+4} - \boldsymbol{X}_{i+8} + \boldsymbol{X}_{i+12} \tag{30}$$
$$\boldsymbol{X}_i^{-+-} = \boldsymbol{X}_i - \boldsymbol{X}_{i+4} + \boldsymbol{X}_{i+8} - \boldsymbol{X}_{i+12} \tag{31}$$
$$\boldsymbol{X}_i^{+--} = \boldsymbol{X}_i + \boldsymbol{X}_{i+4} - \boldsymbol{X}_{i+8} - \boldsymbol{X}_{i+12} \tag{32}$$

where $i = 1, 2, 3, 4$,

$$\boldsymbol{X}_{+++}^{+++} = \boldsymbol{X}_1^{+++} + \boldsymbol{X}_2^{+++} + \boldsymbol{X}_3^{+++} + \boldsymbol{X}_4^{+++} \tag{33}$$
$$\boldsymbol{X}_{--+}^{+++} = \boldsymbol{X}_1^{+++} - \boldsymbol{X}_2^{+++} - \boldsymbol{X}_3^{+++} + \boldsymbol{X}_4^{+++} \tag{34}$$
$$\boldsymbol{X}_{-+-}^{+++} = \boldsymbol{X}_1^{+++} - \boldsymbol{X}_2^{+++} + \boldsymbol{X}_3^{+++} - \boldsymbol{X}_4^{+++} \tag{35}$$

$$\boldsymbol{X}_{+--}^{+++} = \boldsymbol{X}_1^{+++} + \boldsymbol{X}_2^{+++} - \boldsymbol{X}_3^{+++} - \boldsymbol{X}_4^{+++} \tag{36}$$

and similar definitions for the superscripts are $+--$, $-+-$, and $--+$. To create all these combinations we need 64 additions/subtractions per frequency component. Now, similarly as in (19):

$$
\begin{aligned}
\boldsymbol{X} \;=\; & \frac{1}{256} D P B_1 B_2 [\\
& H_{+++} B_2^{-1} B_1^{-1} P^{-1} D^{-1} (\\
& \boldsymbol{X}^{+++}_{+++} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+++} + \boldsymbol{X}^{+++}_{+--} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+--} + \\
& \boldsymbol{X}^{+++}_{-+-} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{-+-} + \boldsymbol{X}^{+++}_{--+} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{--+}) + \\
& H_{+--} B_2^{-1} B_1^{-1} P^{-1} D^{-1} (\\
& \boldsymbol{X}^{+--}_{+++} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+++} + \boldsymbol{X}^{+--}_{+--} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+--} + \\
& \boldsymbol{X}^{+--}_{-+-} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{-+-} + \boldsymbol{X}^{+--}_{--+} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{--+}) + \\
& H_{-+-} B_2^{-1} B_1^{-1} P^{-1} D^{-1} (\\
& \boldsymbol{X}^{-+-}_{+++} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+++} + \boldsymbol{X}^{-+-}_{+--} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+--} + \\
& \boldsymbol{X}^{-+-}_{-+-} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{-+-} + \boldsymbol{X}^{-+-}_{--+} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{--+}) + \\
& H_{--+} B_2^{-1} B_1^{-1} P^{-1} D^{-1} (\\
& \boldsymbol{X}^{--+}_{+++} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+++} + \boldsymbol{X}^{--+}_{+--} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{+--} + \\
& \boldsymbol{X}^{--+}_{-+-} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{-+-} + \boldsymbol{X}^{--+}_{--+} D^{-t} P^{-t} B_1^{-t} B_2^{-t} H^t_{--+})] \cdot \\
& B_2^t B_1^t P^t D^t
\end{aligned} \tag{37}
$$

The number of operations in implementing this formula is 8224 as compared to 16224 in the traditional approach.

