Hybrid DCT/Pixel Domain Architecture for Heterogeneous Video Transcoding

Tamer Shanableh[1]

American University of Sharjah

P.O BOX 26666, Sharjah

UAE

Email **tshanableh@aus.ac.ae**

Tel ++971 6 5152506

Mohammed Ghanbari

University of Essex

Essex, CO4 3SQ, UK

Email ghan@essex.ac.uk

Tel +44 1206 872434

Fax +44 1206 872900

Abstract

In this paper the pixel domain heterogeneous video transcoder proposed by the authors in [1] is implemented in the DCT domain. Consequently, the motion compensation and its inverse and the image down-sampling functions of the pixel domain transcoder are implemented in the frequency domain whilst eliminating the *DCT* and *IDCT* pairs. Moreover the paper proposes two transcoding architectures. In one, the transcoder is simplified by implementing both its *MC*-loops in the DCT domain. While in the other, image decimation is realised through a modified inverse transformation of the top-left *4x4* coefficients. The input and output domains of the mentioned decimator render the decoder's and the encoder's *MC*-loops to be in the DCT domain and the pixel-domain respectively. This results in a unique hybrid DCT, pixel domain transcoding architecture. Various methods for accelerating the process of the *DCT domain* motion compensation are reviewed and classified into lossless and lossy methods. It is shown that both picture quality and performance are enhanced by utilising shared information with successive motion compensated macroblocks. The superiority of the hybrid architecture is then assessed in terms of preserving image quality, feasible functionality and tolerance to lossy acceleration of the DCT domain *MC*.

---

[1] Author for correspondence

1. Introduction

Heterogeneous video transcoding is defined as the process of manipulating coded video streams in order to meet arbitrary network or end-system constraints. The network constraints are usually bandwidth-related caused by either streaming video over heterogeneous networks or by the absence of Quality of Service (QoS) assurances. In contrast, end-system constraints manifest themselves in terms of incompatible video decoders, restricted processing power, display size or memory capacity. Therefore, heterogeneous video transcoding might include bitrate and resolution reductions of coded video whilst converting it into a different format.

Previous work on video transcoding for reducing the bitrate of coded video fall into two categories; manipulating coded coefficients and/or manipulating the spatio-temporal resolutions. Bitrate reduction through the former category includes dropping coded coefficients [2,3], coarse requantization [4] and most importantly requantization with drift correction, which was originally proposed by the co-author of this paper [5].

Despite the simplicity and the drift-free property of these transcoding techniques nevertheless, the bitrate reduction is governed by the spatio-temporal resolution of the incoming video. Therefore a more generic transcoding technique is realised through the addition of spatio-temporal resolution manipulations [6]. Reported work on this transcoding category includes reducing the temporal resolution for meeting network constrains [7] or for accommodating the limited processing power of end-users [8]. Likewise spatial manipulations include reducing the spatial resolution of coded images for smaller displays or mobile end users[1].

Clearly, the flexibility of the spatio-temporal manipulations come at the expense of additional complexity. It was shown by Assunco and Ghanbari that if the incoming and outgoing video streams differ only in bitrate then the two motion compensation (*MC*) loops of the cascaded decoder-encoder can be subtracted from each other and merged into one *DCT* domain *MC*-loop [9]. On the other hand, manipulating the spatio-temporal resolution of coded video renders the incoming prediction error and associated motion vectors absolute. Hence the incoming picture is usually reconstructed, manipulated in the pixel-domain and re-encoded after re-sampling the incoming motion vectors for reuse in the outgoing video stream.

Therefore, the purpose of this paper is to simplify and to enhance the picture quality of the latter transcoding architecture by means of implementing it in the DCT domain

without resorting to full decoding. The paper investigates the integration of DCT domain image decimation and motion compensation into the heterogeneous video transcoder of MPEG-1/2 into H.263. It will become evident that implementing the mentioned transcoder fully in the DCT domain results in a number of shortcomings hence, an intermediate hybrid DCT/pixel domain architecture is derived. The hybrid architecture benefits from the acceleration of the inverse *MC* of the DCT domain processing. Likewise the proposed architecture benefits from the flexibility inherent to the pixel-domain processing.

