A Fast Scheme for Image Size Change in the Compressed Domain

Rakesh Dugad, *Student Member, IEEE*, and Narendra Ahuja, *Fellow, IEEE*

***Abstract*—Given a video frame in terms of its 8×8 block-DCT coefficients, we wish to obtain a downsized or upsized version of this frame also in terms of 8×8 block-DCT coefficients. The DCT being a linear unitary transform is distributive over matrix multiplication. This fact has been used for downsampling video frames in the DCT domain. However, this involves matrix multiplication with the DCT of the downsampling matrix. This multiplication can be costly enough to trade off any gains obtained by operating directly in the compressed domain. We propose an algorithm for downsampling and also upsampling in the compressed domain which is computationally much faster, produces visually sharper images, and gives significant improvements in PSNR (typically 4-dB better compared to bilinear interpolation). Specifically the downsampling method requires 1.25 multiplications and 1.25 additions per pixel of original image compared to 4.00 multiplications and 4.75 additions required by the method of Chang *et al*. Moreover, the downsampling and upsampling schemes combined together preserve all the low-frequency DCT coefficients of the original image. This implies tremendous savings for coding the difference between the original frame (unsampled image) and its prediction (the upsampled image). This is desirable for many applications based on scalable encoding of video. The method presented can also be used with transforms other than DCT, such as Hadamard or Fourier.**

***Index Terms*—Compressed domain processing, DCT, downsampling, image size change, superresolution, upsampling.**

I. Introduction

DUE to the advances in digital signal processing and digital networks, more and more video data is available today in digital format. For economy of storage and transmission, digital video is typically stored in compressed format. However, for many applications, one needs to produce a compressed bitstream containing the video at a different resolution than the original bitstream. For example, for browsing a remote video database, it would be more economical to send low-resolution versions of the video clips to the user and then, if there is interest, progressively enhance the resolution. The same is true about browsing a remote image database. A typical video-conferencing application requires compositing video bitstreams at different resolutions into one bitstream containing the video frames from all bitstreams but possibly at lower or higher resolutions. Similarly, for transmitting video over dual-priority networks, we can transmit a low-resolution version of the video over the high-priority channel and an enhancement layer over the low-priority channel. A solution to transmitting video over bandwidth-constrained channels is to transmit a low-resolution version of the video. Spatial scalability is also used in HDTV for maintaining compatibility with standards other than MPEG-2 (e.g., MPEG-1) and to allow for varied bitrate and computational capabilities of different users. Changing the size of a video frame would also be required to convert between various digital TV standards like standard TV and HDTV, and also to fit the incoming video frame onto the user's TV screen.

Since the original video data is typically stored in compressed format (use of 8×8 block-DCT along with motion compensation for compression is most widespread) and the scaled (in size) version is also typically required in (the same) compressed format, we see that there is a need for changing the size of a video frame in the compressed domain. The straightforward approach of decompressing, carrying out the downsampling in spatial domain and then recompressing, is computationally too intensive for real-time operation on currently available workstations. Hence, recently there has been much work done [1]–[5] in carrying out downsampling, as well as other operations, on video sequences directly in the compressed domain without requiring decompression and recompression.

Let c_1, c_2, c_3, c_4 denote four adjacent 8×8 blocks in the spatial domain as shown in Fig. 1(a). Consider replacing each 2×2 block (of each of the 8×8 blocks) by its average to get the downsampled 8×8 block c shown in Fig. 1. This downsampling operation can also be represented in matrix notation as follows [4], [1], [2]:

$$c = \sum_{i=1}^{4} h_i c_i g_i \tag{1}$$

where the downsampling filters h_i, g_i are given by

$$\begin{aligned} h_2 = g_1^t = g_3^t = h_1 &= \begin{bmatrix} \mathbf{u}_{4\times 8} \\ \mathbf{o}_{4\times 8} \end{bmatrix} \\ h_4 = g_2^t = g_4^t = h_3 &= \begin{bmatrix} \mathbf{o}_{4\times 8} \\ \mathbf{u}_{4\times 8} \end{bmatrix} \end{aligned} \tag{2}$$

where $\mathbf{u}_{4\times 8}$ is defined in Fig. 1(b) and $\mathbf{o}_{4\times 8}$ is a 4×8 zero matrix. Now consider implementing the same operation directly in the DCT domain, i.e., consider obtaining $\mathrm{DCT}(c)$ given, $\mathrm{DCT}(c_i)$ for $i = 1$ to 4. Most previous approaches for downsampling in the compressed domain (say, in the DCT domain) rely on the fact that the DCT, being a linear orthonormal transform, is distributive over matrix multiplication [1], [2],

Manuscript received April 8, 1999; revised October 16, 2000. This work was supported by the Office of Naval Research under Grant N00014-96-1-0502. This paper was recommended by Associate Editor S.-F. Chang.

The authors are with the Department of Electrical and Computer Engineering, Beckman Institute, University of Illinois, Urbana, IL 61801 USA (e-mail: dugad@vision.ai.uiuc.edu; ahuja@vision.ai.uiuc.edu).

Publisher Item Identifier S 1051-8215(01)03014-2.

1051–8215/01$10.00 © 2001 IEEE

[4], [5]. Let T denote the 8×8 DCT operator matrix. Since $TT^t = T^tT = I$, we have

$$\mathrm{DCT}(c) = TcT^t = \sum_{i=1}^{4} \mathrm{DCT}(h_i)\,\mathrm{DCT}(c_i)\,\mathrm{DCT}(g_i). \quad (3)$$

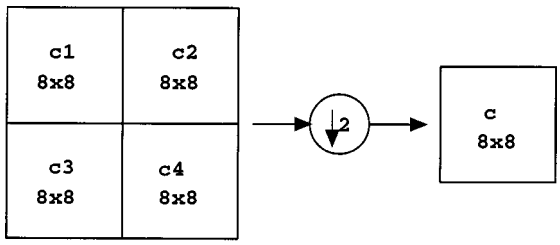
$\mathrm{DCT}(h_i)$ and $\mathrm{DCT}(g_i)$ for $i = 1$ to 4 can be precomputed. Hence, given $\mathrm{DCT}(c_i)$ for $i = 1$ to 4, $\mathrm{DCT}(c)$ can be computed as matrix multiplication. However, even though the filter matrices h_i and g_i are sparse, $\mathrm{DCT}(h_i)$ and $\mathrm{DCT}(g_i)$ are not sparse at all. Hence, the matrix multiplication in (3) can have complexity comparable (the Appendix gives a more quantitative reasoning) to downsampling in spatial domain (by first taking inverse DCT, filtering, and taking DCT) using fast algorithms for computation of the DCT [6].

It can be shown that even if a fast algorithm (tailored for 8-point DCT) is used for forward and inverse DCT computation, the computational complexity of the straightforward approach of computing the inverse DCT, then scaling in the spatial domain using (1) and (2), and then taking the forward DCT would be 3.44 multiplications and 9.81 additions per pixel (of the original image). Specifically, note that this procedure does not exploit the spatial redundancy present in the image (the same number of computations will be required, even if the image is white noise).

The spatial redundancy can be exploited by working in the DCT domain where the block-DCT of the image would be sparse. This also avoids the expensive operations of forward and inverse DCT. Using (3), grouping the terms properly, and further assuming that $\mathrm{DCT}(c_i)$ is 75% sparse (i.e., the high-frequency coefficients are all zero), it is shown in [2] that the computational complexity is reduced to 4 multiplications and 4.75 additions per pixel (of original image). We see that, for the special case of 8-point block-DCT, the tradeoff between working in the spatial domain and using the method of [2] depends critically on how expensive a multiplication is compared to an addition. Note that though the spatial filter matrices h_i are sparse, $\mathrm{DCT}(h_i)$ is not sparse at all, and this plays a significant role in increasing the computations. In this paper, we shall introduce a scheme in which $\mathrm{DCT}(h_i)$ are sparse which yields dramatic improvements in computation. This is achieved without compromising the quality of the filter h_i. In fact we shall see that our scheme is subjectively and PSNR wise better compared to the usual (bilinear interpolation) scheme described above in (1)–(2).

In [1], a fast algorithm has been developed for the computation of (3) based on factorization of the DCT matrix corresponding to the Winograd algorithm. However, their algorithm is mainly aimed at the downsampling matrix in (2) (bilinear interpolation) and it has been stated in the paper that it is not guaranteed that for every reasonable anti-aliasing filter, the method would have smaller complexity than spatial domain methods.

A signal can be synthesized by linearly combining the DCT basis vectors using the DCT coefficients of the signal. Hence, one way to obtain a downsampled version of the signal would be to use downsampled versions of the basis vectors in the synthesis process. One way to obtain a downsampled version of the basis vectors is to resample the original continuous-time basis functions more coarsely. An anti-aliasing filter can be implemented simultaneously by zeroing out the high-frequency coefficients during the synthesis process. Such an idea was utilized in [7] to implement anti-aliasing filtering, downsampling, and taking a forward transform in one step. The same idea is utilized in [8] to obtain computational savings when the downsampled signal is desired in the same format (e.g., 8-point DCT coefficients) as the original signal. However, in their case, the high-frequency coefficients are not zeroed out. The coefficients in the final transformation are approximated with inverse powers of two for further computational efficiency.



(a)

$$\mathbf{u}_{4\times 8} = \begin{bmatrix} .5 & .5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .5 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .5 & .5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .5 & .5 \end{bmatrix}$$

(b)

Fig. 1. (a) Downsampling four consecutive blocks to get a single block. (b) Definition of $\mathbf{u}_{4\times 8}$ used in (2).

Suppose that the original DCT basis vectors consists of eight samples. When downsampling by a factor of two, one would resample the original continuous-time basis functions at four uniformly spaced points. This results in basis vectors of a 4-point DCT transform. When the high-frequency coefficients are zeroed out during downsampling, we see that the above described method is same as taking the 4-point inverse DCT of the four low-frequency coefficients in a 8-point DCT of the original 8-point signal. This will result in four samples, which would be a downsampled version of the original eight samples. Such a scheme was described in [9], though the interpretation in terms of coarsely sampling the continuous-time basis functions was not provided there.

