A Fast Algorithm for DCT Domain Filtering

Neri Merhav
HP Israel Science Center*

and

Vasudev Bhaskaran
Computer Systems Laboratory†

Keywords: DCT domain filtering, data compression.

Abstract

A method is developed and proposed to efficiently implement spatial domain filtering directly on compressed digital video and images in the discrete cosine transform (DCT) domain. It is demonstrated that the computational complexity of this method is significantly smaller than that of the straightforward approach, of converting back to the uncompressed domain, convolving in the spatial domain, and retransforming to the DCT domain. It is assumed that the impulse response of the two dimensional filter is symmetric and separable. The method is applicable to any DCT based data compression standard, such as JPEG, MPEG, and H.261.

*Address: HP Israel Science Center, Technion City, Haifa 32000, Israel. Email: merhav@hp.technion.ac.il
†Address: Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A. Email: bhaskara@hpl.hp.com

1 Introduction

The last few years have witnessed a rapidly growing interest in developing fast algorithms for manipulating compressed images and video streams directly in the compressed domain, without explicit transformation back to the uncompressed domain, which is computationally expensive. When transform domain schemes are considered (like JPEG, MPEG, and H.261), in which the discrete cosine transform (DCT) coefficients are compressed, the "compressed domain" actually refers to the the DCT domain, while the "uncompressed domain" means the spatial domain. The most useful operations of image and video manipulation are scaling, masking, pixel-by-pixel multiplication, translation, rotation, overlapping, inverse motion compensation, and filtering. In this work we focus on the latter, i.e., the filtering (or convolution) operation, which is at the heart of many signal processing applications, e.g., noise suppression, edge sharpening, smoothing, antialiasing operations associated with upsampling and downsampling, and others.

Past work on DCT domain filtering has largely concentrated on convolution-multiplication properties (CMP's) of the DCT, in analogy to the well known CMP of the DFT. Chen and Fralick [1] have first shown that coefficient-by-coefficient multiplication in the DCT domain corresponds to circular convolution of three time domain (or spatial domain) sequences, one of which is a fixed undesired sequence, that can be eliminated by an appropriate modification of the DCT domain filter coefficients. Ngan and Clarke [2] have applied this property to lowpass filtering of images. Chitprasert and Rao [3] have simplified significantly the CMP of Chen and Fralick, however, their method is still applicable only to circular convolution (that causes block edge artifacts) rather than the more desirable linear convolution. More recently, Martucci [4] has derived a complete set of symmetrical convolution routines for a family of discrete trigonometric transforms (including the DCT). His methods can be modified to obtain linear convolution algorithms by appropriate zero padding in the convolution domain. Unfortunately, these algorithms cannot be used in most of our applications since the DCT domain data is already given without prior zero padding in the spatial domain.

As an alternative to the CMP approach, Lee and Lee [5] have used a simple algebraic approach to derive a transform domain linear convolution algorithm and proposed a pipelined hardware architecture. The basic idea in [5] was to precompute the product of the operator matrices corresponding to inverse DCT (IDCT), the convolution, and the DCT, and then to use the combined operator matrix directly in the DCT domain, where the contributions of neighboring DCT data blocks are incorporated similarly as in the overlap and add (OLA) method. Chang and Messerschmitt [6] proposed similar ideas by using the distributive property of the DCT with respect to matrix multiplication. A more thorough study of this approach, in combination with downsampling, has been carried out by Neri *et al.* [7].

In this work, we adopt and further develop the second approach of regarding the DCT domain filtering operator as a combined linear operator that acts directly on the DCT input data. Throughout this work, we assume a separable filter that is symmetric in both dimensions. We demonstrate that a fairly significant fraction of the computations can be saved compared to the straightforward approach of decompressing, filtering, and re-compressing, if one takes advantage of the following: First, the filtering algorithm can be performed recursively in the sense that some of the computations that were performed in filtering the previous block can be used for the current block. Second, the algorithm is tailored to the standard format of 8×8 blocks. Third, by creating certain butterflies on the input data, the combined linear operator (IDCT-convolution-DCT) can be represented by a relatively sparse matrix, and fourth, the typical sparseness of the DCT input data greatly eliminates arithmetic operations. It is demonstrated that if all these points are taken into account, $60-80\%$ of the computations are saved compared to the straightforward filtering method.

