Fast DCT Domain Filtering Using the DCT and the DST[*]

Renato Kresch[†]
Hewlett-Packard Laboratories – Israel
Technion City, Haifa 32000, Israel
and
Neri Merhav[‡]
Department of Electrical Engineering,
Technion, Haifa, 32000, Israel.

Abstract

A method for efficient spatial domain filtering, directly in the DCT domain, is developed and proposed. It consists of using the discrete sine transform (DST) and the discrete cosine transform (DCT) for transform-domain processing on the basis of recently derived convolution-multiplication properties of discrete trigonometric transforms. The proposed scheme requires neither zero padding of the input data nor kernel symmetry. It is demonstrated that, in typical applications, the proposed algorithm is significantly more efficient than the conventional spatial domain and earlier proposed DCT domain methods. The proposed method is applicable to any DCT-based image compression standard, such as JPEG, MPEG, and H.261.

Keywords: DCT-domain filtering, discrete sine transform, image compression, convolution-multiplication properties, trigonometric transforms, compressed-domain processing.

[*]Suitable EDICS categories: 1.2, 1.12, or 1.13.

[†]Corresponding author. Tel: +972-4-823-1237, fax: +972-4-822-0407, e-mail: `renato@hp.technion.ac.il`.

[‡]This work was partially done while N. Merhav was on sabbatical leave at Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA. N. Merhav is also with Hewlett-Packard Laboratories – Israel. E-mail: `merhav@hp.technion.ac.il`.

This work addresses the problem of efficient 2-D linear filtering in the discrete cosine transform (DCT) domain, which is an important problem in the area of processing and manipulation of images and video streams compressed in DCT-based methods, such as JPEG, MPEG, H.261, and others (see, e.g., [1-7]). More specifically, suppose that an input image is given in the format of a sequence of sets of DCT coefficients of 2-D blocks. We assume the DCT to be of type II-e (according to the classification in [4]), which is the type used in JPEG/MPEG/H.261 applications. We are interested in calculating efficiently a filtered image, in the same format, that corresponds to spatial domain convolution between the input image and a given filter.

The straightforward spatial-domain approach consists of calculating the inverse DCT (IDCT) of each block, performing the convolution in the spatial domain, and then transforming the resulting image back to the DCT domain (see Fig. 1(a)). Our aim is to operate directly in the DCT domain, avoiding explicit transformations from the DCT domain to the spatial domain and vice versa (see Fig. 1(b)).

One approach to this problem is to use a convolution-multiplication property (CMP) for the DCT. Like CMP's for the discrete Fourier transform (DFT), the aim of using a CMP for the DCT is to turn the filtering operation into a simple element-by-element multiplication in the transform domain. Chen and Fralick [1] derived one such CMP, and Ngan and Clarke [2] have used it for low-pass filtering of images. Later, Chitprasert and Rao [3] derived a simpler CMP. A thorough study of CMP's was performed by Martucci [4], who derived CMP's for all types of DCT's and DST's (discrete sine transforms), including the previously mentioned CMP's as special cases. Sánchez *et al.* [6] rewrote part of the CMP's (some of those involving DCT's only) in the form of matrix diagonalization properties, and studied their asymptotic behavior. The problem with almost all the schemes proposed in the above mentioned works is that they do not implement *linear* convolution, but rather a "folded" type of convolution, called *symmetric* convolution. An exception is the linear convolution scheme proposed by Martucci [4, 5], who showed that one can obtain linear convolutions from symmetric convolutions through appropriate zero-padding in the spatial domain. Unfortunately, in our application, the data are already provided in the transform domain without prior zero-padding, and therefore Matucci's scheme cannot be used directly. Another disadvantage of the above methods is that they are limited to symmetric and antisymmetric filters.

The works of Lee and Lee [7], Chang and Messerschmitt [8], and Neri *et al.* [9] provide an alternative to the CMP approach. They propose to precompute the product of the operator ma-operator matrix directly in the DCT domain. The efficiency of this approach depends heavily on the sparseness of the DCT data; processing of non-sparse DCT blocks can be of high computational complexity. Merhav and Bhaskaran [14] improved this approach by using certain butterflies on the input data, prior to the filtering operation. Their approach is significantly more efficient than its predecessors.

In this work, we follow the CMP approach, and propose an efficient filtering scheme of the DCT data, that does not require spatial domain zero-padding, and that is suitable for, symmetric, antisymmetric, and non-symmetric filtering. The proposed scheme uses the DCT *and* the DST coefficients of the data, based on the DCT/DST CMP's derived in [4]. Since the DST coefficients are not available in advance, they have to be computed from the given DCT data. To this end, we develop fast algorithms that directly transform from the DCT to the DST (denoted CST) and vice-versa (SCT). Incorporating these algorithms in the filtering scheme, we obtain an overall complexity that is substantially smaller than those of the spatial domain approach, Lee and Lee's, and Merhav and Bhaskaran's schemes in several relevant situations.

For sake of simplicity, and due to the practical usefulness of separable 2-D filters, we confine our attention, in this paper, to separable filters only. Since the DCT/DST CMP's can be easily extended to 2-D signals, the extension of the proposed filtering scheme to nonseparable filters is fairly straightforward.

The outline of this paper is as follows. Section 2 provides the theoretical background for this work. In Section 3, the relevant CMP's are converted to a matrix form to be used in the derivation of the filtering scheme, which is presented in Section 4 for the 1-D case. Section 5 extends the algorithm to 2-D images, and separable kernels. In Section 6, a comparative complexity analysis is provided. Finally, Section 7 concludes the paper.

2 Background

2.1 Convolution-Multiplication Properties of DTT's

There are 8 types of DCT's and 8 types of DST's defined by Wang, which are generically called discrete trigonometric transforms (DTT's) [4]. From the family of DTT's, we are particularly interested in the DCT of type 2e, denoted DCT-IIe, since it is useful in image coding standards JPEG, MPEG, and others.

However, each DTT appears in the literature in several different forms, which differ only by scaling factors. I.e., the operator matrix of each DTT in one form can be converted to another form by pre- and post-multiplication by non-singular diagonal matrices. In [10], the transforms are presented in an *orthogonal form*, where the corresponding operator matrices are unitary. Martucci proposed a *convolution form* in [4], which is more appropriate for presenting the DTT CMP's because it avoids the need for additional scaling or weighting in the CMP formulæ.

Martucci has summarized all the CMP's related to the DTT's in [4, Tables VI and VII]. They are grouped in families of three or four CMP's that share the same input and output DTT types. Since we wish to produce a filtering scheme having DCT-2e blocks as input and output, we are particularly interested on the CMP family for which one of the forward DTT's and the inverse DTT are of type 2e. These are CMP's 4-7 in Table VI of [4], which involve also the DST of the same type, DST-IIe, and the DCT and DST of type 1e, denoted DCT-Ie and DST-Ie, respectively. We review here these CMP's.

The N^{th} order transform matrices of DCT-Ie, DST-Ie, DCT-IIe, and DST-IIe in convolution form are denoted $\mathcal{C}_{1e}$, $\mathcal{S}_{1e}$, $\mathcal{C}_{2e}$, and $\mathcal{S}_{2e}$, respectively, and are defined by [4]:

$$[\mathcal{C}_{1e}]_{mn} = 2k_n \cos\left(\tfrac{mn\pi}{N}\right), \quad m,n = 0,1,\ldots,N \tag{1}$$

$$[\mathcal{S}_{1e}]_{mn} = 2\sin\left(\tfrac{mn\pi}{N}\right), \quad m,n = 1,2,\ldots,N-1 \tag{2}$$

$$[\mathcal{C}_{2e}]_{mn} = 2\cos\left(\frac{m(n+\frac{1}{2})\pi}{N}\right), \quad m,n = 0,1,\ldots,N-1 \tag{3}$$

$$[\mathcal{S}_{2e}]_{mn} = 2\sin\left(\frac{m(n+\frac{1}{2})\pi}{N}\right), \quad \begin{cases} n = 0,\ldots,N-1, \\ m = 1,\ldots,N \end{cases} \tag{4}$$

where $k_i = 1$, for $1 \leq i \leq N-1$, and $k_0 = k_N = 1/2$.

Another basic concept in the theory of DTT's is that of *symmetric-extension.* It consists of replicating a given input sequence in order to produce a symmetric output sequence. There are 16 types of symmetric-extension operators defined in [4], four of which are of interest to us, denoted by HSHS, HAHA, WSWS, WAWA, according to the position of the symmetry point ('H' and 'W' meaning 'Half sample' and 'Whole sample', respectively), and the symmetric/antisymmetric nature of the replication ('S' and 'A' meaning 'symmetric' and 'antisymmetric', respectively). They are given by the following rules [4]:

$$\mathrm{HSHS}(x_1,\ldots,x_n) = \mathcal{R}(x_1,\ldots,x_n,x_n,\ldots,x_2),$$

$$\begin{aligned}\mathrm{WSWS}(x_1,\ldots,x_n) &= \mathcal{R}(x_1,\ldots,x_{n-1},x_n,x_{n-1},\ldots,x_2),\\ \mathrm{WAWA}(x_1,\ldots,x_n) &= \mathcal{R}(0,x_2,\ldots,x_{n-1},0,-x_{n-1},\ldots,-x_2),\end{aligned}$$

where $\mathcal{R}$ performs periodic replication.

The four CMP's associated with the above operators and transforms assume the following form [4]:

$$w_n = \varepsilon_a\{x_n\} \circledast \varepsilon_b\{y_n\} = \mathcal{T}_c^{-1}\{\mathcal{T}_a\{x_n\} \times \mathcal{T}_b\{y_n\}\}, \tag{5}$$

where $\{x_n\}$ and $\{y_n\}$ are two input sequences of finite length, and $\{w_n\}$ is the output convolved sequence. In the above expression, ε_a and ε_b are two symmetric extension operators, and $\circledast$ denotes circular convolution. The operators $\mathcal{T}_a$, $\mathcal{T}_b$, and $\mathcal{T}_c^{-1}$ are, respectively, two forward and one inverse DTT's in convolution form, appropriately selected among the above specific DTT's, and $\times$ denotes element-by-element multiplication. Table 1 lists the specific values for the four CMP's.