The scheme presented in this paper relies on the point of view presented in [9] (which also implies that we shall be zeroing out the high-frequency DCT coefficients). However, our goal is to obtain the downsampled signal in the same format as the original signal. We shall *prove* the sparseness properties of the matrices involved by appealing to the orthogonality and symmetry properties of the DCT matrices involved. This means that the sparseness properties will

hold true for other transformations as well if they satisfy similar orthogonality and symmetry properties. Moreover, our approach also allows us to introduce a corresponding upsampling scheme that preserves all the low-frequency components of the original signal.

We see that most previous approaches view the downsampling operation as low-pass filtering followed by downsampling, and implement this in the DCT domain. The low-pass filter is typically chosen independent of the DCT transform. In our approach, we shall show that it is possible to design a low-pass filter so that DCT of the filter matrix is sparse rather than the filter matrix itself. This results in very high computational savings. We make use of the fact that low-pass filtering and downsampling can be combined and performed directly in the DCT domain [9]. Specifically, we make use of the fact that taking a 4×4 inverse DCT of the 4×4 low-pass coefficients of $\mathrm{DCT}(c)$ (which is 8×8) directly gives a low-passed and half-decimated version of c. Hence, in our method, the downsampled image retains *all* the low-frequency components of the original image.

The previous methods mentioned above do not discuss any upsampling scheme corresponding to the downsampling scheme. Upsampling is important for applications involving multiscale image representations, e.g., spatially scalable encoding of video where a prediction or estimation of a high-resolution frame is obtained by upsampling its low resolution version (which is transmitted as the base layer). We shall present an upsampling scheme which also works in the compressed domain. When the upsampling scheme is used together with our downsampling scheme, all the low-pass coefficients of the original image are preserved. This implies significant savings in the bits allocated for transmitting the enhancement layer, which carries the difference between the original and its prediction (i.e., the result of upsampling the downsampled image); only the high-frequency coefficients need to be transmitted in the enhancement layer. However, the upsampling scheme can also be used on its own without regard to the downsampling scheme presented here. Our experiments show that the upsampling scheme produces visually and PSNR-wise better images (compared to bilinear interpolation), even if the downsampled images are created using bilinear interpolation. This feature is important in applications where the user has no control over the downsampling scheme, or if one is interested in upsampling the original image itself.

Section II presents the downsampling scheme for 1-D signals and then extends it to 2-D signals. It also shows how the scheme is equivalent to designing a downsampling filter whose DCT is sparse rather than the filter itself being sparse. Section III presents details of the upsampling scheme, and we shall see how the downsampling and upsampling schemes combined together preserve all the low-frequency components of the original image. Section IV gives an analysis of aliasing effects for our scheme. Section V presents results showing the subjective and PSNR improvement of our scheme over the bilinear scheme. Section VI presents our conclusions. Finally, the Appendix gives detailed derivations of the exact computations required by our downsampling and upsampling schemes.

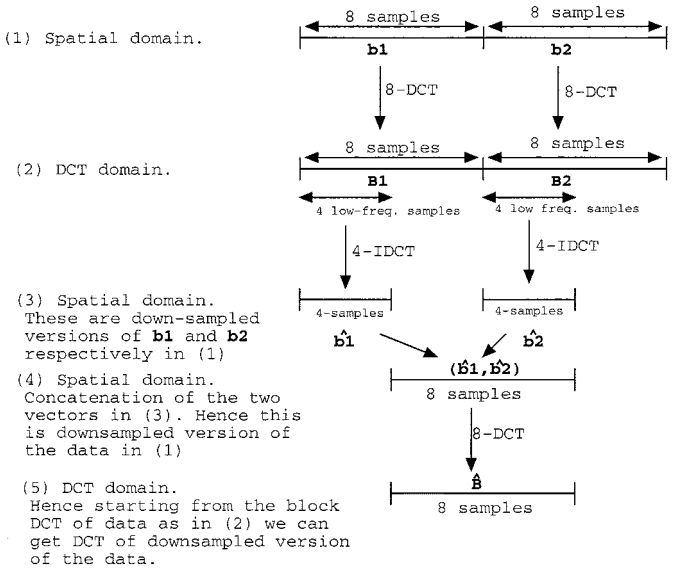


Fig. 2. Schematic of our approach for downsampling in the DCT domain. Starting with the block DCT [in (2)] of data [in (1)], we get the DCT of downsampled version of data in (5). The crucial point is the 4-point IDCT in going from (2) to (3) and the 8-point DCT in going from (4) to (5) can be combined together to gain significant reduction in computations. Also note that the 8-samples in (5) contain all the low-frequency coefficients of the original data in (1).

II. Downsampling in the DCT Domain

The outline of our scheme is shown in Fig. 2 for 1-D signals. Let $\mathbf{B}_1$ and $\mathbf{B}_2$ denote the 8-point DCT of two consecutive 8-sample blocks $\mathbf{b}_1$ and $\mathbf{b}_2$. We wish to generate the 8-point DCT $\hat{\mathbf{B}}$ of the 8-point block got by downsampling $(\mathbf{b}_1, \mathbf{b}_2)$ with an appropriate filter. The downsampling has to be carried out as far as possible in the compressed domain. In principle, the proposed scheme can be viewed as follows. Take 4-point inverse DCT of the four low-pass coefficients in $\mathbf{B}_1$, and similarly for $\mathbf{B}_2$. Concatenate these two 4-point blocks and then take its 8-point DCT. The resulting block is the desired block $\hat{\mathbf{B}}$. It turns out that this series of operations (starting from $\mathbf{B}_1$ and $\mathbf{B}_2$ in DCT-domain to $\hat{\mathbf{B}}$ also in DCT domain) can be implemented in a computationally much more efficient manner compared to matrix multiplication by the DCT of the downsampling filter [cf., (3)]. The scheme works because taking 4-point inverse DCT of the four low-pass coefficients of an 8-point DCT of a block gives a low-passed and downsampled version of the original 8-point block [9].

We shall now elaborate on the downsampling scheme and give a computationally efficient algorithm for it. We shall derive the relevant equations in 1-D and then extend them to 2-D. Let $\mathbf{b}_1$ and $\mathbf{b}_2$ denote two consecutive 8-pixel blocks in the spatial domain. Let $\mathbf{B}_1$ and $\mathbf{B}_2$ be their 8-point DCTs, respectively. Let $\mathbf{b} = \left[\begin{smallmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{smallmatrix}\right]$. Let $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$ denote the first four (low-pass) components of $\mathbf{B}_1$ and $\mathbf{B}_2$. Let $\hat{\mathbf{b}}_1$ and $\hat{\mathbf{b}}_2$ denote the 4-point inverse DCT of $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$. Hence, $\hat{\mathbf{b}}_1$ and $\hat{\mathbf{b}}_2$ are low-passed and downsampled versions of $\mathbf{b}_1$ and $\mathbf{b}_2$, respectively. Let $\hat{\mathbf{b}} = \left[\begin{smallmatrix}\hat{\mathbf{b}}_1\\ \hat{\mathbf{b}}_2\end{smallmatrix}\right]$ and let $\hat{\mathbf{B}}$ be the 8-point DCT of $\hat{\mathbf{b}}$. Hence $\hat{\mathbf{b}}$ is a low-pass filtered downsampled version of $\mathbf{b}$. We need to compute $\hat{\mathbf{B}}$ directly from $\mathbf{B}_1$ and $\mathbf{B}_2$ (i.e., from $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$). Let $T\ (8 \times 8)$ denote the 8-point DCT operator matrix and let $T_4\ (4 \times 4)$ denote the

4-point DCT operator matrix. Then we have the following equation:

$$\hat{\mathbf{B}} = T\hat{\mathbf{b}} = T\begin{bmatrix}\hat{\mathbf{b}}_1\\ \hat{\mathbf{b}}_2\end{bmatrix} = [T_L \quad T_R]\begin{bmatrix}T_4^t\hat{\mathbf{B}}_1\\ T_4^t\hat{\mathbf{B}}_2\end{bmatrix} = T_L T_4^t\hat{\mathbf{B}}_1 + T_R T_4^t\hat{\mathbf{B}}_2 \tag{4}$$

where T_L and T_R are 8×4 matrices denoting the first and last four columns, respectively, of the 8-point DCT operator T. Let us have a look at $T_L T_4^t$ and $T_R T_4^t$ and see how about 50% of the terms in the product turn out to be zeros. The following points can be noted immediately from the expressions[1] for T and T_4 as given in (5), shown at the bottom of the page, and in (6), as follows:

$$T_4 = \begin{pmatrix} 1 & 1 & 1 & 1\\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8}\\ \cos\frac{2\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{2\pi}{8}\\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} & -\cos\frac{9\pi}{8} & -\cos\frac{3\pi}{8}\end{pmatrix}. \tag{6}$$

1) For $k = 0, 1, 2, 3$, the $(2k)$th row of T_L is same as the kth row of T_4 and the the $(2k)$th row of T_R is negative of the kth row of T_4.
2) Since the rows of T_4 are orthogonal, point 1 above implies that *every* $2k$th row of of T_L (and also of T_R) is orthogonal to *every* row of T_4 except the kth row for $k = 0, 1, 2, 3$. Hence, in the 8×4 matrices $T_L T_4^t$ and $T_R T_4^t$, every $(2k)$th row has all its entries as zeros except the kth entry. Hence, we see that about 50% of the entries of these matrices are zero.
3) Odd rows of T are anti-symmetric and even rows are symmetric. Also, odd rows of T_4 are anti-symmetric and even rows are symmetric. These two facts together imply that all the corresponding entries of $T_L T_4^t$ and $T_R T_4^t$ are identical excepPt possibly for a change of sign. Specifically, $T_L T_4^t(i,j) = (-1)^{i+j} T_R T_4^t(i,j)$ for $i = 0, 1, \ldots, 7$ and $j = 0, 1, 2, 3$. This fact can be exploited to further reduce the computations in (4). Grouping the terms for which $i + j$ is even into one matrix C and the remaining terms into another matrix D, we have $T_L T_4^t = C + D$ and $T_R T_4^t = C - D$. Hence, from (4), we have

$$\hat{\mathbf{B}} = (C+D)\hat{\mathbf{B}}_1 + (C-D)\hat{\mathbf{B}}_2 \tag{7}$$

$$= C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_2) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_2). \tag{8}$$

The expression in (8) is much faster to compute compared to that in (7) because each of the matrices C and D have only half the number of nonzero entries compared to $C+D$ or $C-D$. Hence, the number of multiplications involved is reduced drastically. Another way to look at this is that instead of computing expressions like $\alpha\beta + \alpha\gamma$ in (7), we are computing $\alpha(\beta + \gamma)$ in (8).