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 unitarity of S.

Filtering, or convolution, of an input image $\{I(i,j)\}$, (where i and j are integers taking on values in ranges that correspond to the size of the image), by a filter with impulse response $\{f(i,j)\}$, results in an output image $\{J(i,j)\}$ given by

$$J(i,j)=\sum_{i'}\sum_{j'}f(i',j')I(i-i',j-j') \tag{6}$$

where the range of summation over i' and j' is, of course, according to the support of the impulse response $\{f(i,j)\}$. In this work we assume that the filter $\{f(i,j)\}$ is *separable*, that is, $f(i,j)$ can be factorized as

$$f(i,j)=v_i h_j, \tag{7}$$

for some one-dimensional sequences $\{v_i\}$ and $\{h_j\}$, and the support of the impulse response is a rectangle. In this case, eq. (6) can be rewritten as

$$J(i,j)=\sum_{i'}v_{i'}\sum_{j'}h_{j'}I(i-i',j-j'), \tag{8}$$

namely, one can first perform a one-dimensional convolution on each row with the *horizontal filter component* (HFC) $\{h_j\}$, and then another one-dimensional convolution on each resulting column with the *vertical filter component* (VFC) $\{v_i\}$. Of course, the order can be interchanged and the vertical convolutions can be carried out first without affecting the final result. An important special case is the one where $v_i = h_i$ for all i, that is, the VFC and the HFC are the same. In this case, the filter is "isotropic" in the sense that rows and columns undergo the same convolution. We will not assume, however, that this property necessarily holds. We next assume that each filter component is symmetric about the origin, that is, $v_i = v_{-i}$ and $h_j = h_{-j}$. The supports of $\{v_i\}$ and $\{h_j\}$ are $|i| \leq M$ and $|j| \leq N$, respectively, meaning that $f(i,j) = 0$ outside a $(2M+1) \times (2N+1)$ rectangle centered at the origin.

The input image $\{I(i,j)\}$ is given in the compressed domain, that is, we are given a sequence of 8×8 matrices $\boldsymbol{X}_1, \boldsymbol{X}_2, ...$ of DCT coefficients corresponding to spatial domain 8×8 spatial domain blocks $\boldsymbol{x}_1, \boldsymbol{x}_2, ...$ that together form the input image $\{I(i,j)\}$. Our task is to compute the sequence of 8×8 matrices $\boldsymbol{Y}_1, \boldsymbol{Y}_2, ...$ of DCT coefficients of the spatial domain blocks $\boldsymbol{y}_1, \boldsymbol{y}_2, ...$ associated with the filtered image $\{J(i,j)\}$, directly from $\boldsymbol{X}_1, \boldsymbol{X}_2, ...$ without going via the spatial domain and perfoming spatial domain convolution.

We further assume that M and N do not exceed 8 (that is, the filter size is always smaller than 17×17), so that every DCT block $\boldsymbol{Y}$ (associated with the spatial domain block $\boldsymbol{y}$) of the filtered image $\{J(i,j)\}$ depends on the corresponding DCT block $\boldsymbol{X}$ (associated with the spatial domain block $\boldsymbol{x}$) of the input image $\{I(i,j)\}$ and the eight immediate neighbors of $\boldsymbol{X}$. We shall label these neighbors according to their relative location with respect to the current block $\boldsymbol{X}$, i.e., "north", "northeast", "east", etc. Accordingly, the input DCT blocks will be denoted by the appropriate subscript, i.e., $\boldsymbol{X}_N$, $\boldsymbol{X}_{NE}$, $\boldsymbol{X}_E$, and so on. Similarly, the respective spatial domain blocks will be denoted $\boldsymbol{x}_N$, $\boldsymbol{x}_{NE}$, $\boldsymbol{x}_E$, etc.

In summary, we are interested in an efficient algorithm that computes $\boldsymbol{Y}$ from $\boldsymbol{X}$, $\boldsymbol{X}_N$, $\boldsymbol{X}_{NE}$, $\boldsymbol{X}_E$, $\boldsymbol{X}_{SE}$, $\boldsymbol{X}_S$, $\boldsymbol{X}_{SW}$, $\boldsymbol{X}_W$, and, $\boldsymbol{X}_{NW}$.