Equation (5) tells us that symmetric convolution (which means circular convolution between symmetrically-extended versions of the operands) can be obtained by transforming the input signals by DTT's, multiplying the results element-by-element, and then performing an inverse DTT. Equation (5) is a simplified version of the equation provided in [4], adapted to the specific four DTT's considered here.

When using the DTT's in the CMP's, one has to pay particular attention to the input (n) and output (m) index ranges defined within the DTT's definitions (eqs. (1)-(4)). They characterize the elements of the input sequence that should be operated upon, and the index of the transformed-sequence elements. For instance, the input index range of $\mathcal{S}_{1e}$ is $n = 1, \ldots, N-1$, which means that only the input elements with these indices are used in the transform, whereas the elements x_0 and x_N are disregarded. Moreover, its output range is also $m = 1, \ldots, N-1$, which means that the elements obtained from the transformation should be indexed accordingly, while the values of the transform elements with indices 0 and N should be set to 0. The range of the input sequences and the output range of the convolved sequence in Table 1 are directly related to the input and output ranges of the transforms involved in the CMP's.

Some of the CMP's of DTT's presented in [4] (in particular, most of the CMP's involving DCT's only) were converted into matrix form in [6], where they assume the following structure:

$$[\mathcal{Y}^a] = [\mathcal{C}_a]^{-1}[\mathcal{D}\left([\mathcal{C}_b]\boldsymbol{y}\right)][\mathcal{C}_a]. \tag{6}$$

In (6), $[\mathcal{Y}^a]$ is a matrix that performs one of the sixteen types of symmetric convolution of an input signal by the signal $\boldsymbol{y}$, $[\mathcal{D}(\boldsymbol{x})]$ is a diagonal matrix whose diagonal elements are the corresponding elements of $\boldsymbol{x}$, and $[\mathcal{C}_a]$ and $[\mathcal{C}_b]$ are the operator matrices of two DCT's in convolution form. Note that the inverse transform is an IDCT of the same type as one of the forward DCT's, a property which is not necessarily extended for similar matrix formulation for the rest of the DTT CMP's. The meaning of (6) is that the symmetric convolution matrix $[\mathcal{Y}^a]$ is diagonalized by $[\mathcal{C}_a]$, where the eigenvalues are the coefficients of the transform of $\boldsymbol{y}$ by $\mathcal{C}_b$. This is analogous to circular convolution matrices that are diagonalized by the DFT matrix, with eigenvalues given by the DFT coefficients of the kernel. In [6], the above matrix formulation for DCT matrices in convolution form was modified for DCT matrices in orthogonal form as well, where scaled and weighted versions of $[\mathcal{Y}^a]$ are the matrices which are diagonalized.

It was also noted in [6] that the symmetric convolution matrices $[\mathcal{Y}^a]$ can be decomposed as a sum of a symmetric Toeplitz matrix and a Hankel or a nearly Hankel matrix. The latter term corresponds to the "folding" effect caused by the symmetric convolution operation. One can observe in [6] that the Hankel-matrix component is also symmetric or close to symmetric w.r.t. the second main diagonal.

3 Conversion of the CMP's to Matrix Form

The filtering derivation in the paper is done in a matrix form. For this purpose, we need to have all four CMP's of interest in that form. Since only the first of them involves DCT's only, this is the only one that has already been converted to the matrix form in [6]. The conversion of the other three CMP's is performed in this section.

3.1 Conversion

The conversion can be done with the same derivation steps as those used in [6] for the CMP's involving DCT's only, but with the following three considerations. First, the forward and inverse

(6).

The second consideration refers to the symmetric convolution matrices. Associated with each one of the four CMP's of interest here, there is a different type of symmetric convolution as listed in Table 1, namely, HSHS$\circledast$WSWS, HSHS$\circledast$WAWA, HAHA$\circledast$WSWS, and HAHA$\circledast$WAWA. Therefore, we symbolize the symmetric convolution matrices by $[\mathcal{Y}^{a,b}]$, where the indices a and b assume the values $a \in \{\text{HS}, \text{HA}\}$ and $b \in \{\text{WS}, \text{WA}\}$, covering the above four cases. Following the same derivation steps as in [6], one can show that, similarly as before, the above matrices can all be decomposed into sums of Toeplitz and Hankel matrices. However, they can be either symmetric or anti-symmetric, rather than symmetric only. In order to represent all these different cases, we further decompose the corresponding Toeplitz and Hankel matrices into the following triangular matrices:

$$Y_1 \triangleq \begin{pmatrix} \frac{y_0}{2} & 0 & \cdots & 0 & 0 \\ y_1 & \frac{y_0}{2} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ y_{N-2} & \ddots & \ddots & \frac{y_0}{2} & 0 \\ y_{N-1} & y_{N-2} & \ddots & y_1 & \frac{y_0}{2} \end{pmatrix}, \quad Y_2 \triangleq \begin{pmatrix} y_1 & y_2 & \cdots & y_{N-1} & \frac{y_N}{2} \\ y_2 & y_3 & \cdots & \frac{y_N}{2} & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ y_{N-1} & \frac{y_N}{2} & \cdots & \cdots & 0 \\ \frac{y_N}{2} & 0 & \cdots & 0 & 0 \end{pmatrix}, \tag{7}$$

$$Y_3 \triangleq \begin{pmatrix} \frac{y_0}{2} & y_1 & \cdots & y_{N-2} & y_{N-1} \\ 0 & \frac{y_0}{2} & \ddots & \ddots & y_{N-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \frac{y_0}{2} & y_1 \\ 0 & 0 & \ddots & 0 & \frac{y_0}{2} \end{pmatrix}, \quad Y_4 \triangleq \begin{pmatrix} 0 & 0 & \cdots & 0 & \frac{y_N}{2} \\ 0 & 0 & \cdots & \frac{y_N}{2} & y_{N-1} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & \frac{y_N}{2} & \cdots & \cdots & y_2 \\ \frac{y_N}{2} & y_{N-1} & \cdots & y_2 & y_1 \end{pmatrix}. \tag{8}$$

Note that $Y_3 = Y_1^t$, and that Y_4 can be viewed as a transposition of Y_2, but w.r.t. the other main diagonal. Therefore, $Y_1 + Y_3$ is symmetric Toeplitz, $Y_1 - Y_3$ is antisymmetric Toeplitz, $Y_2 + Y_4$ is symmetric Hankel, and $Y_2 - Y_4$ is antisymmetric Hankel, where symmetry in the two latter cases is w.r.t. to the secondary diagonal. The corresponding convolution matrices can now be written as a combination of the above triangular matrices as follows:

$$\begin{aligned} [\mathcal{Y}^{\text{HS,WS}}] &= (Y_1 + Y_3) + (Y_2 + Y_4), \\ [\mathcal{Y}^{\text{HS,WA}}] &= (Y_1 - Y_3) + (Y_2 - Y_4), \\ [\mathcal{Y}^{\text{HA,WS}}] &= (Y_1 + Y_3) - (Y_2 + Y_4), \\ [\mathcal{Y}^{\text{HA,WA}}] &= (Y_1 - Y_3) - (Y_2 - Y_4). \end{aligned} \tag{9}$$

and output ranges involved in the four CMP's of interest, which forces us to be careful with index determination in the matrix format. To make it simpler to deal with this issue, we first extend the above DTT's operator matrices by zero padding, in order to intrinsically adjust index manipulations. Later on, in Section 3.2, the zeros are dropped from the final formulæ. Thus, we produce the following four extended matrices, by simply extending input/output index ranges in the DTT's definitions:

$$[\bar{\mathcal{C}}_{1e}] \triangleq [\mathcal{C}_{1e}], \qquad [\bar{\mathcal{S}}_{1e}] \triangleq \left(\begin{array}{c|c|c} 0 & 0\ldots 0 & 0 \\ \hline 0 & & 0 \\ \vdots & [\mathcal{S}_{1e}] & \vdots \\ 0 & & 0 \\ \hline 0 & 0\ldots 0 & 0 \end{array}\right),$$

$$[\bar{\mathcal{C}}_{2e}] \triangleq \left(\begin{array}{c} [\mathcal{C}_{2e}] \\ \hline 0\ldots 0 \end{array}\right), \qquad [\bar{\mathcal{S}}_{2e}] \triangleq \left(\begin{array}{c} 0\ldots 0 \\ \hline [\mathcal{S}_{2e}] \end{array}\right). \tag{10}$$

Notice that $[\bar{\mathcal{C}}_{1e}]$ and $[\bar{\mathcal{S}}_{1e}]$ are both of size $(N+1)\times(N+1)$, whereas $[\bar{\mathcal{C}}_{2e}]$ and $[\bar{\mathcal{S}}_{2e}]$ are both of size $(N+1)\times N$. Moreover, we define:

$$[\bar{\mathcal{C}}_{2e}]^{-1} \triangleq \left(\begin{array}{c|c} & 0 \\ [\mathcal{C}_{2e}]^{-1} & \vdots \\ & 0 \end{array}\right), \qquad [\bar{\mathcal{S}}_{2e}]^{-1} \triangleq \left(\begin{array}{c|c} 0 & \\ \vdots & [\mathcal{S}_{2e}]^{-1} \\ 0 & \end{array}\right). \tag{11}$$

These matrices are both of size $N\times(N+1)$.

Taking into account the above considerations, the four CMP's of interest can now be written in the matrix form as follows:

$$\begin{aligned} [\mathcal{Y}^{HS,WS}] &= [\bar{\mathcal{C}}_{2e}]^{-1}[\mathcal{D}([\bar{\mathcal{C}}_{1e}]\boldsymbol{y})][\bar{\mathcal{C}}_{2e}], \\ [\mathcal{Y}^{HS,WA}] &= [\bar{\mathcal{S}}_{2e}]^{-1}[\mathcal{D}([\bar{\mathcal{S}}_{1e}]\boldsymbol{y})][\bar{\mathcal{C}}_{2e}], \\ [\mathcal{Y}^{HA,WS}] &= [\bar{\mathcal{S}}_{2e}]^{-1}[\mathcal{D}([\bar{\mathcal{C}}_{1e}]\boldsymbol{y})][\bar{\mathcal{S}}_{2e}], \\ [\mathcal{Y}^{HA,WA}] &= -[\bar{\mathcal{C}}_{2e}]^{-1}[\mathcal{D}([\bar{\mathcal{S}}_{1e}]\boldsymbol{y})][\bar{\mathcal{S}}_{2e}], \end{aligned} \tag{12}$$

where $\boldsymbol{y} = \{y_0, y_1, \ldots, y_{N-1}, y_N\}$.