4) Since T and T_4 are DCT matrices, it is our belief that much faster procedures (based on fast DCT algorithms [10]) could be designed for the scheme [cf., (4)] presented here. But we shall not pursue this in the current paper.

Derivation of the exact number of multiplications and additions required by our method is given in the Appendix. It is shown that the amount of computation for the 2-D case is 1.25 multiplications and 1.25 additions per pixel of the original image.

A. The Downsampling Filter

In this section, we shall derive the downsampling filter, which corresponds to the downsampling operation mentioned above. We show that in our approach, the downsampling filter is such that the DCT of the filter matrix is sparse rather than the filter itself being sparse. This is more desirable from the point of view of reducing the computations in the compressed domain. We have the following equation:

$$\hat{\mathbf{b}}_1 = T_4^t\hat{\mathbf{B}}_1 = T_4^t[I \quad O]\mathbf{B}_1 = T_4^t[I \quad O]T\mathbf{b}_1 = T_4^t M^t T\mathbf{b}_1$$
$$M \stackrel{\text{def}}{=} \begin{bmatrix}I\\ O\end{bmatrix} \tag{9}$$

where I and O denote 4×4 identity and zero matrices, respectively. Hence, we have

$$\begin{bmatrix}\hat{\mathbf{b}}_1\\ \mathbf{0}\end{bmatrix} = MT_4^t M^t T\mathbf{b}_1 = h\mathbf{b}_1; \quad h \stackrel{\text{def}}{=} MT_4^t M^t T \tag{10}$$

where $\mathbf{0}$ is a 4×1 zero vector. Hence, $h(8\times 8)$ gives the downsampling filter matrix corresponding to our scheme. Compare this with the filter matrix h_1 in (2) for the bilinear scheme. h_1 is chosen without considering the fact that finally, in (3), it is the DCT of h_1 that is used. It is more desirable to have $\mathrm{DCT}(h_1)$ to be sparse than to have h_1 as sparse. In our approach, the downsampling filter h in (10) is derived from the DCT basis functions so that $\mathrm{DCT}(h)$ is guaranteed to be very sparse[2]

$$\mathrm{DCT}(h) = ThT^t = [T_L \quad T_R]MT_4^t M^t TT^t = T_L T_4^t M^t. \tag{11}$$

We have already seen above that $T_L T_4^t$ is very sparse. The M^t at the end makes sure that only the first four low-pass components of $\mathrm{DCT}(\mathbf{b}_1)$ are used in the computation of $\hat{\mathbf{b}}_1$. The product

[1]The normalizing constants in these expressions have been omitted since they do not affect our conclusions.

[2]Note that $\mathrm{DCT}(h\mathbf{b}_1) = Th\mathbf{b}_1 = ThT^t T\mathbf{b}_1 = \mathrm{DCT}(h)\,\mathrm{DCT}(\mathbf{b}_1)$.

$$T = [T_L \,|\, T_R] = \left(\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & -\cos\frac{7\pi}{16} & -\cos\frac{5\pi}{16} & -\cos\frac{3\pi}{16} & -\cos\frac{\pi}{16}\\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{3\pi}{8} & \cos\frac{3\pi}{8} & \cos\frac{\pi}{8}\\ \vdots & & & & & & & \vdots \end{array}\right) \tag{5}$$

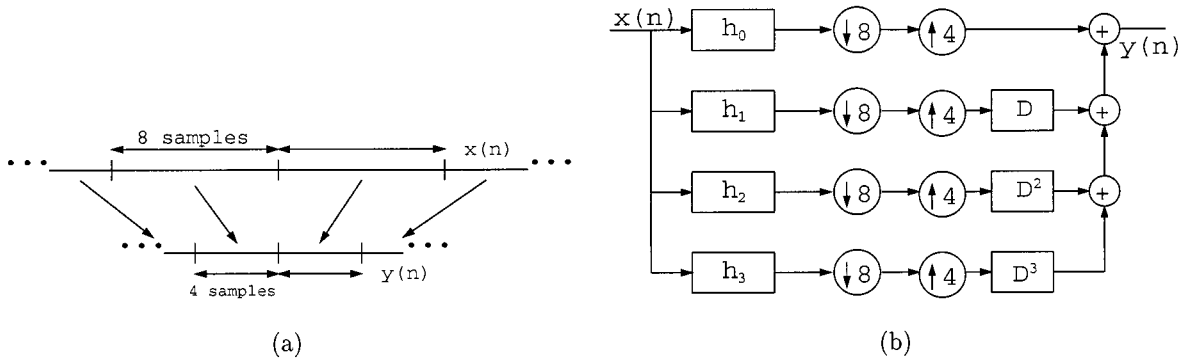


Fig. 3. Downsampling. (a) Here each 4-sample block of the output $y(n)$ is derived from the corresponding 8-sample block of the input $x(n)$ through a linear transformation given by a matrix A. (b) Downsampling operation in (a) is represented here as a filter bank structure. The right side of the structure consisting of the upsamplers followed by delays and adders corresponds to sampling the outputs of the downsamplers in succession. The filters $\{h_i\}_{i=0}^{i=3}$ correspond to the rows of the linear transformation matrix A in (a) written in reverse order.

$T_R T_4^t$ appears in the computation of $\left[\begin{smallmatrix}\mathbf{0}\\ \hat{\mathbf{b}}_2\end{smallmatrix}\right]$, which when added to $\left[\begin{smallmatrix}\hat{\mathbf{b}}_1\\ \mathbf{0}\end{smallmatrix}\right]$ gives the desired low-pass downsampled vector $\hat{\mathbf{b}}$.

B. Extension to 2-D

Let $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ and $\mathbf{b}_4$ denote four consecutive 8×8 blocks numbered in the same way as c_1, c_2, c_3, and c_4 in Fig. 1. Following the same notation as in the 1-D case, let $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ and $\mathbf{B}_4$ denote the DCTs of these blocks, respectively. Let $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2, \hat{\mathbf{B}}_3$, and $\hat{\mathbf{B}}_4$ denote the 4×4 matrices containing the low-pass coefficients of $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ and $\mathbf{B}_4$, respectively. Let $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3$ and $\hat{\mathbf{b}}_4$ denote the 4×4 inverse DCT of $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2, \hat{\mathbf{B}}_3$ and $\hat{\mathbf{B}}_4$, respectively. Then $\hat{\mathbf{b}} \stackrel{\text{def}}{=} \left[\begin{smallmatrix}\hat{\mathbf{b}}_1 & \hat{\mathbf{b}}_2\\ \hat{\mathbf{b}}_3 & \hat{\mathbf{b}}_4\end{smallmatrix}\right]$ denotes the low-pass and downsampled version of $\mathbf{b} \stackrel{\text{def}}{=} \left[\begin{smallmatrix}\mathbf{b}_1 & \mathbf{b}_2\\ \mathbf{b}_3 & \mathbf{b}_4\end{smallmatrix}\right]$. Let $\hat{\mathbf{B}} \stackrel{\text{def}}{=} \text{DCT}(\hat{\mathbf{b}})$. We need to compute $\hat{\mathbf{B}}$ directly from $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$, and $\mathbf{B}_4$ (i.e., from $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2, \hat{\mathbf{B}}_3$, and $\hat{\mathbf{B}}_4$). We have

$$\hat{\mathbf{B}} = T\hat{\mathbf{b}}T^t \tag{12}$$

$$= [T_L \quad T_R]\begin{bmatrix}\hat{\mathbf{b}}_1 & \hat{\mathbf{b}}_2\\ \hat{\mathbf{b}}_3 & \hat{\mathbf{b}}_4\end{bmatrix}\begin{bmatrix}T_L^t\\ T_R^t\end{bmatrix} \tag{13}$$

$$= [T_L \quad T_R]\begin{bmatrix}T_4^t\hat{\mathbf{B}}_1 T_4 & T_4^t\hat{\mathbf{B}}_2 T_4\\ T_4^t\hat{\mathbf{B}}_3 T_4 & T_4^t\hat{\mathbf{B}}_4 T_4\end{bmatrix}\begin{bmatrix}T_L^t\\ T_R^t\end{bmatrix} \tag{14}$$

$$= \left(T_L T_4^t\right)\hat{\mathbf{B}}_1\left(T_L T_4^t\right)^t + \left(T_L T_4^t\right)\hat{\mathbf{B}}_2\left(T_R T_4^t\right)^t + \left(T_R T_4^t\right)\hat{\mathbf{B}}_3\left(T_L T_4^t\right)^t + \left(T_R T_4^t\right)\hat{\mathbf{B}}_4\left(T_R T_4^t\right)^t. \tag{15}$$

We have already seen that $(T_L T_4^t)$ and $(T_R T_4^t)$ are very sparse. We know from 1-D case that $(T_L T_4^t) = C + D$ and $(T_R T_4^t) = C - D$, where each of C and D contain only half as many nonzero entries as $(T_L T_4^t)$ or $(T_R T_4^t)$. Let us see how this can be used to reduce the computations further, as was done in the 1-D case. We have

$$\hat{\mathbf{B}} = (C+D)\hat{\mathbf{B}}_1(C+D)^t + (C+D)\hat{\mathbf{B}}_2(C-D)^t + (C-D)\hat{\mathbf{B}}_3(C+D)^t + (C-D)\hat{\mathbf{B}}_4(C-D)^t \tag{16}$$

$$= [(C+D)\hat{\mathbf{B}}_1 + (C-D)\hat{\mathbf{B}}_3](C+D)^t + [(C+D)\hat{\mathbf{B}}_2 + (C-D)\hat{\mathbf{B}}_4](C-D)^t \tag{17}$$

$$= [C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_3) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_3)](C+D)^t + [C(\hat{\mathbf{B}}_2 + \hat{\mathbf{B}}_4) + D(\hat{\mathbf{B}}_2 - \hat{\mathbf{B}}_4)](C-D)^t \tag{18}$$

$$= X(C+D)^t + Y(C-D)^t \tag{19}$$

$$= (X+Y)C^t + (X-Y)D^t \tag{20}$$

where

$$X = C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_3) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_3) \tag{21}$$

$$Y = C(\hat{\mathbf{B}}_2 + \hat{\mathbf{B}}_4) + D(\hat{\mathbf{B}}_2 - \hat{\mathbf{B}}_4). \tag{22}$$

We know from the 1-D case that it is faster to compute the expression in (17) as shown in (18). Again, from the 1-D case, we know that is faster to compute the expressions in (19) as shown

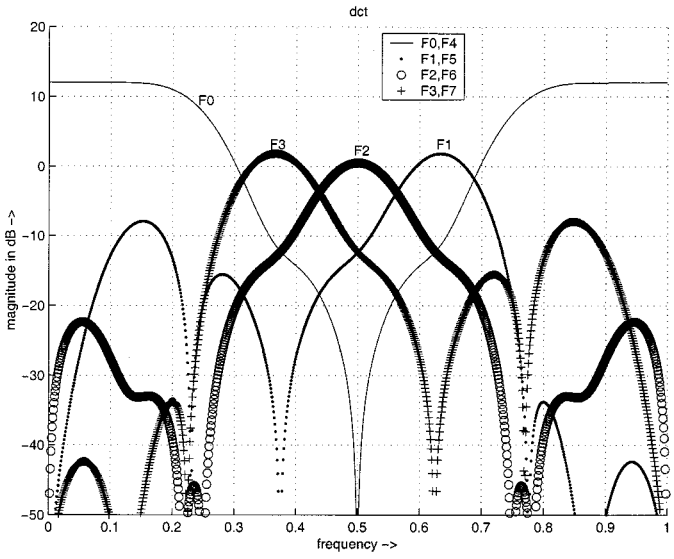


Fig. 4. Magnitude response of F_k's in (32) with $z = \exp(j2\pi f)$, where f denotes frequency in Hz.