3 Mathematical Derivation

In the spatial domain it is convenient to express $\boldsymbol{y}$, in terms of the nine input blocks and the filter, in the following block matrix form

$$\boldsymbol{y} = V \cdot \begin{pmatrix} \boldsymbol{x}_{NW} & \boldsymbol{x}_N & \boldsymbol{x}_{NE} \\ \boldsymbol{x}_W & \boldsymbol{x} & \boldsymbol{x}_E \\ \boldsymbol{x}_{SW} & \boldsymbol{x}_S & \boldsymbol{x}_{SE} \end{pmatrix} \cdot H^t \tag{9}$$

where V is a 8×24 matrix defined as

$$V = \begin{pmatrix} v_8 & v_7 & \cdot & \cdot & \cdot & v_1 & v_0 & v_1 & \cdot & \cdot & \cdot & v_7 & v_8 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & v_8 & v_7 & \cdot & \cdot & \cdot & v_1 & v_0 & v_1 & \cdot & \cdot & \cdot & v_7 & v_8 & 0 & \cdot & \cdot & 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 & \cdot & \cdot & \cdot & 0 & v_8 & v_7 & \cdot & \cdot & \cdot & v_1 & v_0 & v_1 & \cdot & \cdot & \cdot & v_7 & v_8 \end{pmatrix} \tag{10}$$

and where the high order coefficients are zero if $M < 8$. Similarly,

$$H = \begin{pmatrix} h_8 & h_7 & \cdot & \cdot & \cdot & h_1 & h_0 & h_1 & \cdot & \cdot & \cdot & h_7 & h_8 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & h_8 & h_7 & \cdot & \cdot & \cdot & h_1 & h_0 & h_1 & \cdot & \cdot & \cdot & h_7 & h_8 & 0 & \cdot & \cdot & 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 & \cdot & \cdot & \cdot & 0 & h_8 & h_7 & \cdot & \cdot & \cdot & h_1 & h_0 & h_1 & \cdot & \cdot & \cdot & h_7 & h_8 \end{pmatrix} \tag{11}$$

Since the DCT data in eq. (9) is partitioned into 8×8 blocks it will be convenient to do the same with the matrices H and V, that is, to define $V = [V_1\ V_2\ V_3]$ and $H = [H_1\ H_2\ H_3]$, where

$$V_1 = \begin{pmatrix} v_8 & v_7 & v_6 & v_5 & v_4 & v_3 & v_2 & v_1 \\ 0 & v_8 & v_7 & v_6 & v_5 & v_4 & v_3 & v_2 \\ 0 & 0 & v_8 & v_7 & v_6 & v_5 & v_4 & v_3 \\ 0 & 0 & 0 & v_8 & v_7 & v_6 & v_5 & v_4 \\ 0 & 0 & 0 & 0 & v_8 & v_7 & v_6 & v_5 \\ 0 & 0 & 0 & 0 & 0 & v_8 & v_7 & v_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & v_8 & v_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_8 \end{pmatrix} \tag{12}$$

$$V_2 = \begin{pmatrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ v_1 & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\ v_2 & v_1 & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \\ v_3 & v_2 & v_1 & v_0 & v_1 & v_2 & v_3 & v_4 \\ v_4 & v_3 & v_2 & v_1 & v_0 & v_1 & v_2 & v_3 \\ v_5 & v_4 & v_3 & v_2 & v_1 & v_0 & v_1 & v_2 \\ v_6 & v_5 & v_4 & v_3 & v_2 & v_1 & v_0 & v_1 \\ v_7 & v_6 & v_5 & v_4 & v_3 & v_2 & v_1 & v_0 \end{pmatrix} \tag{13}$$

$$V_3 = \begin{pmatrix} v_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_7 & v_8 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_6 & v_7 & v_8 & 0 & 0 & 0 & 0 & 0 \\ v_5 & v_6 & v_7 & v_8 & 0 & 0 & 0 & 0 \\ v_4 & v_5 & v_6 & v_7 & v_8 & 0 & 0 & 0 \\ v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & 0 & 0 \\ v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & 0 \\ v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{pmatrix} \tag{14}$$

and similar definitions for H_1, H_2, and H_3. We can now rewrite eq. (9) as follows.