The remainder of the paper is organised as follows. Section 2 outlines some of the applications of the DCT domain processing. Section 3 explains the theory behind DCT domain *MC* (referred to as *MC-DCT*). In section 4, two DCT domain image decimation methods are reviewed. Based on which, two transcoding architectures are presented in Section 5. Section 6 categorises the acceleration of *MC-DCT* into two categories of lossless and lossy methods where various enhancements to the existing methods are proposed. The half-pixel accurate *MC* is discussed separately in Section 7 where special attention is given to DCT domain rounding, process acceleration and drift free transcoding. The experimental results and discussions are presented in Section 8. Finally, Section 9 concludes the paper.

2. Applications of DCT motion compensation (*MC-DCT)*

The motion compensation in the transform domain dates back to the year 1985 where the motion estimation and compensation of video coding were fully implemented in the transform domain [10,11]. The objective was to improve the accuracy of estimating the displacement of objects and hence generate a better quality over the conventional pixel domain motion estimation and compensation. In [12] a similar method for the computation of *DCT* coefficients for a block from two adjacent blocks was introduced for the interpolation of speech samples transformed into *DCT* blocks.

MC-DCT was also used for multiplexing several bitstreams into a single one for multi-point video conferencing, which might also include transform domain scaling, rotating and translation [13].

In [14] the underlying motivation for the utilisation of inverse *MC-DCT* is to accelerate the process of decompression for fast browsing of video databases. The acceleration was achieved through the partial decompression of only very few *DCT* coefficients of each block. Although the decompressed images or video is degraded in quality, it is quite acceptable for a user who is only interested in rough information

about the underlying image or video. The basic principle behind partial *MC-DCT* was originally proposed by Yeo and Liu for extracting a *dc* sequence from a coded video [15].

Other applications with trade-off between quality and process acceleration also exist. For instance, in homogenous video transcoding of MPEG-2 into lower bitrates by requantizing the *DCT* coefficients, the transcoding error of each transcoded frame is motion compensated in the *DCT* domain and added to future frames that use the current one as a source of prediction , Assuncao and Ghanbari proposed to carry out the *MC-DCT* by using approximate matrices that use only simple arithmetic operations like add and shift [9]. Again, the quality degradation resulting from such a technique is quite acceptable in the sense that it only affects the *MC* of the transcoding error rather than the image as a whole. The basic principle behind the approximation matrices was originally proposed by Natarajan and Bhaskaran for scaling down images in the DCT domain [16].

In the context of this paper, DCT domain transcoding is employed with the following objectives; first, enhancing the quality of transcoded pictures. As such, it is shown that DCT decimation outperforms low-pass filtering in the pixel-domain. Second, since DCT domain processing can distinguish the significant signal components, the respective transcoder can accelerate the *MC-DCT* by employing partial prioritised processing and lastly, simplifying the transcoding architecture by eliminating the DCT/IDCT pairs.

3 *MC-DCT* description

Figure 1 shows four neighbouring blocks b_i, $i=1,...,4$ in a reference frame. The objective is to extract the displaced block pointed at by a motion vector $\vec{v}$ to motion compensate a block in the current frame.

In the pixel domain, this is achieved by simply shifting the current pixel location by $\vec{v}$ and copying the resultant pixels to the current frame. Whereas in the *DCT* domain, the smallest data unit is a coefficient, which is meaningless on its own and has to be processed in the context of its block. Therefore, to extend the pixel domain *MC* to the *DCT* domain, the smallest unit of data must be a block of size *8x8*.

In Figure 1 it is shown that the *MC* block $\hat{b}$ can be calculated as the sum of four *8x8* blocks containing a vertical and horizontal shifts of the sub blocks s_i, $i=1…4$. This is illustrated in Figure 2.