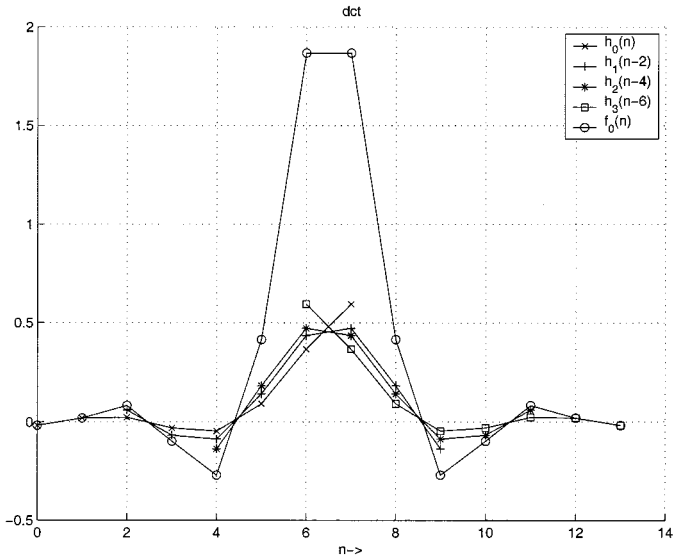


Fig. 5. Time-domain samples of the filter F_0 in (32) along with the filters H_0 through H_3. Note how the significant taps of H_0 through H_3 coincide in location after the appropriate shifts. Adding these filters after the shifts gives F_0. Note that F_0 is even length and symmetric and hence has (generalized) linear phase.

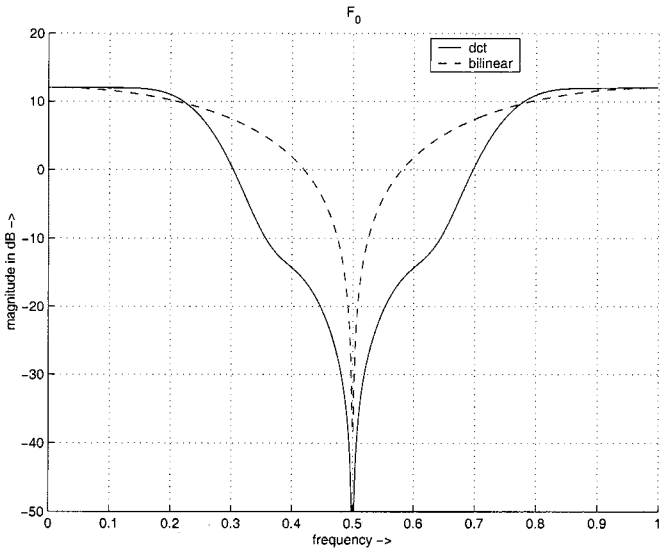


Fig. 6. Comparing the F_0 [see (32)] for our (dct) case with the bilinear case. For the bilinear case F_0 consists of just two consecutive taps, each of magnitude 0.5, and hence the magnitude decays down sinusoidally.

in (20). Hence, for the 2-D case, the procedure is [3] compute X and Y as in (21) and (22), and then compute $\hat{\mathbf{B}}$ as in (20). As mentioned before, it is shown in the Appendix that the downsampling scheme takes 1.25 multiplications and 1.25 additions per pixel of the original image.

Comparing (20)–(22) to (8), it can be seen that the 2-D scheme is separable. Equations (21)–(22) imply that X and Y can be computed by applying the 1-D scheme along each of the columns of $\begin{bmatrix}\hat{\mathbf{B}}_1\\ \hat{\mathbf{B}}_3\end{bmatrix}$ and $\begin{bmatrix}\hat{\mathbf{B}}_2\\ \hat{\mathbf{B}}_4\end{bmatrix}$, respectively. Then $\hat{\mathbf{B}}$ can be computed by applying the 1-D scheme along the rows of $[X\ \ Y]$. (The same computation can also be performed by applying the 1-D scheme along the rows of $[\hat{\mathbf{B}}_1\ \ \hat{\mathbf{B}}_2]$ and $[\hat{\mathbf{B}}_3\ \ \hat{\mathbf{B}}_4]$, respectively, and then applying the 1-D scheme along the columns of the concatenation of the row outputs.) This is another way of interpreting (20)–(22) and does not improve the computational complexity over that mentioned above.

III. Upsampling

In many applications, it is required to obtain an upsampled version of an image from its downsampled version. For example, for spatially scalable encoding of video [11], [12], a prediction of the original frame is obtained by upsampling the lower resolution frame and the difference is encoded in the enhancement or low-priority layer. We have seen that the downsampling scheme preserves all the low-frequency information of the original image. Hence, a natural question is whether we can design an upsampling scheme that works in the compressed domain and can keep all the low-frequency components of the original image in the upsampled image. That way, the residual error

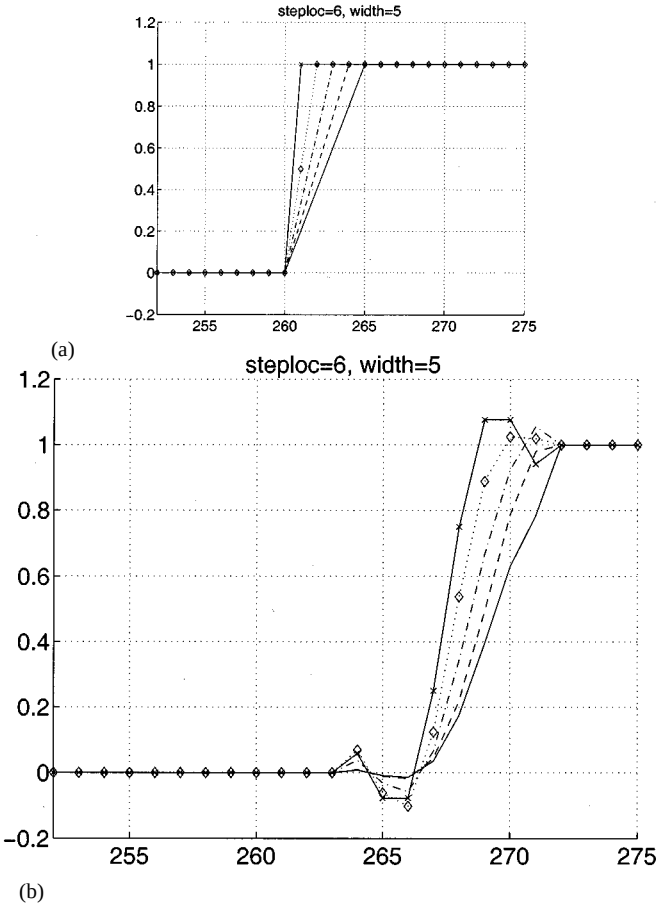


Fig. 7. (b) Outputs (after downsampling and upsampling using our scheme) corresponding to each of the input edges shown in (a), respectively. We see that the ripples in the output die down with decrease in the slope of the input edge and almost no ripples are seen for the four and five pixel wide edges. Similar, but not identical, responses are observed as the starting location of the input edges is changed. This implies that Gibbs phenomenon will be observed only near very sharp edges. The output is delayed with respect to the input due to our filter-bank implementation.

[3]Actually due to the different sizes of the DCTs there would be a factor of half that we have not accounted for so far. Hence, in practice, we have $X = (1/2)[C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_3) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_3)]; Y = (1/2)[C(\hat{\mathbf{B}}_2 + \hat{\mathbf{B}}_4) + D(\hat{\mathbf{B}}_2 - \hat{\mathbf{B}}_4)]$. Note that the factor of $1/2$ can be absorbed in C and D, which would be pre-computed.

Image	Lena	Watch	Cap	F-16
bilinear	30.04	25.15	32.08	28.11
our	34.69	29.09	34.22	32.28

Fig. 8. PSNR values after downsampling and upsampling using bilinear interpolation and our scheme.

Image	Lena	Watch	Cap	F-16
(a)bilinear	30.04	25.15	32.08	28.11
(b)our	34.06	28.63	33.73	31.53

Fig. 9. PSNR values after downsampling using bilinear interpolation and upsampling using (a) bilinear interpolation and (b) our scheme. This table differs from the one in Fig. 8 in that the "our" scheme in Fig. 8 uses our scheme for downsampling whereas in this table the downsampling scheme is bilinear interpolation (averaging). Comparing the two tables makes it evident that the upsampling scheme presented here can be used to gain significant PSNR improvements even if one has no control or knowledge of how the downsampled image was created. Figs. 10–13 make a visual comparison.