$$\begin{aligned} \boldsymbol{y} \;=\; & (V_1\boldsymbol{x}_{NW} + V_2\boldsymbol{x}_W + V_3\boldsymbol{x}_{SW})H_1^t + \\ & (V_1\boldsymbol{x}_N + V_2\boldsymbol{x} + V_3\boldsymbol{x}_S)H_2^t + \\ & (V_1\boldsymbol{x}_{NE} + V_2\boldsymbol{x}_E + V_3\boldsymbol{x}_{SE})H_3^t \end{aligned} \tag{15}$$

Since $S^tS = I$, I being the 8×8 identity matrix, one can insert S^tS between every two multiplied matrices in eq. (15) and then premultiply both sides of the equation by S and postmultiply by S^t. This operation takes eq. (15) to the DCT domain, that is,

$$\begin{aligned}\boldsymbol{Y} &= (\boldsymbol{V}_1\boldsymbol{X}_{NW} + \boldsymbol{V}_2\boldsymbol{X}_W + \boldsymbol{V}_3\boldsymbol{X}_{SW})\boldsymbol{H}_1^t + \\ &\quad (\boldsymbol{V}_1\boldsymbol{X}_N + \boldsymbol{V}_2\boldsymbol{X} + \boldsymbol{V}_3\boldsymbol{X}_S)\boldsymbol{H}_2^t + \\ &\quad (\boldsymbol{V}_1\boldsymbol{X}_{NE} + \boldsymbol{V}_2\boldsymbol{X}_E + \boldsymbol{V}_3\boldsymbol{X}_{SE})\boldsymbol{H}_3^t, \end{aligned} \tag{16}$$

where $\boldsymbol{V}_i$ and $\boldsymbol{H}_i$, $i = 1, 2, 3$, are the DCT's of V_i and H_i, respectively.

While the matrices $\boldsymbol{V}_i$ and $\boldsymbol{H}_i$, $i = 1, 2, 3$, are not sparse in general, it turns out that

$$\boldsymbol{V}_{++} \triangleq \frac{\boldsymbol{V}_1 + \boldsymbol{V}_2 + \boldsymbol{V}_3}{2} \tag{17}$$

and

$$\boldsymbol{V}_{-+} \triangleq \frac{\boldsymbol{V}_1 - \boldsymbol{V}_2 + \boldsymbol{V}_3}{2} \tag{18}$$

never contain more than 20 nonzero elements, and

$$\boldsymbol{V}_{-} \triangleq \frac{\boldsymbol{V}_1 - \boldsymbol{V}_3}{2} \tag{19}$$

has exactly 32 nonzero elements (with a chessboard structure). Of course, the same is true for $\boldsymbol{H}_1$, $\boldsymbol{H}_2$ and $\boldsymbol{H}_3$ with similar definitions of $\boldsymbol{H}_{++}$, $\boldsymbol{H}_{-+}$, and $\boldsymbol{H}_{-}$.

A natural way to utilize this fact is to create "butterfilies" on the input data in such a way that eq. (16) will be expressed only in terms of the sparse matrices defined above. Specifically, let us define

$$\boldsymbol{X}_E^{++} = \frac{\boldsymbol{X}_{NE} + \boldsymbol{X}_{SE}}{2} + \boldsymbol{X}_E \tag{20}$$

$$\boldsymbol{X}_E^{+-} = \frac{\boldsymbol{X}_{NE} + \boldsymbol{X}_{SE}}{2} - \boldsymbol{X}_E \tag{21}$$

$$\boldsymbol{X}_E^{-} = \boldsymbol{X}_{NE} - \boldsymbol{X}_{SE} \tag{22}$$

$$\boldsymbol{X}^{++} = \frac{\boldsymbol{X}_N + \boldsymbol{X}_S}{2} + \boldsymbol{X} \tag{23}$$

$$\boldsymbol{X}^{+-} = \frac{\boldsymbol{X}_N + \boldsymbol{X}_S}{2} - \boldsymbol{X} \tag{24}$$

$$\boldsymbol{X}^{-} = \boldsymbol{X}_N - \boldsymbol{X}_S \tag{25}$$

$$\boldsymbol{X}_W^{++} = \frac{\boldsymbol{X}_{NW} + \boldsymbol{X}_{SW}}{2} + \boldsymbol{X}_W \tag{26}$$

$$\boldsymbol{X}_W^{+-} = \frac{\boldsymbol{X}_{NW} + \boldsymbol{X}_{SW}}{2} - \boldsymbol{X}_W \tag{27}$$

$$\boldsymbol{X}_W^{-} = \boldsymbol{X}_{NW} - \boldsymbol{X}_{SW} \tag{28}$$

Now eq. (16) can be written as follows.