4.3 Scaling by 3

The factor of 3 is more problematic and less elegant than the factors of 2 and 4 because some of the 3×3 blocks to be averaged are not entirely within one 8×8 DCT block. In this case we have three types of decimation matrices:

$$
Q_7 = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

$$Q_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q_9 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

and

$$\begin{aligned} \boldsymbol{x} &= \frac{1}{9}(Q_7\boldsymbol{x}_1Q_7^t + Q_7\boldsymbol{x}_2Q_8^t + Q_7\boldsymbol{x}_3Q_9^t \\ &\quad Q_8\boldsymbol{x}_4Q_7^t + Q_8\boldsymbol{x}_5Q_8^t + Q_8\boldsymbol{x}_6Q_9^t \\ &\quad Q_9\boldsymbol{x}_7Q_7^t + Q_9\boldsymbol{x}_8Q_8^t + Q_9\boldsymbol{x}_9Q_9^t) \end{aligned} \tag{38}$$

Similarly as in the cases of a factor of 2 and 4, we now define

$$T_1 = MA_1A_2A_3Q_7A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{39}$$

$$T_2 = MA_1A_2A_3Q_8A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{40}$$

$$T_3 = MA_1A_2A_3Q_9A_3^{-1}A_2^{-1}A_1^{-1}M^{-1} \tag{41}$$

The computation scheme is based on the fact that the matrices

$$T = T_1 + T_2 + T_3, \tag{42}$$

$$T_+ = T_1 + T_3, \tag{43}$$

and

$$T_- = (T_1 - T_3)/2 \tag{44}$$

are relatively sparse, and on the identity

$$T_1\boldsymbol{X} + T_2\boldsymbol{Y} + T_3\boldsymbol{Z} = T\boldsymbol{Y} + T_+(\frac{\boldsymbol{X}+\boldsymbol{Z}}{2} - \boldsymbol{Y}) + T_-(\boldsymbol{X}-\boldsymbol{Z}). \tag{45}$$

If we transform eq. (38) to the DCT domain and express the fixed matrices in terms of T, T_+, and T_-, using eq. (45), we get, after some algebraic manipulations, the following computation formula:

$$\begin{aligned}
\boldsymbol{X} \;=\; & DPB_1B_2[TB_2^{-1}B_1^{-1}P^{-1}D^{-1} \\
& (\boldsymbol{X}'_1D^{-t}P^{-t}B_1^{-t}B_2^{-t}T^t + \boldsymbol{X}'_2D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_+^t + \boldsymbol{X}'_3D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_-^t) + \\
& T_+B_2^{-1}B_1^{-1}P^{-1}D^{-1} \\
& (\boldsymbol{X}'_4D^{-t}P^{-t}B_1^{-t}B_2^{-t}T^t + \boldsymbol{X}'_5D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_+^t + \boldsymbol{X}'_6D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_-^t) + \\
& T_-B_2^{-1}B_1^{-1}P^{-1}D^{-1} \\
& (\boldsymbol{X}'_7D^{-t}P^{-t}B_1^{-t}B_2^{-t}T^t + \boldsymbol{X}'_8D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_+^t + \boldsymbol{X}'_9D^{-t}P^{-t}B_1^{-t}B_2^{-t}T_-^t)] \cdot \\
& B_2^tB_1^tP^tD^t
\end{aligned} \tag{46}$$

where

$$\boldsymbol{X}'_1 = \boldsymbol{X}_5 \tag{47}$$

$$\boldsymbol{X}'_2 = \frac{1}{2}(\boldsymbol{X}_4 + \boldsymbol{X}_6) - \boldsymbol{X}_5 \tag{48}$$

$$\boldsymbol{X}'_3 = \boldsymbol{X}_4 - \boldsymbol{X}_6 \tag{49}$$

$$\boldsymbol{X}'_4 = \frac{1}{2}(\boldsymbol{X}_2 + \boldsymbol{X}_8) - \boldsymbol{X}_5 \tag{50}$$

$$\boldsymbol{X}'_5 = \frac{1}{2}\left[(\frac{\boldsymbol{X}_1 + \boldsymbol{X}_7}{2} - \boldsymbol{X}_4) + (\frac{\boldsymbol{X}_3 + \boldsymbol{X}_9}{2} - \boldsymbol{X}_6)\right] - \boldsymbol{X}'_4 \tag{51}$$