(difference between original and upsampled images) would be same as the energy in the high-frequency components, which is typically very low. Such an upsampling scheme can indeed be designed as we shall soon see. For creating the downsampled image, we took 4×4 inverse DCT of low-frequency components of four consecutive 8×8 blocks and then took the forward DCT of the concatenation of these 4×4 blocks. This gives us the downsampled image in the DCT domain, and we saw how this could be implemented very efficiently. Hence, in principle, we can get an upsampled image from this downsampled image as follows: For each 8×8 $\hat{\mathbf{B}}$ block in the downsampled image, compute $\hat{\mathbf{b}} = \mathrm{IDCT}(\hat{\mathbf{B}})$. Split $\hat{\mathbf{b}}$ into 4×4 sub-blocks so that $\hat{\mathbf{b}} = \left[\begin{smallmatrix} \hat{\mathbf{b}}_1 & \hat{\mathbf{b}}_2 \\ \hat{\mathbf{b}}_3 & \hat{\mathbf{b}}_4 \end{smallmatrix}\right]$. Now compute $\hat{\mathbf{B}}_1 = \mathrm{DCT}(\hat{\mathbf{b}}_1)$ (4×4 DCT) etc. $\hat{\mathbf{B}}_1$ contains all the low-frequency DCT coefficients of the corresponding 8×8 block in the original image and these are the only coefficients that we can recover from the downsampled image. Hence, let $\tilde{\mathbf{B}}_1 = \left[\begin{smallmatrix} \hat{\mathbf{B}}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{smallmatrix}\right]$, where $\mathbf{O}$ denotes a 4×4 zero matrix. Then $\tilde{\mathbf{B}}_1$ is the block (in DCT domain) corresponding to the block $\mathbf{B}_1$ in the original image. Similarly, one can get $\tilde{\mathbf{B}}_2, \tilde{\mathbf{B}}_3$ and $\tilde{\mathbf{B}}_4$. This way, we have obtained an upsampled image from the downsampled image. Note that the process of downsampling and then upsampling has the effect of truncating to zero all the high-frequency components in each 8×8 DCT block in the original image while preserving all the low-frequency components of such blocks. One would expect such truncation to give rise to ringing artifacts due to the Gibbs phenomenon. It is shown in Section IV that ringing is negligible except around very steep edges. Since most natural images do not exhibit sharp discontinuities, there are no visually noticeable ringing artifacts. This is verified subjectively in Section V. As in the case of downsampling, it turns out that the upsampling procedure can also be implemented very efficiently. In terms of our notation, in the previous section the problem is whether we can get back $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2, \hat{\mathbf{B}}_3$ and $\hat{\mathbf{B}}_4$ from $\hat{\mathbf{B}}$. $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}_1$, etc., are related as in (15). To get the inverse relationships, we need to derive some orthogonality relationships between the matrices involved in that equation. We have

$$I_{8\times 8} = T^t T = \begin{bmatrix} T_L^t \\ T_R^t \end{bmatrix} [T_L \quad T_R] = \begin{bmatrix} T_L^t T_L & T_L^t T_R \\ T_R^t T_L & T_R^t T_R \end{bmatrix}. \quad (23)$$

Hence, we have

$$T_L^t T_L = I_{4\times 4} = T_R^t T_R; \quad T_L^t T_R = O_{4\times 4} = T_R^t T_L. \quad (24)$$

From these equations, we get

$$I_{4\times 4} = T_4 \left(T_L^t T_L\right) T_4^t = \left(T_L T_4^t\right)^t \left(T_L T_4^t\right). \quad (25)$$

Similarly, the following relations can be proven:

$$I_{4\times 4} = \left(T_R T_4^t\right)^t \left(T_R T_4^t\right); \quad O_{4\times 4} = \left(T_L T_4^t\right)^t \left(T_R T_4^t\right)$$
$$O_{4\times 4} = \left(T_R T_4^t\right)^t \left(T_L T_4^t\right) \quad (26)$$

Using (25)–(26) in (15), we get[4]

$$\hat{\mathbf{B}}_1 = \left(T_L T_4^t\right)^t \hat{\mathbf{B}} \left(T_L T_4^t\right); \quad \hat{\mathbf{B}}_2 = \left(T_L T_4^t\right)^t \hat{\mathbf{B}} \left(T_R T_4^t\right) \quad (27)$$
$$\hat{\mathbf{B}}_3 = \left(T_R T_4^t\right)^t \hat{\mathbf{B}} \left(T_L T_4^t\right); \quad \hat{\mathbf{B}}_4 = \left(T_R T_4^t\right)^t \hat{\mathbf{B}} \left(T_R T_4^t\right) \quad (28)$$

Then $\tilde{\mathbf{B}}_1 = \left[\begin{smallmatrix} \hat{\mathbf{B}}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{smallmatrix}\right]$ etc. give us the four blocks in the upsampled image corresponding to the block $\hat{\mathbf{B}}$ in the downsampled image. Note that the upsampling scheme can be applied to any given image, irrespective of whether or not the given image is obtained by the downsampling scheme outlined in Section II. Again, using $T_L T_4^t = C + D$ and $T_R T_4^t = C - D$, (27)–(28) can be implemented efficiently by first computing $C^t \hat{\mathbf{B}} C, C^t \hat{\mathbf{B}} D, D^t \hat{\mathbf{B}} C$ and $D^t \hat{\mathbf{B}} D$. Then, obtaining $\hat{\mathbf{B}}_1$ etc. is only a matter of adding these four matrices with different signs. It is shown in the Appendix that the computational complexity of the upsampling scheme is 1.25 multiplications and 1.25 additions per pixel of the upsampled image.

IV. Analysis of Aliasing

In this section, we shall analyze the anti-aliasing properties of our downsampling scheme for the 1-D case. The same analysis applies to the 2-D case since the 2-D scheme is separable (see Section II-B). The scheme can be represented as shown in Fig. 3(a). Thus, each block of eight samples of the input $x(n)$ is linearly transformed with matrix A to get a corresponding block of four samples of the output $y(n)$. For example, for the bilinear case the matrix A is given by the first four rows of the h_1 matrix in (2). For our scheme A is given by the first four rows of the h matrix in (10). Evaluating the h matrix as given in (10) shows that, unlike the rows of h_1 in (2), the rows of the h matrix are not shifted versions of each other. This implies that unlike the bilinear scheme our scheme can not be represented as a filter followed by downsampling by two. Hence, it is not immediately clear as to what "filter" serves as an equivalent pre-filter

[4]As mentioned in footnote 3, we have to account for a factor of half due to different sizes of the DCTs involved, and hence, in practice, the equations used for upsampling would be $\hat{\mathbf{B}}_1 = 2(T_L T_4^t)^t \hat{\mathbf{B}} (T_L T_4^t)$, etc.

(a) (b) (c) (d)

Fig. 10. Lena downsampled using (a) our method and (b) bilinear interpolation (i.e., replacing every 2×2 block of pixels by its average). (c) The scheme in [8] using exact computation for the S and T matrices in their scheme. (d) The scheme in [8] where the entries of S and T matrices are approximated by inverse powers of two. Artifacts can easily be seen in (d) (e.g., look at the shoulder region and the black strip in the background).

for our downsampling scheme. We shall see that the rows of the h matrix (they are plotted in Fig. 5) are approximately shifted versions of each other and an average of these rows with the corresponding shifts serves as an equivalent pre-filter.

The scheme in Fig. 3(a) can be represented as a filter bank structure as shown in Fig. 3(b). Here, the filter h_i[5] is given by the ith row of the matrix A written in reverse order. Hence, using standard results from multirate signal processing [13], the z-transform of the output can be written as (capital letters are used to denote the z-transforms of the corresponding small-letter time-domain signals)

$$Y(z) = \sum_{i=0}^{M-1} z^{-i} \frac{1}{N} \sum_{k=0}^{N-1} X(W^{-k}\sqrt{z}) H_i(W^{-k}\sqrt{z}) \quad (29)$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} X(W^{-k}\sqrt{z}) \sum_{i=0}^{M-1} z^{-i} H_i(W^{-k}\sqrt{z}) \quad (30)$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} X(W^{-k}\sqrt{z}) F_k(W^{-k}\sqrt{z}) \quad (31)$$

where $N = 8, M = N/2, j = \sqrt{-1}, W = \exp(j2\pi/N)$ and

$$F_k(z) = \sum_{i=0}^{M-1} z^{-2i} H_i(z) W^{-2ki}. \quad (32)$$

From this expression and the definition of W, it is clear that

$$F_{k+M} \equiv F_k \quad \text{for } k = 0 \text{ through } M-1. \quad (33)$$

Equation (31) can be written as

$$Y(z) = U(\sqrt{z}) \quad (34)$$

[5]This section uses its own notation and should not be confused with the notation used in Sections I and II and referred in the previous paragraph.

Fig. 11. Lena: (a) Original; (b) downsampled and upsampled by our method; (c) downsampled and upsampled using bilinear interpolation (as in [4], [2]); and (d) downsampled by bilinear interpolation and upsampled by our method. We see that the image in (b) is sharper compared to that in (c); this can be particularly noticed in the feather and eye regions. The image in (d) is also sharper compared to (c) and has almost the same visual features as that in (b), even though we use the bilinear scheme for downsampling. PSNR values with respect to the original are: (b) 34.69 dB; (c) 30.04 dB; and (d) 34.06 dB.