We now perform two modifications in the above formulæ in order to obtain the final equations that are used for the derivation of the filtering algorithm in the next sections. First, we convert the type-2e transforms to the orthogonal form. This is done for simplicity of the derivation, and because this form is closer to the one used in compression standards. The DCT-IIe and DST-IIe in orthogonal form are denoted in [4] by $\mathcal{C}_{IIe}$ and $\mathcal{S}_{IIe}$, respectively. Here, however, we simplify the notation by referring to them as C and S, respectively. They are defined as follows [10]:

$$C \triangleq [\mathcal{C}_{IIe}]_{mn} = \sqrt{\tfrac{2}{N}} k_m \cos\left(\frac{m(n+\frac{1}{2})\pi}{N}\right), \quad m, n = 0, 1, \ldots, N-1 \tag{13}$$

$$S \triangleq [\mathcal{S}_{IIe}]_{mn} = \sqrt{\tfrac{2}{N}} k_m \sin\left(\frac{m(n-\frac{1}{2})\pi}{N}\right), \quad m, n = 1, 2, \ldots, N. \tag{14}$$

Notice that both matrices are of size $N \times N$. Since the above matrices are in orthogonal form, their inverses are given by transposition, that is, $C^{-1} = C^t$ and $S^{-1} = S^t$.

The relationships between the extended 2e-type transform matrices in convolution and orthogonal forms are given by:

$$\begin{aligned} [\bar{\mathcal{C}}_{2e}] &= R\bar{C}, & [\bar{\mathcal{S}}_{2e}] &= R\bar{S}, \\ [\bar{\mathcal{C}}_{2e}]^{-1} &= \bar{C}^{-1}R^{-1}, & [\bar{\mathcal{S}}_{2e}]^{-1} &= \bar{S}^{-1}R^{-1}, \end{aligned} \tag{15}$$

where $\bar{C}$, $\bar{S}$, $\bar{C}^{-1}$, and $\bar{S}^{-1}$ are obtained from C, S, C^{-1}, and S^{-1} with the same zero-padding as above for $[\bar{\mathcal{C}}_{2e}]$, $[\bar{\mathcal{S}}_{2e}]$, $[\bar{\mathcal{C}}_{2e}]^{-1}$, and $[\bar{\mathcal{S}}_{2e}]^{-1}$, respectively, and R is a $(N+1) \times (N+1)$ diagonal matrix, with diagonal elements given by $r_i = \sqrt{2N}$, for $i = 1, \ldots, N-1$, and $r_0 = r_N = 2\sqrt{N}$.

Notice that upon substituting (15) into (12), the matrix R is canceled out. Therefore, all instances of $[\bar{\mathcal{C}}_{2e}]$, $[\bar{\mathcal{S}}_{2e}]$, $[\bar{\mathcal{C}}_{2e}]^{-1}$, and $[\bar{\mathcal{S}}_{2e}]^{-1}$ in the CMP's can be replaced by $\bar{C}$, $\bar{S}$, $\bar{C}^{-1}$, and $\bar{S}^{-1}$, respectively.

The second modification is to remove the padded zeros from the CMP's, producing more efficient formulæ.

The above two modifications together with (9) lead to the following final form of the desired four diagonalization properties:

$$[\mathcal{Y}^{HS,WS}] = Y_1 + Y_3 + Y_2 + Y_4 = C^t\left[\mathcal{D}\left(\{\mathcal{C}_{1e}\boldsymbol{y}\}_0^{N-1}\right)\right]C, \tag{16}$$

$$[\mathcal{Y}^{HS,WA}] = Y_1 - Y_3 + Y_2 - Y_4 = S^t\left[\mathcal{D}_1\left(S_{1e}\{\boldsymbol{y}\}_1^{N-1}\right)\right]C, \tag{17}$$

$$[\mathcal{Y}^{HA,WS}] = Y_1 + Y_3 - Y_2 - Y_4 = S^t\left[\mathcal{D}\left(\{C_{1e}\boldsymbol{y}\}_1^{N}\right)\right]S, \tag{18}$$

$$[\mathcal{Y}^{HA,WA}] = Y_1 - Y_3 - Y_2 + Y_4 = -C^t\left[\mathcal{D}_{-1}\left(S_{1e}\{\boldsymbol{y}\}_1^{N-1}\right)\right]S, \tag{19}$$

input vector, and $\{\boldsymbol{x}\}_a^b$ denotes the sequence $\{\boldsymbol{x}_a, \ldots, \boldsymbol{x}_b\}$. These relations are the basis of the filtering scheme developed in the next sections.

4 The 1-D Filtering Scheme

In this section, we develop the DCT-domain filtering scheme for the 1-D case. This is done in two steps, detailed in the following subsections: First, in Subsection 4.1, the resulting matrix equations are combined to obtain an efficient spatial domain filtering scheme. Then, in Subsection 4.2, the spatial domain scheme is converted into a DCT-domain scheme.

4.1 Filtering in the Spatial Domain

Suppose that a 1-D signal $x(n)$ is given, and we wish to perform linear filtering by a kernel $h(n)$, obtaining the output signal $w(n) = x(n) * h(n)$. Suppose also that $x(n)$ is given as a set of vectors $\{\boldsymbol{x}_i\}$, consisting of non-overlapping N-point segments of the input signal, and that we wish to obtain the output $w(n)$ in the same format, i.e., N-point segments $\{\boldsymbol{w}_i\}$. Let us assume that the support of the filter $h(n)$ is within the interval $[-N, N]$, and use the notation $h_n \triangleq h(n)$, for $|n| \leq N$. In this case, only the segments $\boldsymbol{x}_{i-1}$, $\boldsymbol{x}_i$, and $\boldsymbol{x}_{i+1}$, are needed in the calculation of the output segment $\boldsymbol{w}_i$. Specifically, one can express the filtering operation as follows:

$$\begin{aligned} \boldsymbol{w}_i &= H \begin{bmatrix} \boldsymbol{x}_{i-1} \\ \boldsymbol{x}_i \\ \boldsymbol{x}_{i+1} \end{bmatrix} \triangleq \begin{bmatrix} H_2^+ & \left(H_1^+ + H_1^-\right) & H_2^- \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{i-1} \\ \boldsymbol{x}_i \\ \boldsymbol{x}_{i+1} \end{bmatrix} & (20) \\ &= H_2^+ \boldsymbol{x}_{i-1} + H_1^+ \boldsymbol{x}_i + H_1^- \boldsymbol{x}_i + H_2^- \boldsymbol{x}_{i+1}, & (21) \end{aligned}$$

where H_1^+, H_2^+, H_2^-, and H_1^- are defined as follows.

$$H_2^+ = \begin{pmatrix} h_N & h_{N-1} & \cdots & h_2 & h_1 \\ 0 & h_N & \ddots & \ddots & h_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & h_N & h_{N-1} \\ 0 & 0 & \ddots & 0 & h_N \end{pmatrix}, \quad H_1^+ = \begin{pmatrix} \alpha \cdot h_0 & 0 & \cdots & 0 & 0 \\ h_1 & \alpha \cdot h_0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{N-2} & \ddots & \ddots & \alpha \cdot h_0 & 0 \\ h_{N-1} & h_{N-2} & \ddots & h_1 & \alpha \cdot h_0 \end{pmatrix}, \qquad (22)$$

[1]The ℓ^{th} off-diagonal of a square matrix $A = \{A_{ij}\}$ consists of the elements for which $j = i + \ell$.

$$H_1^- = \begin{pmatrix} \beta\cdot h_0 & h_{-1} & \cdots & h_{-(N-2)} & h_{-(N-1)} \\ 0 & \beta\cdot h_0 & \ddots & \ddots & h_{-(N-2)} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \beta\cdot h_0 & h_{-1} \\ 0 & 0 & \ddots & 0 & \beta\cdot h_0 \end{pmatrix}, \tag{23}$$

$$H_2^- = \begin{pmatrix} h_{-N} & 0 & \cdots & 0 & 0 \\ h_{-(N-1)} & h_{-N} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{-2} & \ddots & \ddots & h_{-N} & 0 \\ h_{-1} & h_{-2} & \ddots & h_{-(N-1)} & h_{-N} \end{pmatrix}, \tag{24}$$

where α is an arbitrary real number, and $\beta \triangleq 1-\alpha$. In the general case considered herein, the choice of α is immaterial from the aspects of computational efficiency. In certain special cases that will be studied later, however, the choice of α will be important.

Note that, if we set $y_i = h_i$ in (7), for $i = 1, \ldots, N-1$, $y_0 = 2\alpha h_0$, and $y_N = 2h_N$, then we obtain $Y_1 = H_1^+$ and $Y_2 = H_2^+\Phi$, where Φ is a "flipping matrix", given by:

$$\Phi = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix}. \tag{25}$$

It is easy to verify that post-multiplying a matrix by Φ reverses the order of its columns, and pre-multiplying a column vector by Φ reverses the order of its elements.

The above relationships between Y_1 and H_1^+, and between Y_2 and H_2^+ enable us to use eqs. (16)-(19) to rewrite H_1^+ and H_2^+ in terms of the DCT-IIe and DST-IIe transforms. Specifically, note that $Y_2 = (\mathcal{Y}^{\mathrm{HS,WS}} + \mathcal{Y}^{\mathrm{HS,WA}} - \mathcal{Y}^{\mathrm{HA,WS}} - \mathcal{Y}^{\mathrm{HA,WA}})/4$, and $Y_1 = (\mathcal{Y}^{\mathrm{HS,WS}} + \mathcal{Y}^{\mathrm{HS,WA}} + \mathcal{Y}^{\mathrm{HA,WS}} + \mathcal{Y}^{\mathrm{HA,WA}})/4$, and therefore:

$$H_2^+ = (C^t\boldsymbol{H}_{cc}^+C + S^t\boldsymbol{H}_{sc}^+C - S^t\boldsymbol{H}_{ss}^+S + C^t\boldsymbol{H}_{cs}^+S)\Phi, \tag{26}$$

$$H_1^+ = C^t\boldsymbol{H}_{cc}^+C + S^t\boldsymbol{H}_{sc}^+C + S^t\boldsymbol{H}_{ss}^+S - C^t\boldsymbol{H}_{cs}^+S, \tag{27}$$

where the specific values of the filtering kernels $\boldsymbol{H}_p^+$, $p \in \{cc, ss, cs, sc\}$, are given in the left side of Table 2.