$$\begin{aligned} \boldsymbol{Y} &= (\boldsymbol{V}_{++}\boldsymbol{X}_W^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}_W^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}_W^{-})\boldsymbol{H}_1^t + \\ &\quad (\boldsymbol{V}_{++}\boldsymbol{X}^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}^{-})\boldsymbol{H}_2^t + \\ &\quad (\boldsymbol{V}_{++}\boldsymbol{X}_E^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}_E^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}_E^{-})\boldsymbol{H}_3^t \end{aligned} \tag{29}$$

Let us denote

$$\boldsymbol{Z}_1 = \boldsymbol{V}_{++}\boldsymbol{X}_W^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}_W^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}_W^{-} \tag{30}$$

$$\boldsymbol{Z}_2 = \boldsymbol{V}_{++}\boldsymbol{X}^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}^{-} \tag{31}$$

$$\boldsymbol{Z}_3 = \boldsymbol{V}_{++}\boldsymbol{X}_E^{++} + \boldsymbol{V}_{-+}\boldsymbol{X}_E^{+-} + \boldsymbol{V}_{-}\boldsymbol{X}_E^{-} \tag{32}$$

If we assume that the blockwise filtering procedure is performed in a raster scan order (i.e., left-to-right and top-to-bottom), then it is easy to note that $\boldsymbol{Z}_2$ is identical to $\boldsymbol{Z}_3$ of the previously processed DCT block, and similarly, $\boldsymbol{Z}_1$ is the same as $\boldsymbol{Z}_3$ two steps ago. Thus, one needs only to calculate $\boldsymbol{Z}_3$ in every step and to save it for the next two steps. Finally, we can rewrite eq. (29) as follows.

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{Z}_1\boldsymbol{H}_1^t + \boldsymbol{Z}_2\boldsymbol{H}_2^t + \boldsymbol{Z}_3\boldsymbol{H}_3^t \\ &= \boldsymbol{Z}_{++}\boldsymbol{H}_{++}^t + \boldsymbol{Z}_{+-}\boldsymbol{H}_{-+}^t + \boldsymbol{Z}_{-}\boldsymbol{H}_{-}^t, \end{aligned} \tag{33}$$

where

$$\boldsymbol{Z}_{++} = \frac{\boldsymbol{Z}_1 + \boldsymbol{Z}_3}{2} + \boldsymbol{Z}_2 \tag{34}$$

$$\boldsymbol{Z}_{+-} = \frac{\boldsymbol{Z}_1 + \boldsymbol{Z}_3}{2} - \boldsymbol{Z}_2 \tag{35}$$

$$\boldsymbol{Z}_{-} = \boldsymbol{Z}_1 - \boldsymbol{Z}_3 \tag{36}$$

4 The Proposed Algorithm

We can now summarize the proposed filtering algorithm. In the parentheses we provide the number of arithmetic operations associated with each step in the form of an expression $\alpha \cdot m + \beta \cdot a$, which means α multiplications plus β additions.

1. Store the previous value of $\boldsymbol{Z}_2$ in $\boldsymbol{Z}_1$ and the previous $\boldsymbol{Z}_3$ in $\boldsymbol{Z}_2$.