The four blocks contributing to $\hat{b}$ can be calculated by multiplying the corresponding b_i with displacement matrices d_{hi}, d_{wi}, $i=1…4$ that perform vertical (*h*) and horizontal (*w*) shifts respectively, where the height and width of each sub block s_i define the matrices to be used. Therefore the calculation of the *MC* block $\hat{b}$ is formalised as

$$\hat{b} = \sum_{i=1}^{4} d_{hi} b_i d_{wi} \qquad 1 \le hi, wi < 8 \qquad (1)$$

The matrices d_{hi}, d_{wi} have the structure of either U_n or L_m as defined below:

$$U_n = \begin{pmatrix} 0 & I_{n \times n} \\ 0 & 0 \end{pmatrix}_{8 \times 8} \qquad (2)$$

$$L_m = \begin{pmatrix} 0 & 0 \\ I_{m \times m} & 0 \end{pmatrix}_{8 \times 8} \qquad (3)$$

Where I_{nxn} and I_{mxm} are identity matrices of size *n* and *m* respectively. The pre-multiplication by d_{hi} shifts the corresponding b_i block vertically whereas, the post-multiplication by d_{wi} shifts the resultant block horizontally.

For calculating the motion compensation block $\hat{b}$ in the *DCT* domain, the *DCT* is applied to both sides of Equation 1 as follows:

$$DCT(\hat{b}) = DCT(\sum_{i=1}^{4} d_{hi} b_i d_{wi}) \qquad 1 \le hi, wi < 8 \qquad (4)$$

By using the distributive property of matrix multiplication with respect to the *DCT*, Equation 4 can be rewritten as;

$$DCT(\hat{b}) = \sum_{i=1}^{4} DCT(d_{hi}) DCT(b_i) DCT(d_{wi}) \qquad \text{or}$$

$$\hat{B} = \sum_{i=1}^{4} D_{hi} B_i D_{wi} \qquad 1 \le hi, wi < 8 \qquad (5)$$

It should be noted however, that shifting B_i vertically followed by horizontally is equivalent to shifting it horizontally first. This is due to the associative property of matrix multiplication, namely:

$$(D_{hi} B_i) D_{wi} = D_{wi} (B_i D_{hi}) \qquad (6)$$

In later sections, it is shown how this property can contribute to the optimisation of the *MC-DCT*.

4   DCT domain image decimation

For integrating the *MC-DCT* with transcoding into lower spatial resolutions, the motion compensated pictures of the incoming bitstream are decimated whilst in the *DCT* domain. In the following sub-sections, two *DCT* domain image decimation methods are reviewed. In the first method, the simple pixel averaging and down sampling is extended to the *DCT* domain. On the other hand, the second method benefits from the *DCT* energy compaction property for modified inverse transformation and decimation.

4.1 Extending pixel averaging and down sampling

The framework of extending the pixel averaging and down sampling to the *DCT* domain was introduced by [17] and subsequently optimised for fast processing by [18] and [19].

For extension to the *DCT* domain, the pixel averaging and down sampling is performed on *8x8* block basis rather than individual pixel basis. That is, the smallest unit involved in the decimation ought to be an *8x8* block of pixels. Once this is performed, the extension to the *DCT* domain is realised through transforming all the involved blocks to the *DCT* domain.

For instance, the down sampling of four adjacent blocks b_i, $i=1...4$ into one block b on *8x8* block basis is illustrated in Figure 3. The down sampling is performed according to the following steps. First, each four adjacent pixels in a block b_i are summed up to create a new pixel. This implies that the input block is replaced by *4x4* pixels in the top-left corner , the rest of the block is padded with zeros. Second, each of the calculated *4x4* pixels are shifted according to the location of the underlying block b_i. That is, the *4x4* pixels are shifted to the top left corner in b_1 and to the top right in b_2 and so forth. Finally, the four new blocks are added and divided by four to generate the down sampled block b.

These steps are formalised by the following equation:

$$b = 1/4(q_1 b_1 q_1^T + q_1 b_2 q_2^T + q_2 b_3 q_1^T + q_2 b_4 q_2^T) \qquad (7)$$

where

$$q_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad (8) \text{ and}$$

$$q_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad (9)$$

Owing to the unitary property of the DCT, Equation 7 can be performed in the DCT domain by simply transforming all the participating *8x8* blocks to the transform domain. This method is hereby referred to as Pixel Averaging and Down-sampling or *PAD* for short.