where

$$U(z) = \frac{1}{N} \sum_{k=0}^{N-1} X(W^{-k}z) F_k(W^{-k}z). \tag{35}$$

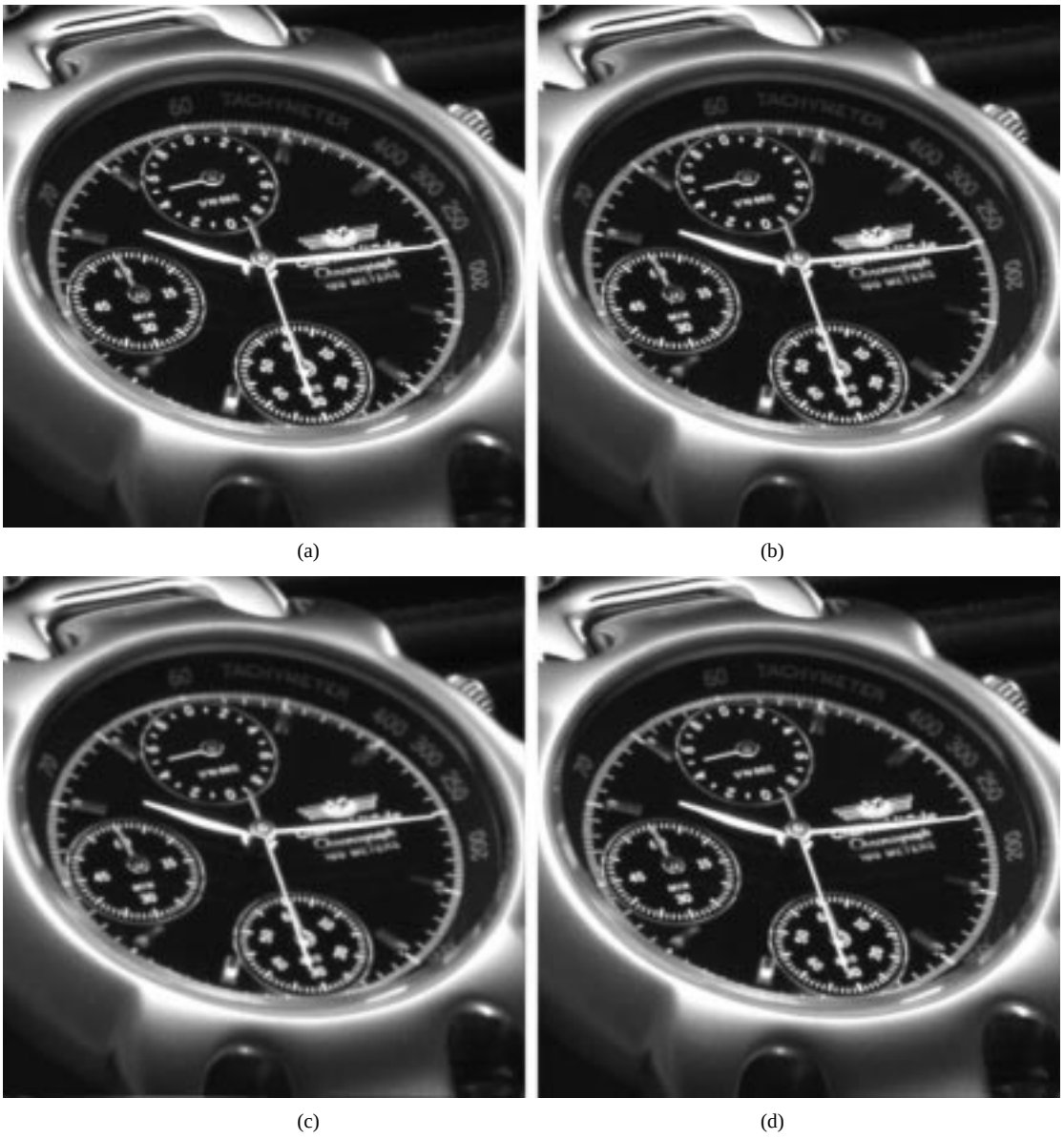
Putting $z = \exp(j\omega)$, we have

$$U(\omega) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\omega - \frac{2\pi}{N}k\right) F_k\left(\omega - \frac{2\pi}{N}k\right). \tag{36}$$

The following points can be noted from this equation.

1) Using $N = 8$ and (33), we see that the last four terms of the summation in (36) contribute the same as the first four terms shifted by π. This makes $U(\omega)$ periodic with a period of π. Hence, $Y(\omega)$ [see (34)] would be periodic with a period of 2π, as expected. We only need to look at $U(\omega)$ between 0 and π to get $Y(\omega)$.

2) The F_k's have been plotted in Fig. 4 for the case when our scheme is used for downsampling. Let us concentrate on the first four terms in (36). In the low-frequency regime (0 to $\pi/2$ and $3\pi/2$ to 2π), we see that the magnitudes of F_1 through F_3 are more than 20-dB below that of F_0. Hence, if $x(n)$ has predominantly low-pass components, we see that the shaping of $U(\omega)$ (and hence, of $Y(\omega)$) is predominantly decided by F_0 ($\equiv F_4$). Since F_0 has almost a flat response in the low-pass regime and decays fast outside that regime, we see that the low-pass components of $X(\omega)$ are preserved in $Y(\omega)$ (we shall consider phase characteristics of F_0 shortly). From (32), it is clear that F_0 is obtained by adding H_0 through H_3 successively shifted by multiples of two. These filters, along with F_0, are shown in Fig. 5 for the case when our scheme is used for downsampling. We see that F_0 is even length and symmetric, i.e., it is a type-II FIR filter [14], and hence has (generalized) linear phase. Note how the shift by multi-



(a) (b)

(c) (d)

Fig. 12. Watch image: (a) Original; (b) downsampled and upsampled by our method; (c) downsampled and upsampled using bilinear interpolation; and (d) downsampled by bilinear interpolation and upsampled by our method. We see that the image in (b) is sharper compared to that in (c), e.g., the numbers on the dial at left in (b) are much clearer compared to those numbers in (c). Also the numbers toward the rim of the watch are also much clearer in (b) than (c). The image in (d) is also sharper compared to (c) and has almost the same visual features as that in (b) even though we use bilinear scheme for downsampling. PSNR values with respect to the original are: (b) 29.09 dB; (c) 25.15 dB; and (d) 28.63 dB.

ples of two "aligns" the significant magnitude taps of H_0 through H_3.

3) In the high-frequency regime, the magnitudes of F_1 through F_3 are more than 10-dB below the magnitude of F_0 in the low-frequency regime. According to (36) to obtain $U(\omega)$, $X(\omega)$ is multiplied by F_1, F_2 and F_3 successively and shifted by $\pi/4, \pi/2$ and $3\pi/4$, respectively, and then added to the product of $X(\omega)$ and F_0. Looking at the filters F_1, F_2, and F_3 in Fig. 4, we see that the shifts by $\pi/4, \pi/2$ and $3\pi/4$, respectively, would cause all the corresponding products to have high magnitude at the knee (around $3\pi/2$) of F_0. The last four terms in (36) will produce a similar effect at the knee of F_0 at $\pi/2$. Hence, we see that if $X(\omega)$ has significant high-frequency content, then those frequencies will contribute to $U(\omega)$ at $\pi/2$ (and at $3\pi/2$ by periodicity), and hence to the output $Y(\omega)$ at π (and $-\pi$). Hence, significant high-frequency content in the input will cause aliasing in the output.
4) Now consider the F_k's for the bilinear case. Here, we can show that $F_k \equiv 0$ for $k = 1, 2, 3, 5, 6, 7$. The response of $F_0 (\equiv F_4)$ in comparison with our scheme is shown in Fig. 6. From the figure, we see that F_0 decays very slowly for the bilinear case. Hence, in this case, though F_1, F_2, and F_3 do not cause the high frequencies in $X(\omega)$ to aggregate at the knee of F_0, F_0 itself has high magnitude in the high-frequency region. Hence, if $X(\omega)$ has significant high-frequency content, we will see aliasing in the output.

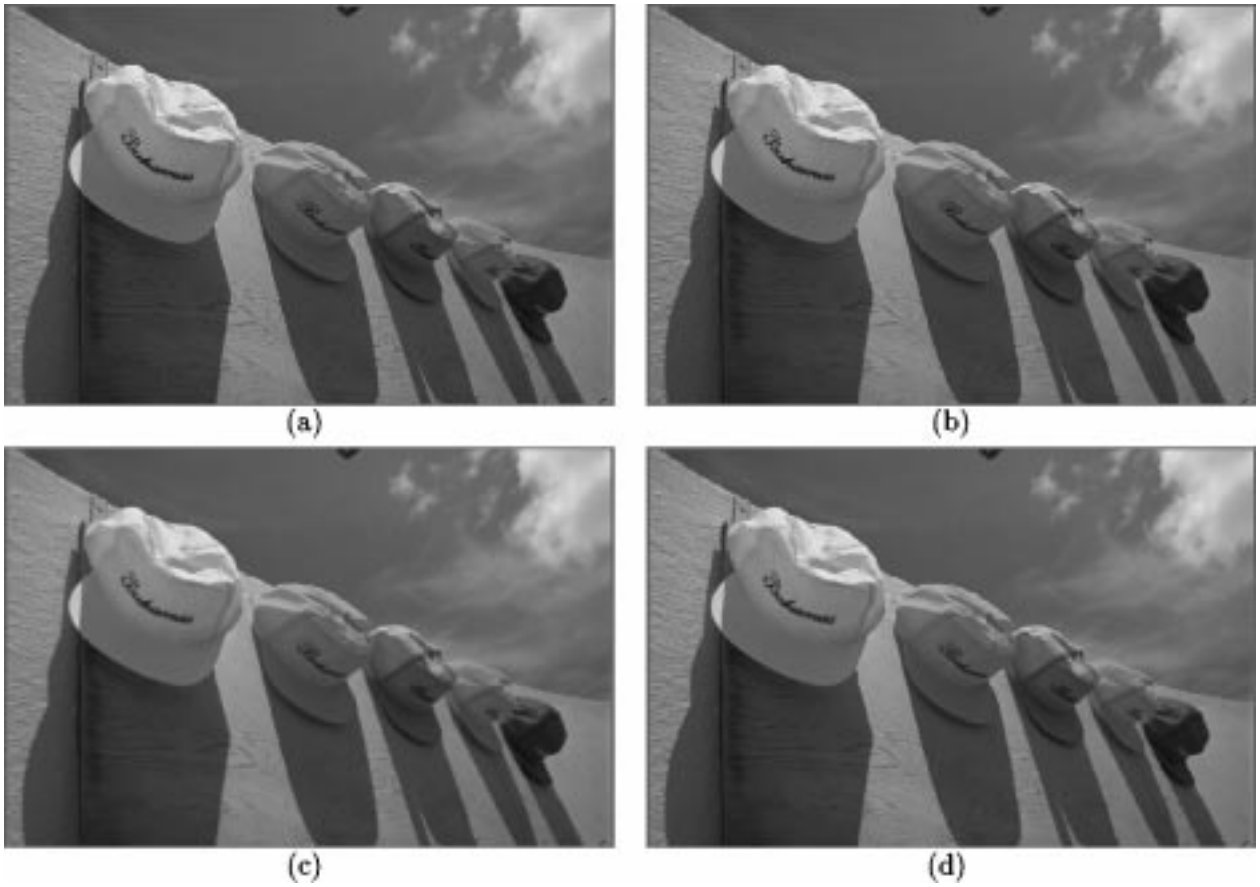


Fig. 13. Cap: (a) Original; (b) downsampled and upsampled by our method; (c) downsampled and upsampled using bilinear interpolation; and (d) downsampled by bilinear interpolation and upsampled by our method. We see that the image in (b) is sharper compared to that in (c)., e.g., the letters on the cap are sharper in (b). Also, (b) has more texture on the left side of the image. The image in (d) is also sharper compared to (c) and has almost the same visual features as that in (b), even though we use bilinear scheme for downsampling. PSNR values with respect to the original are: (b) 34.22 dB; (c) 32.08 dB; and (d) 33.73 dB.