$Y_3 = H_1^-$ and $Y_4 = H_2^-\Phi$, and by using the relations $Y_4 = (\mathcal{Y}^{\mathrm{HS,WS}} - \mathcal{Y}^{\mathrm{HS,WA}} - \mathcal{Y}^{\mathrm{HA,WS}} + \mathcal{Y}^{\mathrm{HA,WA}})/4$ and $Y_3 = (\mathcal{Y}^{\mathrm{HS,WS}} - \mathcal{Y}^{\mathrm{HS,WA}} + \mathcal{Y}^{\mathrm{HA,WS}} - \mathcal{Y}^{\mathrm{HA,WA}})/4$, we get:

$$H_2^- = (C^t \boldsymbol{H}_{cc}^- C - S^t \boldsymbol{H}_{sc}^- C - S^t \boldsymbol{H}_{ss}^- S - C^t \boldsymbol{H}_{cs}^- S)\Phi, \tag{28}$$

$$H_1^- = C^t \boldsymbol{H}_{cc}^- C - S^t \boldsymbol{H}_{sc}^- C + S^t \boldsymbol{H}_{ss}^- S + C^t \boldsymbol{H}_{cs}^- S, \tag{29}$$

where the values of $\boldsymbol{H}_p^-$, are shown in Table 2 as well.

A filtering scheme is then obtained on substituting eqs. (26)-(29) into eq. (21), and re-arranging the terms:

$$\begin{aligned} \boldsymbol{w}_i = \; & C^t \left[\boldsymbol{H}_{cc}^+ C(\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i-1}) - \boldsymbol{H}_{cs}^+ S(\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i-1}) + \right. \\ & \left. \boldsymbol{H}_{cc}^- C(\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i+1}) + \boldsymbol{H}_{cs}^- S(\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i+1}) \right] + \\ & S^t \left[\boldsymbol{H}_{ss}^+ S(\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i-1}) + \boldsymbol{H}_{sc}^+ C(\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i-1}) + \right. \\ & \left. \boldsymbol{H}_{ss}^- S(\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i+1}) - \boldsymbol{H}_{sc}^- C(\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i+1}) \right]. \end{aligned} \tag{30}$$

The implementation steps of the above scheme according to (30) are as follows. First, calculate $\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i-1}$, $\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i+1}$, $\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i-1}$, and $\boldsymbol{x}_i - \Phi\boldsymbol{x}_{i+1}$, which consist of "folding" the adjacent spatial segments $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_{i+1}$ onto the current one, $\boldsymbol{x}_i$. Notice that the order of the elements in the adjacent vectors is reversed in that operation. Next, calculate the DCT-II of the positively folded data, and the DST-II of the negatively folded data. Then, operate upon the transformed data by appropriately multiplying it by the kernel matrices. Notice that this consists of *element-by-element* multiplications between the input data and the significant diagonals of the kernels matrices. Finally, calculate the inverse DCT-II and inverse DST-II of some combinations of the multiplied data.

4.2 Conversion to the DCT Domain

The algorithm derived in Subsection 4.1 is suitable for applications where the input data is given in the spatial (or time) domain. However, in our application, the data is given in the DCT-IIe domain in the form of segments (vectors) $\boldsymbol{X}_i^c = C\boldsymbol{x}_i$, and therefore this algorithm, in its present form, is not suitable here. In this section, we modify the above algorithm to apply directly in the DCT domain.

We pre-multiply both sides of eq. (30) by C, in order to obtain the output in the DCT-IIe domain. Then, we use the linearity of matrix multiplication to write $C(\boldsymbol{x}_i + \Phi\boldsymbol{x}_{i-1}) = X_i^c + C\Phi\boldsymbol{x}_{i-1}$, proved: $C\Phi = \boldsymbol{\Psi} C$ and $S\Phi = \boldsymbol{\Psi} S$, where

$$\boldsymbol{\Psi} \stackrel{\triangle}{=} \mathcal{D}\left(\{(-1)^m\}_{m=0}^{N-1}\right). \tag{31}$$

After incorporating all the above steps in (30), we obtain the following DCT-domain scheme:

$$\begin{aligned} \boldsymbol{W}_i^c \;=\;& \boldsymbol{H}_{cc}^{+}(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^c) - \boldsymbol{H}_{cs}^{+}(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^s) + \\ & \boldsymbol{H}_{cc}^{-}(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i+1}^c) + \boldsymbol{H}_{cs}^{-}(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i+1}^s) + \\ & T^t\left[\boldsymbol{H}_{ss}^{+}(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^s) + \boldsymbol{H}_{sc}^{+}(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^c) + \right. \\ & \left. \boldsymbol{H}_{ss}^{-}(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i+1}^s) - \boldsymbol{H}_{sc}^{-}(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i+1}^c)\right], \end{aligned} \tag{32}$$

where $T \stackrel{\triangle}{=} SC^t$ is interpreted as the 1-D cosine to sine transform (CST) operator matrix (hence T^t is the sine to cosine (SCT) operator matrix), and $\{\boldsymbol{X}_i^s\}$ are the set of DST-IIe coefficients of the input data, i.e., $\boldsymbol{X}_i^s = S\boldsymbol{x}_i$, for all i.

Equation (32) represents the proposed filtering scheme in the 1-D case. The implementation of the scheme consists of the following steps. For every i, first, the DST segments $\boldsymbol{X}_{i-1}^s$, $\boldsymbol{X}_i^s$, and $\boldsymbol{X}_{i+1}^s$ must be calculated from the corresponding input DCT segments. Actually, one can assume that the values of $\boldsymbol{X}_{i-1}^s$ and $\boldsymbol{X}_i^s$ have already been calculated and stored while processing previous blocks $(i-1)$ and $(i-2)$. Therefore, only $\boldsymbol{X}_{i+1}^s$ has to be actually calculated (and stored for the next two iterations). This is done by pre-multiplying $\boldsymbol{X}_{i+1}^c$ by T. The second step is to create the "butterflies" corresponding to the expressions in the parentheses in (32). These correspond to the spatial domain "folding" in eq. (30). Note that reversing the order of elements in the spatial domain corresponds, in both the DCT and the DST domains, to a simple modulation (pre-multiplication by $\boldsymbol{\Psi}$). Next, the appropriate diagonal kernel filters operate upon the butterflies, and finally, part of the results is converted from the DST domain to the DCT domain using pre-multiplication by T^t, while the other part of the data is already in the DCT domain. Finally, these two parts are summed.

Equation (32) yields an efficient filtering scheme provided that the SCT can be implemented efficiently. This is true because all the $\boldsymbol{H}$-matrices in (32) are diagonal. Note that multiplication by $\boldsymbol{\Psi}$ is costless. Nevertheless, unlike the DCT coefficients, the DST coefficients are not available in advance, and hence must be calculated. Therefore, a fast CST, in addition to a fast SCT, is required. Fast routines for both transforms are derived in Section 4.4.

Of particular importance are special cases where the kernel is either symmetric/antisymmetric, causal/anticausal, or a combination of both.

In the *symmetric case*, $h_{-n} = h_n$, for $n = 1, \ldots, 8$. Here, by setting $\alpha = \beta = 1/2$ in (22) and (23), we obtain $\boldsymbol{H}_p^+ = \boldsymbol{H}_p^-$, for $p \in \{cc, ss, cs, sc\}$. Therefore, by defining $\boldsymbol{H}_p \stackrel{\Delta}{=} \boldsymbol{H}_p^+$, for all p, the scheme in (32) is reduced to:

$$\begin{aligned} \boldsymbol{W}_i^c &= \boldsymbol{H}_{cc}[2\boldsymbol{X}_i{}^c + \boldsymbol{\Psi}(\boldsymbol{X}_{i-1}^c + \boldsymbol{X}_{i+1}^c)] + \boldsymbol{H}_{cs}\boldsymbol{\Psi}(\boldsymbol{X}_{i-1}^s - \boldsymbol{X}_{i+1}^s) + \\ &\quad T^t \left\{\boldsymbol{H}_{ss}[2\boldsymbol{X}_i^s - \boldsymbol{\Psi}(\boldsymbol{X}_{i-1}^s + \boldsymbol{X}_{i+1}^s)] + \boldsymbol{H}_{sc}\boldsymbol{\Psi}(\boldsymbol{X}_{i-1}^c - \boldsymbol{X}_{i+1}^c)\right\}. \end{aligned} \tag{33}$$

Fig. 2(a) shows the flow graph of the above scheme.

By comparing (33) with (32), one observes that about half of the computations are saved when using a symmetric kernel. In the *antisymmetric* case, i.e., $h_{-n} = -h_n$, $n = 1, \ldots, 8$, $h_0 = 0$, a similar scheme is obtained, with some sign changes.

In the *causal case*, defined by $h_n = 0$ for $n < 0$, setting $\beta = 1 - \alpha = 0$ in (22) and (23), gives $\boldsymbol{H}_{cc}^- = \boldsymbol{H}_{cs}^- = \boldsymbol{H}_{ss}^- = \boldsymbol{H}_{sc}^- = \boldsymbol{0}$, where $\boldsymbol{0}$ is the 8×8 null matrix. Incorporating this fact into the filtering equation (32), one obtains

$$\begin{aligned} \boldsymbol{W}_i^c &= \boldsymbol{H}_{cc}^+(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^c) - \boldsymbol{H}_{cs}^+(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^s) + \\ &\quad T^t \left[\boldsymbol{H}_{ss}^+(\boldsymbol{X}_i^s - \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^s) + \boldsymbol{H}_{sc}^+(\boldsymbol{X}_i^c + \boldsymbol{\Psi}\boldsymbol{X}_{i-1}^c)\right], \end{aligned} \tag{34}$$

which is also a significant simplification. Fig. 2(b) shows the flow graph of the scheme. Notice that, in this case, $\boldsymbol{X}_{i+1}^p$, $p \in \{c, s\}$, are not needed in the computation, but only $\boldsymbol{X}_i^p$ and $\boldsymbol{X}_{i-1}^p$. The anticausal filtering scheme, where $h_n = 0$, for $n > 0$, is obtained similarly, by setting the $\boldsymbol{H}^+$-type matrices to zero, instead of the $\boldsymbol{H}^-$-type ones.