4.2 Modified inverse transformation and decimation

While the previous *PAD* method is derived directly from pixel averaging and down sampling, in the following method, the transform coefficients pass through a proper low-pass filter prior to the decimation.

It was observed in [20] that decimation in the transform DFT domain is realised through inverse transforming of the required filtered coefficients determined by the decimation ratio. This method was extended to the DCT domain by [21] and more recently by [22]. It was shown that decimating an *NxN* block of coefficients into an *MxM* block is realised through inverse transforming of the top left *MxM* coefficients using the normalisation term of the *NxN* transform. It was also shown that such a method of image decimation combines both the low-pass filtering and the decimation into the modified inverse transformation kernel. The steps needed to decimate an *8x8* DCT block into a *4x4* block of pixels are illustrated in Figure 4.

The combination of low-pass filtering and decimation in the DCT domain has a number of useful and interesting applications. For instance, converting HDTV into a standard NTSC signal as reported in [23]. The decimation method was also used for

layered image coding employing a pyramidal structure for effective utilisation of network bandwidth or end-system limitations [22]. It was also used for layered video coding for service inter-working and error resilience[24]. In this paper, the mentioned method of decimation is employed for spatial resolution reduction in heterogeneous video transcoding where it manifests itself in a new transcoding architecture as explained in the next section.

5   Transcoding Architectures

The two methods of image decimation described differ in terms of the output domain. While the input and output of the *PAD* decimator are both in the DCT domain, the output of the Modified Inverse Transformation and decimation method (*MIT* for short) is in the pixel domain as illustrated in Figure 5.

These two decimation methods impose two different transcoding architectures. In *PAD*, the input to the decimator is a block of DCT coefficients, consequently the *MC* loop of the decoder part of the transcoder can employ *MC-DCT*. Again, the output of the decimator is also in the *DCT* domain hence the encoder part of the transcoder can likewise employ *MC-DCT*. This transcoding architecture is hereby referred to as *DCT-to-DCT* transcoder.

On the other hand, in the second architecture, the input to the decimator is in the *DCT* domain and the output is in the pixel domain. It follows that the decoder part can employ *MC-DCT* whereas the encoder part employs pixel domain *MC*. This transcoding architecture is hereby referred to as the *DCT-to-PEL* transcoder.

In this work, the *DCT-to-DCT* and *DCT-to-PEL* transcoders are compared against the pixel domain transcoder introduced by the authors in [1]. The block diagram of the latter *PEL-toPEL* transcoder is shown Figure 6. Full description of the transcoder is provided in the cited reference nevertheless, the following is a brief description of its block diagram:

I. The output of the transcoder is a low resolution H.263 video with a sequence structure of *IPPPP*…. i.e. (N=∞, M=3). Noting that the input sequence is MPEG-1/2 video coded with a *GoP* structure of (N=12, M=3) it follows that the incoming motion vectors are post-processed to suite the nature of the outgoing sequence, hence;

II. A list of candidate motion vectors is compiled per outgoing macroblock. The candidate motion vector that results in a prediction error with the minimum *SAD* is then selected.

III. Motion estimation refinement is an optional process of refining the coordinates of the best candidate motion vector.

IV. Lastly, note that the temporal resolution reduction process is eliminated in the remainder of the discussion for simplicity of presentation and for its indirect relevance to the *MC-DCT* process.

In the *DCT-to-DCT* transcoding architecture, the mentioned motion estimation refinement is removed due to its rather complex nature when implemented in the DCT domain. For instance, one pixel motion estimation refinement with half pixel accuracy might require calculating 8 *MC-DCT* blocks for the pixel accurate motion vector and an additional 8 blocks for the half pixel accuracy resulting in calculating 16 *MC-DCT* blocks. This is equivalent to *MC-DCT* of four luminance macroblocks.

Additionally, since the motion compensation and image decimation are performed in the *DCT* domain it follows that the *DCT* and *IDCT* blocks of Figure 6 are no longer needed. This results in the simplified block diagram shown in Figure 7.