A similar analysis can be carried out for the upsampling scheme and is not given here. The outputs for the upsampling scheme corresponding to edges of different slopes is shown in Fig. 7 (the outputs are similar, but not the same when the starting location of the step edges is changed). We see that there are ripples in the outputs corresponding to strong edges (steep slopes), but they die down as the slope of the edge is decreased, and almost no ripples are seen for the outputs corresponding to 4- and 5-pixel-wide edges.

V. Results

It is shown in the Appendix that our downsampling scheme requires 1.25 multiplications and 1.25 additions per pixel of the original image. It is also shown in the Appendix that the straightforward spatial domain approach using very fast DCT algorithm would take 3.44 multiplications and 9.81 additions per pixel and the compressed domain approach of [2] takes 4.00 multiplications and 4.75 additions per pixel. Hence, we see that our scheme is computationally very efficient. We also presented a computationally fast upsampling scheme that works in the compressed domain.

In this section, we shall see that the subjective and PSNR quality of the images obtained by downsampling and upsampling using our scheme is much better compared to bilinear interpolation. Since we preserve most of the significant energy (the low-pass coefficients of each block) of the image, the image obtained after downsampling and upsampling looks much sharper compared to the image obtained after bilinear interpolation; the latter looks blurred. Comparing the PSNR shows that the image obtained by our downsampling and upsampling scheme is typically about 4-dB better than the image obtained by bilinear interpolation.

Fig. 11(a) shows the original Lena image and Fig. 10 shows the Lena images obtained after downsampling. From Figs. 10(a)–(c), we see that the downsampled images using our scheme, the bilinear scheme, and the scheme in [8] with full-precision matrices look about the same and there are no visual artifacts. But artifacts can easily be seen in Fig. 10(d), which is obtained using the scheme in [8] with the transformation matrices' entries approximated by inverse powers of two for computational efficiency. Since the images in Fig. 10(a)–(c) look about the same, we have not shown the downsampled images for the Watch and Cap images.

Results for the upsampling scheme with Lena image are shown in Fig. 11. We see that the Lena image obtained by our method [Fig. 11(b)] is much sharper compared to the bilinearly interpolated Lena image [Fig. 11(c)]. This is particularly noticeable in the feather and eye regions. The PSNR values are

tabulated in Fig. 8. We see an improvement of 4.65 dB with our method compared to the conventional bilinear interpolation method [4], [1], [2]. Fig. 11(d) is described later.

Similar results for the Watch and Cap image are shown in Figs. 12 and 13.[6]

Now consider another possibility in which the upsampling scheme is used without any knowledge or control of the downsampling scheme. Does the upsampling scheme presented here still give visual and PSNR improvements even if the downsampling scheme used is not the one presented in this paper? For example, the user may have no control over the video frames being broadcast from a studio, but it might still be desirable to upsample those frames to fit the user's TV screen. It turns out that the upsampling scheme provides visual as well as PSNR improvement, even if the downsampling scheme used was something different (from our scheme) like bilinear interpolation. Fig. 9 gives the PSNR values when the original image is downsampled using bilinear interpolation and then upsampled using bilinear interpolation and our upsampling scheme (cf., Section III). Comparing the second rows of the tables in Figs. 8 and 9, we see that the upsampling scheme preserves the PSNR gains even if bilinear interpolation is used for the downsampling scheme. Figs. 11(d), 12(d), and 13(d) show the corresponding upsampled images. We see that the visual quality is still much better compared to bilinear interpolation scheme and is close to the case when our scheme is used for *both* downsampling and upsampling (part (b) of the corresponding figures).

Another possibility is to use our downsampling scheme and bilinear interpolation for upsampling. It turns out that in this case the visual quality, as well as PSNR values of the resulting upsampled images, is very close to the case in which bilinear interpolation is used for both downsampling and upsampling (see the "bilinear" row in Fig. 8). Hence, we have considered all four possibilities: using bilinear or our scheme for downsampling, and then using bilinear or our scheme for upsampling. We see that the upsampling scheme plays the deciding role for the final image quality. This also implies that the decision regarding which downsampling scheme to use should be made depending on the domain in which the original images are available. For example, if the original images are available in raw format, then the bilinear scheme should be used for downsampling as that would be faster. However, if the original images are available in compressed (DCT) format, then our scheme should be used as it is faster in the compressed domain. In either case, the upsampling scheme presented here should be used for upsampling to minimize the energy in the residual (difference between the original and the upsampled image) and obtain visually better image quality.

VI. Conclusion

Most previous approaches to downsampling in the compressed domain start with the problem stated in the time domain and then carry out its equivalent operation in the compressed domain. We showed in this paper that thinking of the downsampling operation directly in the compressed domain leads to computationally much faster algorithms. We proposed an algorithm which takes this viewpoint and showed that it is computationally more efficient. In particular, we showed that in our case, it is the DCT of the downsampling filter matrix which is sparse rather than the filter matrix itself being sparse (which is the case with schemes that start from the time-domain definition of the problem). The increased efficiency is due to using a filter matrix whose entries depend on the DCT basis functions. Previous approaches pick a filter matrix without regard to the characteristics of the DCT basis functions.

The downsampled image obtained by our method contains all the low-frequency DCT-coefficients of the original image. This, in turn, implies that one can obtain an upsampled image (a prediction for the original image) which contains all the low-frequency DCT-coefficients of the original image from the downsampled image. Since typical images have very little energy in their high-frequency DCT coefficients, this property implies tremendous savings for coding the residual image in scalable video coding. In particular, it is possible to design a scalable encoding scheme for video which can have the benefits of both spatial scalability and frequency scalability [15] without the attendant complexity of spatial scalability. We are currently researching on this possibility.

The method presented here mainly uses the orthogonality and symmetry properties of the DCT to claim most of the computa-

TABLE I
Computational Requirements Per Pixel of the Original Image for Downsampling in the Compressed Domain

Scheme	#multiplications	#additions
Spatial	3.4375	9.8125
Chang et al. [2]	4.000	4.7500
Our	1.2500	1.2500

TABLE II
Account of Computations

$C^t * \hat{\mathbf{B}}$	$8 \times (10M + 6A)$
$P = C^t\hat{\mathbf{B}} * C$	$4 \times (10M + 6A)$
$Q = C^t\hat{\mathbf{B}} * D$	$4 \times (10M + 6A)$
$D^t * \hat{\mathbf{B}}$	$8 \times (10M + 6A)$
$R = D^t\hat{\mathbf{B}} * C$	$4 \times (10M + 6A)$
$S = D^t\hat{\mathbf{B}} * D$	$4 \times (10M + 6A)$
Subtotal	$32 \times (10M + 6A)$

$P + Q$	$16A$
$P - Q$	$16A$
$R + S$	$16A$
$R - S$	$16A$
$\hat{\mathbf{B}}_{\{1,2,3,4\}}$(see Eqs.(40) − (41))	$4 \times 16A$
Subtotal	$8 \times 16A$

[6]Detailed results can also be seen at http://vision.ai.uiuc.edu/~dugad/draft/dct.html. The improvement in visual quality is more clearly seen here.

tional savings. Hence, it is clear that similar computational savings can be obtained even if other transforms (having similar orthogonality and symmetry properties) like Fourier or Hadamard are used. The scheme presented here works for increasing or decreasing image size by a factor of two and hence by extension works for any power of two. A large number of nonpower of two and noninteger scaling factors are possible if the output image is desired in the spatial domain. For example to obtain a signal $3/4$th the size of original signal when the original is given in terms of 8-point DCT coefficients one can apply six-point inverse DCT on the six low-frequency coefficients. Similarly, padding with zeros can be used for upsampling. Another approach for upsampling and downsampling by arbitrary ratios when the output image is desired in spatial domain in given in [16].

APPENDIX

In this section, we shall obtain quantitative estimates of the computation required by our scheme for downsampling and upsampling in the compressed domain. We are given the block-DCT (8×8) coefficients of an image and we wish to obtain the block-DCT (also 8×8) coefficients of a 2×2 downsampled version of this image. Consider the amount of computation required if a fast[7] DCT algorithm like the one in [6] is used which takes 11 multiplications (denoted $11M$) and 29 additions (denoted $29A$) for computing 1-D 8-point DCT. Since a 2-D DCT can be implemented as 1-D DCT along each row and then along each column, for 2-D 8×8 DCT we would need $176M$ and $464A$.

A. Straightforward Spatial Domain Approach

First consider the straightforward approach of taking the inverse block-DCT (involves 4 inverse 8×8 DCT), downsampling in the time domain and then taking a forward 8×8 DCT. Assume that the original image is of size 16×16 so that the downsampled image would be of size 8×8. The amount of computation is as follows.

- Four inverse 8×8 DCT: $4 \times (176M + 464A)$.
- Averaging: three additions per four pixels. Hence $3 \times 16 \times 16/4 = 192A$. Since the scaling is by four, it can be implemented with a shift and we do not count it.
- One forward 8×8 DCT: $176M$ and $464A$.
- Total: $880M + 2512A$.
- Total per pixel of original image: $3.4375M + 9.8125A$.

Compare this with the compressed domain approach of Chang *et al.* [2] based on (3). Assuming that the block-DCT of an image is 75% sparse (i.e., only the upper left 4×4 low freqrency coefficients are nonzero), their approach takes 4 multiplications and 4.75 addtions per pixel of original image (these figures are obtained by putting $\beta = 4$ and $N = 8$ in Table I in [2]). Hence, we see that for the case of 8×8 block-DCT (which is by far the most widely used), the compressed domain processing approach of [2] has a difficult time competing with the spatial domain approach due to the existance of very fast DCT algorithms tailored for the 8-point DCT.

[7]Note that the algorithm of Arai *et al.* [17], which combines the DCT and the quantizer, cannot be used here because we need to process the spatial domain pixels to get the downsampled image, i.e., we need the actual pixel values and not their quantization levels for this purpose.