The best special case of the proposed scheme, in terms of complexity, is obtained with a 4-pixel delayed *causal-symmetric* filter, for which both the causality and the symmetry properties can be used to save computations. A k-pixel delayed causal symmetric filter will be defined as a causal filter $\{h_n\}_{n=0}^{2k}$ with $h_n = h_{2k-n}$ for all $0 \le n \le 2k$. Obviously, a causal symmetric filter is a delayed version of a non-causal filter that is symmetric about the origin. In the causal symmetric case, when $k = 4$, on the top of the simplification due to causality for $\alpha = 1$, we also have the even elements of the sequences $\mathcal{C}_{1e}\left[2h_0, h_1, \ldots, h_{N-1}, 2h_N\right]^t$ and $\mathcal{S}_{1e}\left[h_1, h_2, \ldots, h_{N-1}\right]^t$ equal to zero. Therefore, according to their definitions (see Table 2), the matrices $\boldsymbol{H}_p^+$, $p \in \{cc, ss, cs, sc\}$, have only 4 nonzero elements each. This case can be of much interest for fast symmetric convolution with a (see Discussion in Section 6.3).

4.4 Efficient CST and SCT Algorithms

In this subsection, we devise efficient CST and SCT algorithms, i.e., fast multiplication by T and T^t. We will assume $N = 8$, which is the case in JPEG/MPEG applications, but the algorithm derivations can be extended to any value of N.

The main idea is to factorize T into a product of sparse matrices. First, we shall use the following property relating the DST matrix to the DCT matrix:

$$S = \Phi C \mathbf{\Psi}, \tag{35}$$

where Φ and $\mathbf{\Psi}$ are defined in (25) and (31), respectively. This relation holds for any value of N.

Next, we consider the factorization of C that corresponds to the Winograd DCT, which is the fastest existing algorithm for 8-point DCT due to Arai, Agui, and Nakajima [11] (see also [12]). According to this factorization, C is represented as follows [12, pages 53-57]:

$$C = DPB_1B_2MA_1A_2A_3, \tag{36}$$

where operation by B_1, B_2, A_1, A_2, and A_3 requires a total of 26 additions, operation by M requires 5 multiplications and 3 additions, P is a permutation matrix, and D is a diagonal matrix given by:

$$D = \mathcal{D}\{0.3536, 0.2549, 0.2706, 0.3007, 0.3536, 0.4500, 0.6533, 1.2814\}. \tag{37}$$

Thus, we have

$$T = SC^t = \Phi C \mathbf{\Psi} C^t = \Phi DPB_1B_2MA_1A_2A_3\mathbf{\Psi}A_3^tA_2^tA_1^tM^tB_2^tB_1^tP^tD. \tag{38}$$

The proposed CST algorithm is based on the observation that the product

$$G \triangleq \frac{1}{2}MA_1A_2A_3\mathbf{\Psi}A_3^tA_2^tA_1^tM^t \tag{39}$$

$$G = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -a & 0 & b & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & c \\ 0 & 0 & 0 & 0 & \frac{b}{2} & 0 & \frac{a}{2} & 1 \\ 0 & -a & 0 & \frac{b}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & b & 0 & \frac{a}{2} & 0 & 0 & 0 & 0 \\ 1 & 1 & c & 1 & 0 & 0 & 0 & 0 \end{pmatrix},$$

where $a = 2.6132$, $b = 1.0824$, and $c = 0.7071$.

Therefore, a fast CST is obtained by following the decomposition:

$$T = \hat{D}(\Phi\boldsymbol{\tau})(2D), \tag{40}$$

where $\hat{D} \overset{\Delta}{=} \Phi D \Phi$ is a diagonal matrix with the same diagonal elements as D but in reversed order, and:

$$\boldsymbol{\tau} = PB_1B_2GB_2^tB_1^tP^t. \tag{41}$$

A flow graph of an efficient implementation of $\boldsymbol{\tau}$ is shown in Fig. 3. One can observe that it requires 8 multiplications, 28 additions, and 2 right-shifts (multiplications by 1/2). The complete CST algorithm requires 16 more multiplications, corresponding to the multiplications by the diagonal matrices $\hat{D}$ and $(2D)$. The SCT operation is expressed by $T^t = (2D)(\boldsymbol{\tau}\Phi)\hat{D}$, and requires the same number of operations as the CST. Notice that the straightforward implementation of T by performing IDCT followed by DST ($\Phi C \boldsymbol{\Psi}$) requires 26 multiplications and 58 additions, which is 2 multiplications and 30 additions more than the proposed algorithm.

The authors of the fast DCT algorithm pondered [11] that, if the output coefficients are to be quantized (like in a JPEG or MPEG coding process), then the scaling matrix D can be combined with the quantization table, thus saving multiplications. Similar manipulations can be done regarding the above fast CST and SCT algorithms. In Section 5.1 below, we describe how to combine the matrices $\hat{D}$ and $2D$ with the quantization and dequantization tables, as well as with the filtering kernels, so that only the unscaled transforms $\Phi\boldsymbol{\tau}$ and $\boldsymbol{\tau}\Phi$ have to be performed in place of the complete CST and SCT, respectively.

The 1-D scheme derived in Section 4 is extended here to 2-D signals. The purpose of this extension is to apply the scheme to DCT blocks obtained during the decoding of JPEG or MPEG bitstreams. Therefore, we suppose now that the input and output data are sets of 8×8 DCT blocks, which we denote by $\{\boldsymbol{X}^c_{i,j}\}$ and $\{\boldsymbol{W}^c_{i,j}\}$, respectively.

We will consider here the case where the 2-D filter kernel is separable, i.e., the corresponding spatial filtering operation can be written in the form:

$$\boldsymbol{w}_{i,j} = V \begin{bmatrix} \boldsymbol{x}_{i-1,j-1} & \boldsymbol{x}_{i-1,j} & \boldsymbol{x}_{i-1,j+1} \\ \boldsymbol{x}_{i,j-1} & \boldsymbol{x}_{i,j} & \boldsymbol{x}_{i,j+1} \\ \boldsymbol{x}_{i+1,j-1} & \boldsymbol{x}_{i+1,j} & \boldsymbol{x}_{i+1,j+1} \end{bmatrix} H^t, \tag{42}$$

where $\{\boldsymbol{x}_{i,j}\}$ and $\{\boldsymbol{w}_{i,j}\}$ are, respectively, the input and output signals in the spatial domain, H is a 8×24 filter matrix, defined as in (20)-(24) for a given sequence $\{h_n\}$, and V is similarly defined for a given $\{v_n\}$. Thus, we assume that the filter support is of size up to 17×17 pixels.

In this case, the 2-D filtering can be implemented by means of the proposed 1-D scheme, by simply applying the latter first to the input DCT block columns, and then to the rows of the resulting blocks. Specifically, the first pass (column processing) corresponds to the following spatial processing:

$$\boldsymbol{z}_{i,j} \triangleq V \begin{bmatrix} \boldsymbol{x}_{i-1,j} \\ \boldsymbol{x}_{i,j} \\ \boldsymbol{x}_{i+1,j} \end{bmatrix}, \tag{43}$$

and the second pass (row processing) corresponds to:

$$\boldsymbol{w}_{i,j} = \left(H \begin{bmatrix} \boldsymbol{z}^t_{i,j-1} \\ \boldsymbol{z}^t_{i,j} \\ \boldsymbol{z}^t_{i,j+1} \end{bmatrix} \right)^t. \tag{44}$$

The above column and row passes are performed in the DCT/DST domains using the transform domain versions of the V and H kernels. Therefore, the column processing produces $\{\boldsymbol{Z}^c_{i,j}\}$, which are the 2-D DCT version of the blocks $\{\boldsymbol{z}_{i,j}\}$, while the row processing produces the output $\{\boldsymbol{W}^c_{i,j}\}$.

Although straightforward, a few comments regarding the separable 2-D extension of the 1-D scheme are in order:

1. The 8×8 input blocks $\{\boldsymbol{X}^c_{i,j}\}$ are assumed to be given in a raster-scan order (i.e., left-to-right and top-to-bottom in respect to the image frame). Therefore, at the step of the algorithm

blocks $\boldsymbol{W}^c_{u,v}$, for $u < i$, or $u = i$ and $v < j$, have already been calculated.

2. Because of the above, during the column pass corresponding to the step (i, j) of the algorithm, $\boldsymbol{Z}^c_{i,j}$ and $\boldsymbol{Z}^c_{i,j-1}$ do not have to be calculated, since they have been previously computed (and assumed stored) in the two previous steps $(i-1, j)$ and $(i-2, j)$. Therefore, only $\boldsymbol{Z}^c_{i,j+1}$ has to be calculated.

3. The 2-D DST coefficients of the input blocks are never calculated during the algorithm. During the column pass, we obtain a 2-D mixed DST/DCT domain block $\boldsymbol{X}^{sc}_{i,j} \triangleq T\boldsymbol{X}^c_{i,j}$, which corresponds to the DCT of the spatial rows and to the DST of the columns. Similarly, during the row processing, the algorithm operates on 2-D mixed DCT/DST domain blocks $\boldsymbol{Z}^{cs}_{i,j} \triangleq \boldsymbol{Z}^c_{i,j}T^t$.

4. Similarly to the situation in item 2 above, at a given step (i, j), only the mixed DST/DCT block $\boldsymbol{X}^{sc}_{i+1,j+1}$ needs to be calculated at the beginning of the first pass, since $\boldsymbol{X}^{sc}_{i,j+1}$ and $\boldsymbol{X}^{sc}_{i-1,j+1}$ were previously calculated and are assumed to be stored. In the same way, $\boldsymbol{Z}^{cs}_{i,j}$ and $\boldsymbol{Z}^{cs}_{i,j-1}$ are also available, so only the 2-D mixed DCT/DST block $\boldsymbol{Z}^{cs}_{i,j+1}$ has to be calculated at the beginning of the row pass.