2. Compute $\boldsymbol{X}_E^{+} \triangleq (\boldsymbol{X}_{NE} + \boldsymbol{X}_{SE})/2$. ($64a$)

3. Compute $\boldsymbol{X}_E^{++} = \boldsymbol{X}_E^{+} + \boldsymbol{X}_E$. ($64a$)

4. Compute $\boldsymbol{X}_E^{+-} = \boldsymbol{X}_E^{+} - \boldsymbol{X}_E$. ($64a$)

5. Compute $\boldsymbol{X}_E^{-}$ according to eq. (22). ($64a$)

6. Compute $\boldsymbol{Z}_3$ according to eq. (32). ($576m + 576a$)

7. Compute $\boldsymbol{Z}_+ \triangleq (\boldsymbol{Z}_1 + \boldsymbol{Z}_3)/2$. $(64a)$
8. Compute $\boldsymbol{Z}_{++} = \boldsymbol{Z}_+ + \boldsymbol{Z}_2$. $(64a)$
9. Compute $\boldsymbol{Z}_{+-} = \boldsymbol{Z}_+ - \boldsymbol{Z}_2$. $(64a)$
10. Compute $\boldsymbol{Z}_-$ according to eq. (36). $(64a)$
11. Compute $\boldsymbol{Y}$ according to eq. (33). $(576m + 576a)$

The total number of operations is $1152m + 1664a$.

Let us compare this result with the total number of computations associated with the straightforward approach of going explicitly via the spatial domain. We assume that the DCT and the IDCT are performed by the fastest known 8-point algorithm due to Arai, Agui, and Nakajima [8] (see also [9]), which requires 5 multiplications and 29 additions.

1. Store $\boldsymbol{x}_N$, $\boldsymbol{x}$, $\boldsymbol{x}_S$, $\boldsymbol{x}_{NE}$, $\boldsymbol{x}_E$, and $\boldsymbol{x}_{SE}$ of the previous step in $\boldsymbol{x}_{NW}$, $\boldsymbol{x}_W$, $\boldsymbol{x}_{SW}$, $\boldsymbol{x}_N$, $\boldsymbol{x}$, and $\boldsymbol{x}_S$, respectively.
2. Compute the IDCT's $\boldsymbol{x}_{NE}$, $\boldsymbol{x}_E$, and $\boldsymbol{x}_{SE}$. (total: $16 \times 3 \times (5m + 29a) = 240m + 1392a$)
3. Compute the vertical convolution using the symmetry of $\{v_i\}$, i.e., $v_0 x_0 + \sum_{i=1}^{M} v_i(x_i + x_{-i})$. $(64(M+1) \cdot m + 128M \cdot a)$
4. Compute similarly the horizontal convolution. $(64(N+1) \cdot m + 128N \cdot a)$
5. Compute the DCT of the filtered block. $(16 \times (5m + 29a) = 80m + 464a)$

Thus, the total number of operations associated with the straightforward approach is

$$(64L + 448)m + (128L + 1856)a \tag{37}$$

where $L \triangleq M + N$.

In this work, we are more interested in the number of basic arithmetic operations on the PA-RISC processor (see also [10], [11]). 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 in this example.

A close inspection of the above mentioned fast DCT/IDCT algorithm reveals that it takes 50 elementary operations to transform an 8-point vector and hence $16 \times 50 = 800$ operations to transform an 8×8 matrix. As for the other arithmetic operations, which depend on the given filter coefficients, we assume that multiplication by a constant takes on the average 3 elementary PA-RISC operations, i.e., $m = 3$ and $a = 1$. Thus, the proposed approach requires $1152 \times 3 + 1664 = 5120$ operations while the straightforward approach requires $320L + 3584$ operations. This means that the proposed approach is preferrable for every $L = M + N \geq 5$. In the extreme case $L = 16$, the new approach saves 41% of the computations.

5 Sparse DCT Domain Input Data

So far our analysis was general in the sense that no assumptions were made on the structure of the DCT input data. It is well known, however, that typically, DCT coefficient matrices of real images are very sparse, especially after quantization. This happens because the DCT has the property [12, Sect. 6.3] that most of the energy is concentrated on a relatively small number of coefficients, normally, the ones corresponding to low spatial frequencies.

Similarly as in [10] and [11], we shall define a DCT coefficient matrix as *sparse* if only its 4×4 upper left quadrant (corresponding to low frequencies in both directions) contains nonzero elements. A very high percentage of the DCT blocks usually satisfy this requirement and it is fairly easy to check directly in the compressed format.

We have redesigned the filtering algorithm under the assumption that all nine DCT data blocks are sparse, and arrived at the following operation count.