B. Our Approach

Now consider the computational complexity of our approach. Actual computation shows that the matrices C and D in (21) are given below as

$$C = \begin{bmatrix} .7071 & .0000 & .0000 & .0000 \\ .0000 & .2960 & .0000 & .0162 \\ .0000 & .0000 & .0000 & .0000 \\ .0000 & .5594 & .0000 & -.0690 \\ .0000 & .0000 & .7071 & .0000 \\ .0000 & -.2492 & .0000 & .3468 \\ .0000 & .0000 & .0000 & .0000 \\ .0000 & .1964 & .0000 & .6122 \end{bmatrix}$$

$$D = \begin{bmatrix} .0000 & .0000 & .0000 & .0000 \\ .6407 & .0000 & -.0528 & .0000 \\ .0000 & .7071 & .0000 & .0000 \\ -.2250 & .0000 & .3629 & .0000 \\ .0000 & .0000 & .0000 & .0000 \\ .1503 & .0000 & .5432 & .0000 \\ .0000 & .0000 & .0000 & .7071 \\ -.1274 & .0000 & -.2654 & .0000 \end{bmatrix} \quad (37)$$

Now consider the amount of computation in computing $C \cdot V$ where V is a 4×1 vector. Since there are ten nonzero entries in C we need ten multiplications. Also there are four rows in C each containing two nonzero entries, and hence, the number of additions is four. Total: $(10M + 4A)$. Similarly for $D \cdot V$. Now consider (21)

$$X = C * (\hat{\mathbf{B}}_1 \pm \hat{\mathbf{B}}_3) \oplus D * (\hat{\mathbf{B}}_1 \pm \hat{\mathbf{B}}_3). \quad (38)$$

Here, $\hat{\mathbf{B}}_1$ etc. are 4×4. Hence, each of the $\pm$ takes $16A$. Each of the $*$ takes $4 \times (10M + 4A)$. The third and seventh rows of C (and hence those of $C * (\hat{\mathbf{B}}_1 \pm \hat{\mathbf{B}}_3)$) and the first and fifth rows of D (and hence those of $D * (\hat{\mathbf{B}}_1 \pm \hat{\mathbf{B}}_3)$) are zero. Hence, no additions are required for adding the 1st, 3rd, 5th, and 7th rows of the operands of $\oplus$. Hence, $\oplus$ involves only $16A$ instead of $32A$. Summing up the total for computing X becomes $(80M + 80A)$. Ditto for Y. Now consider computing $\hat{\mathbf{B}}$ in (20)

$$\hat{\mathbf{B}}^t = C * (X \pm Y)^t \oplus D * (X \pm Y)^t. \quad (39)$$

Here, X^t and Y^t are 4×8. Hence, each $*$ requires $8 \times (10M + 4A)$ operations. Each $\pm$ requires $32A$. As before, due to the zero rows in C and D, $\oplus$ requires only $32A$ instead of $64A$. Hence, for computation $\hat{\mathbf{B}}$ from X and Y is $(160M + 160A)$. Adding this to the computation for X and Y, we see that the overall total for $\hat{\mathbf{B}}$ is $2 \times (80M + 80A) + (160M + 160A)$, i.e., $(320M + 320A)$. Dividing this by the total number of pixels in the original image *viz.* 256, we get the count as $(1.25M + 1.25A)$ per pixel of the original image.

A summary of the computational requirements per pixel of the original image for downsampling in the compressed domain is shown in Table I.

Now consider the complexity of the upsampling scheme described in Section III. Let $P \stackrel{\text{def}}{=} C^t \hat{\mathbf{B}} C, Q \stackrel{\text{def}}{=} C^t \hat{\mathbf{B}} D, R \stackrel{\text{def}}{=}$

$D^t\hat{\mathbf{B}}C$ and $S \stackrel{\text{def}}{=} D^t\hat{\mathbf{B}}D$. Then using $T_L T_4^t = C + D$ and $T_R T_4^t = C - D$ (27)–(28) become

$$\hat{\mathbf{B}}_1 = (P+Q)+(R+S); \quad \hat{\mathbf{B}}_2 = (P-Q)+(R-S) \tag{40}$$

$$\hat{\mathbf{B}}_3 = (P+Q)-(R+S); \quad \hat{\mathbf{B}}_4 = (P-Q)-(R-S). \tag{41}$$

To compute P and Q, we first compute $C^t\hat{\mathbf{B}}$ and then compute $P = (C^t\hat{\mathbf{B}})C$ and $Q = (C^t\hat{\mathbf{B}})D$. This way we compute $C^t\hat{\mathbf{B}}$ only once. Similarly, to compute R and S, we first compute $D^t\hat{\mathbf{B}}$. Consider computing $C^t \cdot U$, i.e., $U^t \cdot C$ where U is 8×1 vector. Since C has ten nonzero entries this takes $10M$. Also, two columns of C have four nonzero entries each and other columns have only one nonzero entry. Hence, we need $2 \times 3 = 6$ additions amounting to total computation of $(10M + 6A)$. Considering that $\hat{\mathbf{B}}$ is 8×8 and $C^t\hat{\mathbf{B}}$ is 4×8, etc., we have the following account of computations. The figures in the right columns show the computations for the $*, +$ or $-$ operation in the left column—see Table II.)

Hence, our overall total is $32 \times (10M + 6A) + 8 \times 16A$, i.e., $320M + 320A$. Dividing this by the number of pixels in the upsampled image*viz.* 256, we get 1.25 multiplications and 1.25 additions per pixel of the upsampled image.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers whose comments have greatly improved the presentation of this paper.

REFERENCES

[1] N. Merhav and V. Bhaskaran, "Fast algorithms for DCT-domain image down-sampling and for inverse motion compensation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 468–476, June 1997.

[2] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1–11, Jan. 1995.

[3] B. C. Smith and L. Rowe, "Algorithms for manipulating compressed images," *IEEE Comput. Graph. Applicat. Mag.*, vol. 13, pp. 34–42, Sept. 1993.

[4] Q. Hu and S. Panchanathan, "Image/video spatial scalability in compressed domain," *IEEE Trans. Ind. Electron.*, vol. 45, pp. 23–31, Feb. 1998.

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

[6] C. Loeffer, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, vol. 2, May 1989, pp. 988–991.

[7] J. M. Adant, P. Delogne, E. Lasker, B. Macq, L. Stroobants, and L. Vandendorpe, "Block operations in digital signal processing with application to TV coding," *Signal Processing*, vol. 13, pp. 385–397, Dec. 1987.

[8] B. K. Natarajan and B. Vasudev, "A fast approximate algorithm for scaling down digital images in the DCT domain," in *IEEE Int. Conf. Image Processing*, vol. 2, Oct. 1995, pp. 241–243.

[9] K. N. Ngan, "Experiments on two-dimensional decimation in time and orthogonal transform domains," *Signal Processing*, vol. 11, pp. 249–263, 1986.

[10] W. B. Pannebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand, 1993.

[11] A. Puri and A. Wong, "Spatial domain resolution scalable video coding," in *Proc. SPIE Visual Communications and Image Processing*, Boston, MA, Nov. 1993, pp. 718–729.

[12] *Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC 13 818-2—Rec. ITU-T H.262, Nov. 1994.

[13] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, July 1992.

[14] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, Nov. 1990.

[15] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*, ser. Digital Multimedia Standards Series. New York: Chapman & Hall, 1997.

[16] S. P. Kim and W. Su, "Direct image resampling using block transform coefficients," *Signal Processing: Image Commun.*, vol. 5, pp. 259–272, May 1993.

[17] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," *Trans. Inst. Electron. Commun. Eng. Jpn.*, pt. A, vol. E71, pp. 1095–1097, Nov. 1988.

Rakesh Dugad (S'97) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, India, in 1996, the M.S. degree in electrical engineering in 1998 from the University of Illinois at Urbana-Champaign where he is currently working toward the Ph.D. degree.

He received the M. E. VanValkenburg Graduate Research Award in Electrical and Computer Engineering for 2001–2002 and the Rambus Electrical and Computer Engineering Fellowship for the 1999–2000 academic year. In the summer of 1999, he was a summer intern at the IBM T. J. Watson Research Center, Yorktown Heights, NY. His research interests are in digital image/video processing, communications, and networking.

Narendra Ahuja (F'92) received the B.E. degree (Hons.) in electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 1972, the M.E. degree with distinction in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1974, and the Ph.D. degree in computer science from the University of Maryland, College Park, in 1979.

From 1974 to 1975, he was a Scientific Officer in the Department of Electronics, Government of India, New Delhi. From 1975 to 1979, he was at the Computer Vision Laboratory, University of Maryland. Since 1979, he has been with the University of Illinois at Urbana-Champaign, where he is currently Donald Biggar Willet Professor in the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Beckman Institute. His interests are in computer vision, robotics, image processing, image synthesis, sensors, and parallel algorithms. His current research emphasizes integrated use of multiple image sources of scene information to construct 3-D descriptions of scenes, the use of integrated image analysis for realistic image synthesis, parallel architectures and algorithms and special sensors for computer vision, extraction and representation of spatial structure, e.g., in images and video, and use of the results of image analysis for a variety of applications, including visual communication, image manipulation, video retrieval, robotics, and scene navigation. He has co-authored the books *Pattern Models* (New York: Wiley, 1983), and *Motion and Structure from Image Sequences* (New York: Springer-Verlag, 1992), and co-edited the book *Advances in Image Understanding* (IEEE, 1996).

Dr. Ahuja is on the Editorial Boards of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE; *Computer Vision, Graphics, and Image Processing*; the *Journal of Mathematical Imaging and Vision*; the *Journal of Pattern Analysis and Applications*; the *International Journal of Imaging Systems and Technology*; and the *Journal of Information Science and Technology*. He is also Guest Co-Editor of the *Artificial Intelligence Journal*'s Special Issue on Vision. He received the 1999 Emanuel R. Piore Award of the IEEE and the 1998 Technology Achievement Award of the International Society for Optical Engineering. He was selected as Associate (1998–1999) and Beckman Associate (1990–1991) in the University of Illinois Center for Advanced Study. He received the University Scholar Award (1985), Presidential Young Investigator Award (1984), National Scholarship (1967–1972), and President's Merit Award (1966). He is a fellow of the American Association for Artificial Intelligence, the International Association for Pattern Recognition, the Association for Computing Machinery, the American Association for the Advancement of Science, and the International Society for Optical Engineering, and a member of the Optical Society of America.