Lastly, the *DCT-to-PEL* architecture is a mixture of both the transcoders derived above. While the *DCT* and *IDCT* are removed from the first *MC* loop, the output of the image decimator is in the pixel domain. As such, the processes of selecting the best candidate motion vector and motion estimation refinement can be added to the *DCT-to-PEL* architecture to produce the transcoder shown in Figure 8.

In the experimental results section, these transcoding architectures are compared against each other in terms of quality, performance and functionality.

6   Methods for accelerating the *MC-DCT*

Although the *MC-DCT* for a block $\hat{B}$ using Equation 5 above might seem computationally expensive, it should be noted that this is not always the case. For instance, in situations where the motion vector $\vec{v}$ is aligned either horizontally ($h=8$) or vertically ($w=8$), only two sub blocks need to be calculated instead of four hence, the right hand side of Equation 5 is reduced to two terms only. Moreover, if $\vec{v}$ is aligned in both directions, $w=h=8$, then the motion compensation block $\hat{B}$ is copied to the current frame without involving Equation 5.

Another factor that helps in reducing the computational complexity comes from the sparseness of the *DCT* blocks B_i especially when transcoding to low bit rates. This helps in reducing the number of multiplications and additions involved in shifting the sub blocks S_i, *i=1…4.* Additionally, since the displacement matrices are fixed it follows that they can be pre-computed and stored for later use.

Nevertheless *MC-DCT* is still considered computationally expensive and there are some hidden properties that can be exploited to enhance its performance. In this work, such methods are divided into two broad categories of lossless and lossy methods. In the former, the computational complexity is reduced whilst preserving the image quality, whereas in the latter, better performance is achieved at the expense of degrading image quality.

6.1 Lossless Methods

(a) Utilising shared information in a macroblock.

In the previous section an *8x8* block was treated as the basic unit of *MC-DCT*. Taking into account that a macroblock in the luminance part contains four *8x8* blocks, utilising shared information among them reduces the computational complexity.

To illustrate this, Figure 9 shows a *MC* macroblock containing four *MC* blocks each with four sub blocks S_{ji} , *j,i=1…4.* Each of the reference frame blocks is subscripted according to its relative location within the underlying macroblocks i.e. B_{11} is the top left block in the figure and so forth. The bi-directional arrows show where the shared information can be utilised.

Consider the computation of S_{21} and S_{12}, both sub blocks share the same underlying block B_{21} and both have the same height h. The first step in calculating either S_{21} or S_{12} is to shift B_{21} vertically by h, hence instead of repeating this step twice, the result of multiplying D_{hi} by B_{21} can be stored and reused by S_{12}. The same idea is also applicable for all the horizontal bi-directional arrows in the figure.

For vertical shifting, consider for instance S_{31} and S_{13}, both share the same underlying block B_{31}. Both blocks also share the same width w but differ in height h. In this case if B_{31} is shifted vertically first for the calculation of S_{31} then no intermediate results or information can be shared with S_{13}. Whereas if B_{31} is first shifted horizontally by w then the matrix multiplication of B_{31} *and* D_{wi} can be stored and reused for the calculation of S_{13}. Cleary this follows from the associatively of matrix multiplication as shown in Equation 6. This technique shall be referred to as as Utilising Intermediate Shared Information or *UISI* for short.

A similar technique for utilising shared information in a *MC* macroblock was proposed and formalised by [25]. For instance the computation of S_{11} and S_{21} is given by:

$$S_{11} + S_{21} = D_{h1}\, B_{11}\, D_{w1} + D_{h2}\, B_{21}\, D_{w2} \qquad (10)$$

since $D_{h1} = D_{h2}$ then

$$S_{11} + S_{21} = D_{h1}\, (\, B_{11}\, D_{w1} + B_{21}\, D_{w2}\,) \qquad (11)$$

by defining $P^0 = D_{w1} + D_{w2}$ then

$$S_{11} + S_{21} = D_{h1}\, (\, B_{11}\, D_{w1} + B_{21}\, (P^0 - D_{w1})\,) \qquad (12)$$

$$S_{11} + S_{21} = D_{h1}\, (\, B_{11} - B_{21}\,)\, D_{w1} + \underline{D_{h1}\, B_{21}\, P^0} \qquad (13)$$