5. The 2-D mixed DST/DCT (resp. DCT/DST) coefficients can be calculated using the same 1-D CST algorithm as before by applying the latter to each separate column (resp. row) of the input 2-D DCT block. Similarly, the same 1-D SCT algorithm as before can be used within the filtering process.

5.1 Combining the Scaling Matrices with the Kernels and Quantizers

Typically, the input DCT blocks are obtained from the dequantization step of a JPEG decoding procedure, whereas the output is fed to the quantizer of a JPEG encoder (as seen in Fig. 1(b)). In this case, substantial savings in multiplications can be obtained by absorbing the scaling matrices $\hat{D}$ and $2D$ in the kernel matrices $\boldsymbol{V}^r_i$ and $\boldsymbol{H}^r_i$, $i \in \{cc, ss, cs, sc\}$ and $r \in \{+, -\}$, and in the quantization/dequantization look-up tables.

We assume, therefore, that each input block $\boldsymbol{X}^c_i$ is the result of element-by-element multiplication of a certain quantized block by a dequantization matrix Q^d. Moreover, the output $\boldsymbol{W}^c_i$ serves as an input to a quantizer that divides it by a quantization matrix Q^q (usually identical to Q^d) element-by-element.

dures, corresponding to the multiplication by T and T^t, respectively, by their unscaled versions, corresponding to the multiplication by $\Phi\boldsymbol{\tau}$ and $\boldsymbol{\tau}\Phi$, respectively.

By extending eq. (32) to the 2-D separable case considered above, and using the relationship between T and $\boldsymbol{\tau}$ (eq. (40)), one can show that the following further modifications should be performed in order to preserve the validity of the algorithm.

- One has to alter the quantization/dequantization tables in the following way:

$$Q^d \leftarrow 4DQ^dD, \qquad Q^q \leftarrow D^{-1}Q^qD^{-1}/4. \tag{45}$$

  Note that, given the same quantized input data as before, the dequantized input blocks are now given by $4D\boldsymbol{X}_i^cD$, instead of $\boldsymbol{X}_i^c$. Similarly, assuming that the the same quantized output data is produced as before, the output filtered blocks are now given by $D^{-1}\boldsymbol{W}_i^cD^{-1}/4$.

- The kernel matrices must be modified as follows:

$$\begin{aligned} \boldsymbol{A}_{cc} \leftarrow D^{-1}\boldsymbol{A}_{cc}D^{-1}/4, &\qquad \boldsymbol{A}_{cs} \leftarrow D^{-1}\boldsymbol{A}_{cs}\hat{D}/2, \\ \boldsymbol{A}_{ss} \leftarrow \hat{D}\boldsymbol{A}_{ss}\hat{D}, &\qquad \boldsymbol{A}_{sc} \leftarrow \hat{D}\boldsymbol{A}_{sc}D^{-1}/2, \end{aligned} \tag{46}$$

  where $\boldsymbol{A}$ denotes any of the symbols: $\{\boldsymbol{V}^+, \boldsymbol{V}^-, \boldsymbol{H}^+, \boldsymbol{H}^-\}$.

The modified algorithm is significantly more efficient than the original one since it avoids the scaling steps involving the matrices $\hat{D}$ and D without any degradation at the output.

6 Complexity Analysis

In this section, we compare the proposed filtering scheme and previously reported schemes in terms of computational complexity. Two sets of results will be presented: Theoretical complexity, expressed in terms of the number of multiplications m and additions a, required for processing of each 8×8 output block, and actual running times in computer simulations with real grayscale images.

6.1 Data Sparseness

An important factor to be taken into account in the implementation is that of typical sparseness of the quantized DCT input data blocks. We define a DCT coefficient block as *sparse* if only its elements. A very high percentage of the DCT blocks usually satisfy this requirement, which is fairly easy to check directly in the compressed format.

Incorporating the sparseness above defined assumption into the modified CST algorithm, proposed in section 4.4, results in 7 multiplications and 17 additions, as opposed to 8 multiplications and 28 additions in the general case. On the top of this advantage, sparseness also saves multiplications and additions during the DCT-domain filtering process itself, since null coefficients need not be processed.

Unfortunately, DCT-domain block sparseness is not generally preserved after conversion to the DST domain. On the other hand, mixed DCT/DST and DCT/DST blocks have half the "amount of sparseness" of that of sparse DCT blocks. Specifically, if the corresponding DCT block is sparse, then the last 4 rows (resp. columns) of the mixed DCT/DST (resp. DST/DCT) block are null. These facts are accounted for in the following efficiency analysis.

6.2 Computational Efficiency and Comparison to Other Approaches

6.2.1 Theoretical Analysis

In the complexity analysis presented here, the filtering cases are given in terms of *kernel type* and *data sparseness.* The kernel type is either *general*, *symmetric*, *causal*, or *causal-symmetric*, where each type is related to a different implementation of the proposed algorithm, as detailed in Section 4.3. For causal or causal-symmetric kernels, *non-sparse data* refers to the case where at least one of the 4 causal input blocks $\boldsymbol{X}^c_{u,v}$, $i-1 \leq u, v \leq i$, is not sparse, whereas *sparse data* means that all the 4 blocks are sparse. For general or symmetric kernels, these definitions are the same, but w.r.t. the nine input blocks $i-1 \leq u, v \leq i+1$.

The computational complexity of the proposed 2-D separable filtering scheme has been calculated for the different filtering cases, and are presented in Table 3. Computation and implementation details for the causal-symmetric case can be found in Table 4. The other cases have similar derivations, which can be found in [13].

Let us compare the complexity of the proposed scheme to that of three other approaches:

1. **The spatial domain approach**, where the input DCT blocks are first transformed to the spatial domain, then spatial convolution is applied, and finally, the result is re-transformed to the DCT domain. We will assume here that the same fast DCT and IDCT algorithms as the

algorithms require 5 multiplications and 29 additions each, assuming that the scaling matrix D is combined with the quantization/dequantization tables. For non-symmetric filtering, the spatial domain approach requires $(128L + 160)$ multiplications and $(128L + 800)$ additions per 8×8 output block, where $L \times L$ is the size in pixels of the region of support of the kernel. For symmetric filtering, $(64L + 224)$ multiplications and $(128L + 800)$ additions are required. These results hold regardless of the amount of sparseness in the data.

2. **The Pipeline approach** by Lee and Lee, presented in [7]. In this approach, the product of the operator matrices corresponding to the IDCT, the convolution, and the DCT is pre-computed, and the resulting combined matrix is then applied directly in the DCT domain. In this operation, the contributions of the neighboring blocks are incorporated similarly as in (20). Complexity results for this approach are given in Table 5.

3. **The Butterfly approach** presented by Merhav and Bhaskaran in [14]. This approach is similar to the Pipeline approach, but certain butterflies are created on the input data, prior to the DCT domain filtering. This was shown [14] to cause the filtering kernels to become much sparser than in the Pipeline approach, which leads to savings in complexity. The butterfly approach was developed for symmetric kernels only, in which case careful implementation leads to a complexity of $1152m + 1536a$, for non-sparse data, and $432m + 528a$, for sparse data, for any filter size up to 17×17 pixels.

In order to compare the above approaches, we consider the overall number of operations required by each approach. To this end, we will assume a processor for which each multiplication is roughly equivalent to 3 additions on the average (e.g., PA-RISC processor). For this case, the number of operations for the different approaches is summarized in the graphs shown in Figure 4. In these graphs, six schemes are compared for each kernel type: The spatial domain scheme for small, medium, and large size kernels, the Pipeline, the proposed, and the Butterfly schemes. For general and symmetric kernels, the terms small, medium, and large size kernels refer to kernels of sizes 3×3, 9×9, and 17×17, respectively. For causal or causal-symmetric kernels, they refer to the sizes 3×3, 5×5, and 9×9. Notice that the complexity of the spatial domain scheme does not depend on data sparseness, whereas that of the other schemes do not depend on the kernel size. Notice also that the Butterfly scheme is considered only for symmetric kernels.

From the above graphs, one can conclude the following:

causal or causal-symmetric filtering (see discussion below), then the proposed scheme is the best option in terms of complexity: In the causal case, it saves 17-61% of the number of operations required by any other scheme, depending on the kernel size. In the causal-symmetric case, the range becomes 35-64%.

- For sparse data and general kernel, the proposed scheme is the best one for medium-size and large kernels.

- If the data is not sparse, then the proposed scheme is the most efficient DCT-domain approach. It is also more efficient than the spatial domain approach for large kernels, and also for medium kernels in the symmetric and causal-symmetric cases.

6.2.2 Real-Data Simulations

The causal-symmetric version of the proposed algorithm and the spatial domain approach were implemented in C, and applied to several 512×512 grayscale images on an HP J-200 workstation. Sharpening filters were used, with sizes 3×3, 5×5, and 9×9. The algorithm addresses each input block as if it is non-sparse, but eventual sparseness contributes to complexity reduction since multiplications and additions by zero usually have low computation cost. The scaling matrices were embedded in quantization/dequantization in both schemes. The results are given in Table 6, and one can see that they roughly agree with the theoretical analysis.

6.3 Causal Versus Noncausal Kernels

The proposed approach is much more efficient in the causal (resp. causal-symmetric) case than in the general (resp. symmetric) case, as seen in Figure 4. Also, the causal versions use half of the memory that is used in the noncausal versions of the approach.

However, causal kernels are seldom encountered in image processing applications, while the use of symmetric kernels is fairly common. How could one take advantage of the efficiency of the causal versions of the algorithm to implement a noncausal filtering?

If a given symmetric 2-D filter is not longer than 9 taps in each dimension, then it can always be trivially transformed into a causal-symmetric kernel, simply by applying a 4-pixel shift (in each dimension) to its spatial coefficients. In this case, the use of this causal-symmetric version of an original symmetric kernel, instead of the symmetric kernel itself, results in a 4-pixel shift of the those involving human visualization. On the other hand, as mentioned above, it leads to great savings in computations (about 36%) and in memory requirements (50%).