1. Store the previous value of $\boldsymbol{Z}_2$ in $\boldsymbol{Z}_1$ and the previous $\boldsymbol{Z}_3$ in $\boldsymbol{Z}_2$.
2. Compute $\boldsymbol{X}_E^+ \triangleq (\boldsymbol{X}_{NE} + \boldsymbol{X}_{SE})/2$. $(16a)$
3. Compute $\boldsymbol{X}_E^{++} = \boldsymbol{X}_E^+ + \boldsymbol{X}_E$. $(16a)$
4. Compute $\boldsymbol{X}_E^{+-} = \boldsymbol{X}_E^+ - \boldsymbol{X}_E$. $(16a)$
5. Compute $\boldsymbol{X}_E^-$ according to eq. (22). $(16a)$
6. Compute $\boldsymbol{Z}_3$ according to eq. (32). $(80m + 96a)$
7. Compute $\boldsymbol{Z}_+ \triangleq (\boldsymbol{Z}_1 + \boldsymbol{Z}_3)/2$. $(32a)$
8. Compute $\boldsymbol{Z}_{++} = \boldsymbol{Z}_+ + \boldsymbol{Z}_2$. $(32a)$
9. Compute $\boldsymbol{Z}_{+-} = \boldsymbol{Z}_+ - \boldsymbol{Z}_2$. $(32a)$
10. Compute $\boldsymbol{Z}_-$ according to eq. (36). $(32a)$
11. Compute $\boldsymbol{Y}$ according to eq. (33). $(288m + 256a)$

The total is therefore $368m + 544a$, which yields 1648 basic PA-RISC operations under the assumptions of Section 4. This means, that even for the case where $L = 2$, approximately 60% of the computations are saved. On the other extreme, $L = 16$, the saving factor is about 80%.

A point to observe is that while in the general case (Section 4), steps 2-6 were structurally identical to steps 7-11, respectively, and hence required the same number of computations, here this is no longer the case. The reason is that now, in steps 2-6 the algoritm acts directly on the input data matrices which are assumed sparse and hence have at most 16 nonzero elements, but in steps 7-11, the inputs are the $\boldsymbol{Z}$-matrices for which 32 elements may not be zero.

6 Conclusion

We have developed an algorithm that efficiently implements spatial domain filtering directly on compressed digital video and images in the discrete cosine transform (DCT) domain. We assumed that the given two-dimensional filter is symmetric in both dimensions and separable, which is normally the case in many applications. It has been demonstrated that the computational complexity of this method is significantly smaller than that of the straightforward approach, of converting back to the uncompressed domain, convolving in the spatial domain, and retransforming to the DCT domain. Specifically, for typically sparse DCT input data $60 - 80\%$ of the computations are saved depending on the size of the filter. The approach developed here is easy to extend to situations where either M or N or both may exceed the value 8, by taking into account additional neighboring blocks of the input image. A similar derivation can be easily carried out for filters that are antisymmetric rather than symmetric in one direction or both.

8 References

[1] W. H. Chen and S. C. Fralick, "Image enhancement using cosine transform filtering," *Image Sci. Math. Symp.*, Montrey, CA, November 1976.

[2] K. N. Ngan and R. J. Clarke, "Lowpass filtering in the cosine transform domain," *Int. Conf. on Commun.*, Seattle, WA, pp. 37.7.1-37.7.5, June 1980.

[3] B. Chitpraset and K. R. Rao, "Discrete cosine transform filtering," *Signal Processing*, vol. 19, pp. 233-245, 1990.

[4] S. A. Martucci, "Symmetric convolution and discrete sine and cosine transforms," *IEEE Trans. on Signal Processing*, vol. SP-42, no. 5, pp. 1038-1051, May 1994.

[5] J. B. Lee and B. G. Lee, "Transform domain filtering based on pipelining structure," *IEEE Trans. on Signal Processing*, vol. SP-40, no. 8, pp. 2061-2064, August 1992.

[6] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Selected Areas in Communications*, vol. 13, no. 1, pp. 1-11, January 1995.

[7] A. Neri, G. Russo, and P. Talone, "Inter-block filtering and downsampling in DCT domain," *Signal Processing: Image Communication*, vol. 6, pp. 303-317, 1994.

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

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

[10] N. Merhav and V. Bhaskaran, "A transform domain approach to spatial domain image scaling," HPL Technical Report #HPL-94-116, December 1994.

[11] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT domain inverse motion compensation," HPL Technical Report #HPL-95-17, February 1995.

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