In a similar derivation, S_{21} and S_{12} takes the following form

$$S_{21} + S_{12} = D_{h1}\, (\, B_{13} - B_{12}\,)\, D_{w1} + \underline{D_{h1}\, B_{21}\, P^0} \qquad (14)$$

Note that the underlined terms in Equations 13 and 14 are identical, hence calculated only once. The principle idea is also valid for the vertical bi-directional arrows of Figure 9. This technique is referred to as Utilising Shared Information or (*USI*) for short.

In contrast to *USI*, in the former *UISI* technique, the sub blocks at the corners of the *MC* macroblock i.e. S_{11}, S_{22}, S_{33} and S_{44}, are not involved in utilising shared information. However, one way of utilising them is drawn from the fact that in natural images neighbouring horizontal and/or vertical motion vectors can show high resemblance. As a result, the motion compensated macroblocks of the reference frame are aligned.. Therefore, analogous to treating aligned blocks in a *MC* macroblock as, aligned motion compensated macroblocks can likewise be treated as one unit. Hence utilising shared information among them. This is illustrated in Figure 10 where for each corner sub block the vertically or horizontally aligned *MC* macroblocks can be used for utilising shared information.

This observation is further appreciated when considering the motion compensation of the chrominance blocks in the 4:2:0 format where only one block for each colour component exists. In such a format there is no shared information unless one considers the vertically and/or horizontally aligned motion compensated blocks resulting from similar motion vectors. Therefore, as far as this technique is concerned, ignoring the resemblance of the successive motion vectors renders the utilisation of shared information in the motion compensation of the chrominance blocks absolute.

(b) Lossless Partial MC-DCT

As mentioned in the introduction, the partial *MC-DCT* was proposed in [14] for accelerating the process of decompression for fast browsing of video databases. The authors proposed a method in which only the *dc* and the first six coefficients obtained by zigzag scanning are motion compensated for each block, the technique is referred to as the *3-2-1* case. In the *DCT-to-PEL* video transcoder of Figure 8 where image decimation is realised by filtering and inverse transforming the top left *4x4* coefficients whilst discarding the high frequencies, it seems natural not to motion compensate the high frequencies in the first place. That is, for each motion compensation block, only the top left *4x4* coefficients are calculated whilst ignoring the rest.

Considering that this section deals with lossless optimisation of the *MC-DCT* it follows that the above statement is partially true. Namely, for B-pictures one can motion compensate for the *4x4* coefficients and ignore the rest without affecting the quality of other frames. This is because B-pictures are not used as a source of prediction. Whereas for P-pictures, although the motion compensation of the *4x4* coefficients does not affect the image decimation, it will in fact affect the quality of future reconstructed pictures that use the current one as a source of prediction. Therefore, in this section we only consider the partial *MC-DCT* for B-pictures where Equation 5 can be rewritten as follows:

$$\hat{B} = \sum_{i=1}^{4} T(D_{hi}B_i D_{wi})T \qquad\qquad 1 \le hi, wi < 8 \qquad\qquad (15)$$

where
$$T = \begin{pmatrix} I_{4x4} & 0 \\ 0 & 0 \end{pmatrix} \qquad\qquad (16)$$

and I_{4x4} is an identity matrix.

The computational savings in Equation 15 is further appreciated by considering the bi-directional prediction of B-pictures where each bidirectionally predicted block is reconstructed by motion compensating up to two blocks from a past and a future picture. Therefore the computational savings of Equation 15 is doubled.

In terms of implementation, the *T* matrix need not exists where otherwise it will contribute to the number of multiplications and additions. A simple implementation of Equation 15 is to compute only the first four lines of the term $D_{hi}B_i$ followed by the first four columns of $(D_{hi}B_i)D_{wi}$. In this case the result is equivalent to Equation 5 with less computational complexity as shown in later sections.