Similar considerations apply also for using causal kernels instead of general kernels. In this case, on the other hand, the shift is not restricted to 4 pixels; the smallest number of pixels that turns the kernel into a causal kernel can be used.

7 Conclusion

In this work, we propose an efficient filtering scheme, to be applied directly to DCT data blocks given in JPEG/MPEG applications. The scheme also outputs the filtered data in the same DCT format. It is based on the convolution-multiplication properties of the discrete trigonometric transforms, and it requires the calculation of DST coefficients, which are used together with the DCT coefficients in the filtering process. A fast CST (cosine to sine transform) was thus derived, to reduce the computational overhead of this operation. No zero-padding of the input data is required or assumed. Four versions of the approach are proposed: For general, symmetric, causal, and causal-symmetric kernels.

The 2-D separable version of the proposed algorithm is compared to previous DCT-domain approaches and the straightforward approach of converting back to the pixel domain, convolving, and re-transforming to the DCT domain. It was demonstrated that, by taking into account the typical sparseness of the input DCT-data, the proposed algorithm provides the best results for symmetric filtering, if a 4-pixel translation of the image in both directions is allowed (causal-symmetric filtering). In this case, 35-64% of the computations are saved, depending on the kernel size. The approach is also typically the most efficient for long or medium-length, non-symmetric, kernels. Twice as much memory is required by the proposed algorithm in comparison to the others, since DCT *and* DST coefficients are to be temporarily stored, instead of only the spatial data (as in the straightforward approach) or only the DCT coefficients (as in the DCT-domain schemes).

A nonseparable version of the algorithm was also derived, and is now under study. It is based on a direct extension of the DTT CMP's to 2-D signals. The proposed scheme is expected to be even more efficient (relatively to the spatial domain approach) in the nonseparable case than in the separable one.

[1] W. H. Chen and S. C. Fralick, "Image enhancement using cosine transform filtering," *Image Sci. Math. Symp.*, Monterey, 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. Chitprasert 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] S. A. Martucci, "Digital filtering of images using the discrete sine or cosine transform," *Optical Engineering*, Vol. 35, no. 1, pp. 119-127, January 1996.

[6] V. Sánchez, P. García, A.M. Peinado, J.C. Segura, and A.J. Rubio, "Diagonalizing Properties of the Discrete Cosine Transforms," *IEEE Trans. on Signal Processing*, Vol. 43, no. 11, pp. 2631-2641, November 1995.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] R. Kresch and N. Merhav, "Fast DCT domain filtering using the DCT and the DST," HPL Technical Report #HPL-95-140, December 1995.

[14] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT domain filtering," submitted for publication. Also, HPL Technical Report #HPL-95-56, May 1995.

		input index ranges		output			
ε_a	ε_b	$x(n)$	$y(n)$	index range	$\mathcal{T}_a$	$\mathcal{T}_b$	$\mathcal{T}_c$
HSHS	WSWS	$0 \to N-1$	$0 \to N$	$0 \to N-1$	$\mathcal{C}_{2e}$	$\mathcal{C}_{1e}$	$\mathcal{C}_{2e}$
HAHA	WSWS	$0 \to N-1$	$0 \to N$	$0 \to N-1$	$\mathcal{S}_{2e}$	$\mathcal{C}_{1e}$	$\mathcal{S}_{2e}$
HSHS	WAWA	$0 \to N-1$	$1 \to N-1$	$0 \to N-1$	$\mathcal{C}_{2e}$	$\mathcal{S}_{1e}$	$\mathcal{S}_{2e}$
HAHA	WAWA	$0 \to N-1$	$1 \to N-1$	$0 \to N-1$	$\mathcal{S}_{2e}$	$\mathcal{S}_{1e}$	$-\mathcal{C}_{2e}$

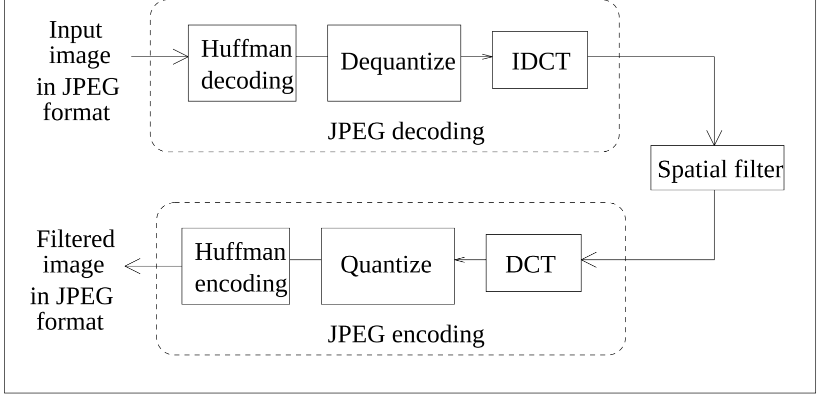
Table 1: Convolution-multiplication properties of DCT and DST of types Ie and IIe.

p	$\boldsymbol{H}_p^+$	$\boldsymbol{H}_p^-$
cc	$\frac{1}{4}\mathcal{D}\left(\{\mathcal{C}_{1e}[2\alpha h_0, h_1, \ldots, h_{N-1}, 2h_N]\}_0^{N-1}\right)$	$\frac{1}{4}\mathcal{D}\left(\{\mathcal{C}_{1e}[2\beta h_0, h_{-1}, \ldots, h_{-(N-1)}, 2h_{-N}]\}_0^{N-1}\right)$
ss	$\frac{1}{4}\mathcal{D}\left(\{\mathcal{C}_{1e}[2\alpha h_0, h_1, \ldots, h_{N-1}, 2h_N]\}_1^{N}\right)$	$\frac{1}{4}\mathcal{D}\left(\{\mathcal{C}_{1e}[2\beta h_0, h_{-1}, \ldots, h_{-(N-1)}, 2h_{-N}]\}_1^{N}\right)$
cs	$\frac{1}{4}\mathcal{D}_{-1}\left(\mathcal{S}_{1e}[h_1, \ldots, h_{N-1}]\right)$	$\frac{1}{4}\mathcal{D}_{-1}\left(\mathcal{S}_{1e}[h_{-1}, \ldots, h_{-(N-1)}]\right)$
sc	$\frac{1}{4}\mathcal{D}_{1}\left(\mathcal{S}_{1e}[h_1, \ldots, h_{N-1}]\right)$	$\frac{1}{4}\mathcal{D}_{1}\left(\mathcal{S}_{1e}[h_{-1}, \ldots, h_{-(N-1)}]\right)$

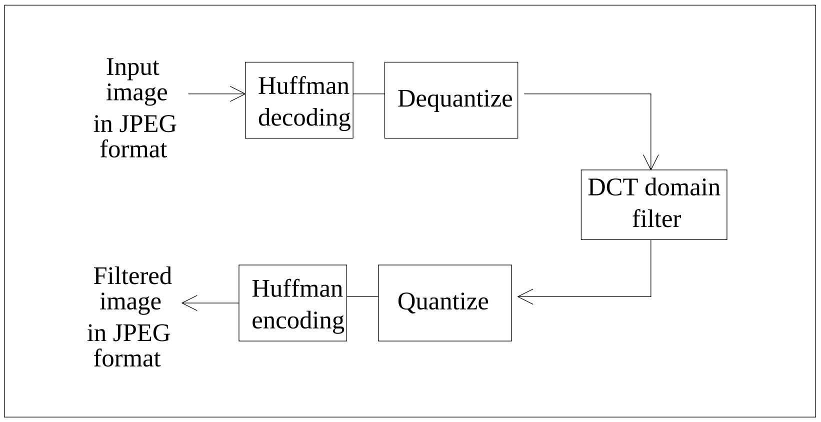
Table 2: Filter kernels.

Complexity of the Proposed Scheme				
	Data Type			
	Non-Sparse		Sparse	
Kernel Type	m	a	m	a
General	1216	2688	716	1516
Symmetric	736	1984	448	1124
Causal	736	1728	448	980
Causal-Symmetric	512	1280	296	688

Table 3: Computational efficiency of the proposed 2-D separable scheme (according to data and kernel types) in terms of number of multiplications and additions required for processing each 8×8 output block.

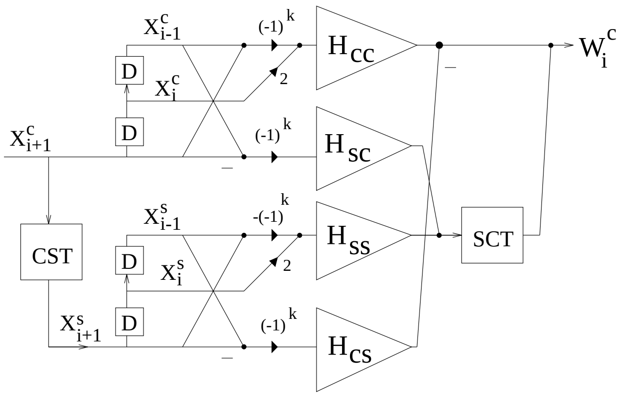


(a)

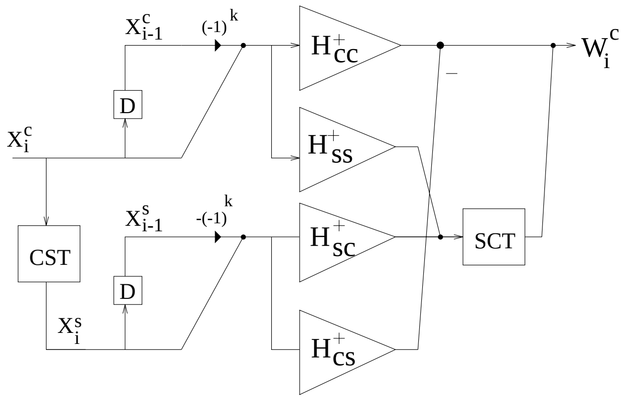


(b)

Figure 1: Spatial filtering of compressed images: (a) Straightforward spatial approach, and (b) direct DCT-domain approach.