$$\boldsymbol{X}'_6 = (\frac{\boldsymbol{X}_1 + \boldsymbol{X}_7}{2} - \boldsymbol{X}_4) - (\frac{\boldsymbol{X}_3 + \boldsymbol{X}_9}{2} - \boldsymbol{X}_6) \tag{52}$$

$$\boldsymbol{X}'_7 = \boldsymbol{X}_2 - \boldsymbol{X}_8 \tag{53}$$

$$\boldsymbol{X}'_8 = \frac{1}{2}(\boldsymbol{X}_1 - \boldsymbol{X}_7 + \boldsymbol{X}_3 - \boldsymbol{X}_9) - \boldsymbol{X}'_7 \tag{54}$$

$$\boldsymbol{X}'_9 = \boldsymbol{X}_1 - \boldsymbol{X}_7 - \boldsymbol{X}_3 + \boldsymbol{X}_9 \tag{55}$$

The transformation from $\{\boldsymbol{X}_1, ..., \boldsymbol{X}_9\}$ to $\{\boldsymbol{X}'_1, ..., \boldsymbol{X}'_9\}$ can be done in 18 operations per frequency component. The total number of operations associated with the implementation of eq. (46) is 5728. On the other hand, the number of operations associated with the traditional spatial domain approach is 9392, that is, the reduction in the number of computations is about 39%.

5. Arithmetic Precision

As explained in the last paragraph of Section 3, the proposed computation scheme provides better arithmetic accuracy than the standard approach. To demonstrate this fact we have tested both schemes for the case of scaling by 2, where each element in each of the above defined fixed matrices is represented by 8 bits.

In the first experiment, we have the chosen the elements of $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$ as statistically independent random integers uniformly distributed in the set $\{0, 1, ..., 255\}$. We first computed $\boldsymbol{x}$ (and then $\boldsymbol{X}$) directly from $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$ for reference. We then computed the DCT's $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ where all DCT coefficients are quantized and then dequantized according to a given quantization matrix Δ. From $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$, we have computed $\boldsymbol{X}$ using both the standard approach and the proposed approach, and compared to the reference version, where the precision in each approach was measured in terms of the sum of squares of errors (MSE) in the DCT domain (and hence also in the spatial domain). For the case where Δ was an all-one matrix, the MSE of the proposed approach was about 3dB better than that of the standard approach. For the case where Δ was the recommended quantization matrix of JPEG for luminance [6, p. 37], the proposed approach outperformed the standard approach by 1.2dB. These results are reasonable because when the step sizes of the quantizer increase, quantization errors associated with the DCT coefficients tend to dominate roundoff errors associated with inaccurate computations.

The second experiment was similar but that test data that was a real image ("Lenna") rather than random data. Now, for the case where Δ was the all-one matrix, the standard approach yielded SNR of 46.08dB while the proposed approach gave 49.24dB, which is again a 3dB improvement. For the case where Δ was the JPEG default quantizer, the figures were 36.63dB and 36.84dB, respectively. Here the degree of improvement is less than in the case of random data because most of the DCT coeffients are rounded to zero in both techniques.

1 References

[1] R. B. Lee at al, "Achieving Realtime Software MPEG Decompression on a Multimedia-Enhanced PA-RISC Processor," . *Proc. Hewlett-Packard Image and Data Compression Conference*, Palo Alto, California, May 1994.

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

[3] W. Kou and T. Fjalbrant, "A Direct Computation of DCT Coefficients for a Signal Block Taken from Two Adjacent Blocks," *IEEE Trans. Signal Proc.*, Vol. SP-39, pp. 1692-1695, July 1991.

[4] J. B. Lee and B. G. Lee, "Transform Domain Filtering Based on Pipelining Structure," *IEEE Trans. Signal Proc.*, Vol. SP-40, pp. 2061-2064, August 1992.

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

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