(a)



(b)

Figure 2: Flow graph of the proposed filtering scheme for (a) symmetric kernels and (b) causal kernels. The lines in the graph refer to N-point vectors, the bullets to element-by-element addition, the small triangles to element-by-element multiplication, the large triangles to matrix multiplication, and the "D" boxes to a 1-step delay.

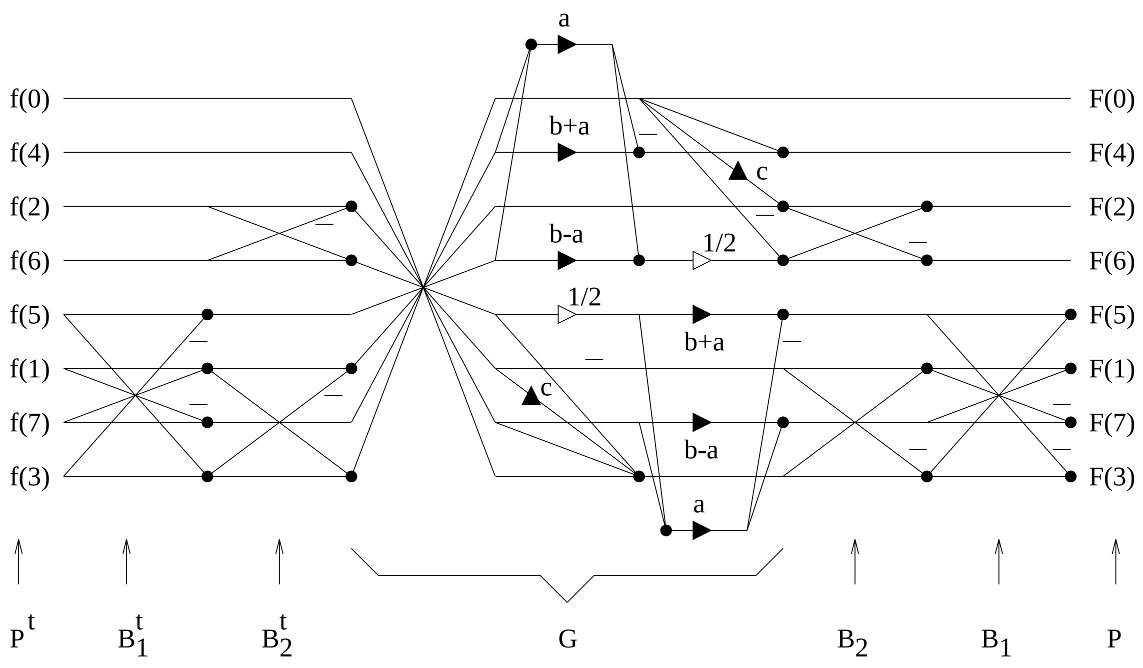


Figure 3: Flow graph of the core operations ($\boldsymbol{\tau}$) for the fast CST and SCT. The bullets, black triangles, and white triangles depict additions, multiplications, and right shifts, respectively. The sequences $f(n)$ and $F(n)$, $n = 0, \ldots, 7$, denote the input and output vectors, respectively.

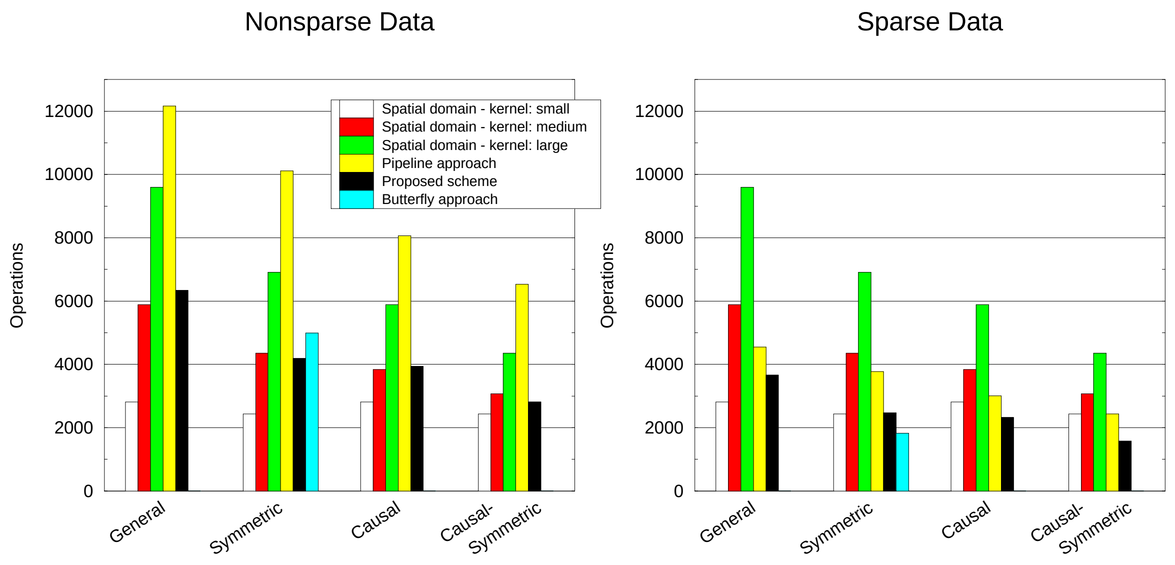


Figure 4: Computational efficiency of the proposed 2-D separable scheme as compared to the spatial domain and other DCT-domain schemes. The results are given in terms of number of operations *op*, required for processing each 8×8 output block. Number of operations is calculated here assuming each multiplication to be equivalent to three additions, i.e., $op = 3m + a$.

Loop commands for each "step (i,j)"	Complexity	
	Non-sparse data	Sparse data
$\boldsymbol{X}^{sc}_{i-1,j} \leftarrow \boldsymbol{X}^{sc}_{i,j}$ $\boldsymbol{Z}^{c}_{i,j-1} \leftarrow \boldsymbol{Z}^{c}_{i,j}$ $\boldsymbol{Z}^{cs}_{i,j-1} \leftarrow \boldsymbol{Z}^{cs}_{i,j}$		
$\boldsymbol{X}^{sc}_{i,j} \leftarrow$ Column-UCST$(\boldsymbol{X}^{c}_{i,j})$	$8 \times (8m + 28a) =$ $64m + 224a$	$4 \times (7m + 17a) =$ $28m + 68a$
$E_1 \leftarrow \boldsymbol{X}^{c}_{i,j} + \boldsymbol{\Psi}\boldsymbol{X}^{c}_{i-1,j}$	$64a$	$16a$
$E_2 \leftarrow \boldsymbol{X}^{sc}_{i,j} - \boldsymbol{\Psi}\boldsymbol{X}^{sc}_{i-1,j}$	$64a$	$32a$
Odd rows of $B \leftarrow -\boldsymbol{V}^{+}_{sc}E_1$	$32m$	$8m$
Even rows of $B \leftarrow \boldsymbol{V}^{+}_{ss}E_2$	$32m$	$16m$
$B \leftarrow$ Column-USCT(B)	$8 \times (8m + 28a) =$ $64m + 224a$	$4 \times (8m + 28a) =$ $32m + 112a$
Odd rows of $\boldsymbol{Z}^{c}_{i,j} \leftarrow \boldsymbol{V}^{+}_{cc}E_1 +$ Odd rows of B	$32m + 32a$	$8m + 8a$
Even rows of $\boldsymbol{Z}^{c}_{i,j} \leftarrow \boldsymbol{V}^{+}_{cs}E_2 +$ Even rows of B	$32m + 32a$	$16m + 16a$
$\boldsymbol{Z}^{cs}_{i,j} \leftarrow$ Row-UCST$(\boldsymbol{Z}^{c}_{i,j})$	$8 \times (8m + 28a) =$ $64m + 224a$	$8 \times (7m + 17a) =$ $28m + 68a$
$E_1 \leftarrow \boldsymbol{Z}^{c}_{i,j} + \boldsymbol{Z}^{c}_{i,j-1}\boldsymbol{\Psi}$	$64a$	$32a$
$E_2 \leftarrow \boldsymbol{Z}^{cs}_{i,j} - \boldsymbol{Z}^{cs}_{i,j-1}\boldsymbol{\Psi}$	$64a$	$64a$
Odd columns of $B \leftarrow -E_1(\boldsymbol{H}^{+}_{sc})^t$	$32m$	$16m$
Even columns of $B \leftarrow E_2\boldsymbol{H}^{+}_{ss}$	$32m$	$32m$
$B \leftarrow$ Row-USCT(B)	$8 \times (8m + 28a) =$ $64m + 224a$	$8 \times (8m + 28a) =$ $64m + 224a$
Odd columns of $\boldsymbol{W}^{c}_{i,j} \leftarrow E_1\boldsymbol{H}^{+}_{cc} +$ Odd columns of B	$32m + 32a$	$16m + 16a$
Even columns of $\boldsymbol{W}^{c}_{i,j} \leftarrow E_2(\boldsymbol{H}^{+}_{cs})^t +$ Even columns of B	$32m + 32a$	$32m + 32a$
Total	$512m + 1280a$	$296m + 688a$

Table 4: Implementation of the proposed scheme for causal-symmetric filtering. Embedding of scaling matrices in quantization/dequantization tables and filtering kernels is assumed. Column-UCST (resp. USCT) means performing 1-D unscaled CST, $\Phi\boldsymbol{\tau}$ (resp. 1-D unscaled SCT, $\boldsymbol{\tau}\Phi$) of each separate column of the input block. Similarly for Row-UCST and Row-USCT. E_1, E_2, and B denote 8×8 buffers. Even and odd columns/rows refer to the indexing $\{1, \ldots, 8\}$.

Complexity of the Pipeline Scheme				
	Data Type			
	Non-Sparse		Sparse	
Kernel Type	m	a	m	a
General	3072	2944	1152	1088
Symmetric	2560	2432	960	896
Causal	2048	1920	768	704
Causal-Symmetric	1664	1536	624	560

Table 5: Computational efficiency of the Pipeline scheme, proposed in [7].

	Spatial domain scheme			Proposed scheme
	3×3	5×5	9×9	
Average running time (secs.)	2.55	2.74	3.18	2.35

Table 6: Simulation results on 512×512 grayscale images. The running times correspond to processing only; I/O procedures were not considered.