6.2 Lossy methods

The previous section considered two methods of accelerating the computation of *MC-DCT* without affecting the image quality. In this section two crude methods that trade image quality for higher computational savings are discussed.

(a) *Lossy Partial MC-DCT*

As mentioned earlier, motion compensating the top left *4x4* coefficients for the reconstruction of B-pictures results in lossless optimisation. However, consider substituting T in Equation 15 by:

$$T = \begin{pmatrix} I_{2x2} & 0 \\ 0 & 0 \end{pmatrix} \qquad (17)$$

This results in motion compensation of the top left *2x2* coefficients, which is more efficient in terms of computational complexity, but since the image decimation depends on the *4x4* coefficients then the quality of the decimated image will be degraded. Moreover, applying Equation 15 to P-pictures will also affect the quality of future pictures that uses it as a source of prediction. Nevertheless, the modified T matrix will surely accelerate the computation of the *MC-DCT*. The resultant quality degradations are shown in the experimental results section.

(b) Approximation of displacement matrices

The basic idea of the approximation as proposed by [9] is to substitute the floating-point arithmetic multiplication by basic integer operations like shift-right and integer additions. This is achieved by approximating each coefficient value in the displacement matrices by a sum of powers of twos with a maximum distortion of 1/32. For example a coefficient value of 0.316942 can be approximated to 1/4 + 1/16 which is 0.3125.

Again, as mentioned in the introduction, this technique was implemented for *motion compensation of the transcoding error whilst in the DCT domain.* In contrast, employing matrix approximation to motion compensate a whole image, rather than a transcoding error, results in a severe quality degradation as shown in the experimental results section.

7   MC-DCT for half pixel accuracy

The previous sections discussed the general case of integer accurate motion vectors. In this section various methods of *MC-DCT* for half pixel accuracy are discussed with elaboration upon associated performance and implementation issues.

When a motion vector is non-integer then each pixel is predicted from either two or four pixels from the reference frame. In terms of blocks, this is equivalent to averaging either two or four blocks shifted apart by one pixel either horizontally and/or vertically. The locations of the shifted blocks are shown in Figure 11.

Therefore, the brute-force method for calculating the *MC-DCT* for half pixel accuracy in both directions is the average of the four motion compensated blocks as follows:

$$
\begin{aligned}
\hat{B} = \tfrac{1}{4} [\; & D_h B_1 D_w &+ D_h B_2 D_{8-w} &+ D_{8-h} B_3 D_w &+ D_{8-h} B_4 D_{8-w} \; + \\
& D_h B_1 D_{w-1} &+ D_h B_2 D_{8-w+1} &+ D_{8-h} B_3 D_{w-1} &+ D_{8-h} B_4 D_{8-w+1} + \\
& D_{h-1} B_1 D_w &+ D_{h-1} B_2 D_{8-w} &+ D_{8-h+1} B_3 D_w &+ D_{8-h+1} B_4 D_{8-w} + \\
& D_{h-1} B_1 D_{w-1} &+ D_{h-1} B_2 D_{8-w+1} &+ D_{8-h+1} B_3 D_{w-1} &+ D_{8-h+1} B_4 D_{8-w+1}]
\end{aligned} \qquad (18)
$$

Where $1 \leq h, w < 8$, and the first row of the equation corresponds to the *MC-DCT* of the basic block, the second line for the horizontally shifted block, the third line for the vertically shifted block and finally, the last line for both the horizontally and vertically shifted block. This equation implies that the computational complexity of *MC-DCT* for a block with a half pixel motion vector in both directions is about four times as complex as for the case of *MC-DCT* for blocks with integer motion vectors.

To reduce the computational complexity, it was proposed in [9] to extract the shifted blocks directly from the basic block i.e. the block with the integer valued motion vector Depending on the directionality of the half pixel motion vector, this is achieved by filtering the basic block either horizontally or vertically. Therefore, Equation 5 is executed only once for the integer motion vector rather than up to four times as suggested by Equation 18. Obviously, this computational saving comes at the expense of introducing some distortions at the right and/or bottom boundaries of the filtered blocks. This is because the last row and/or column lie outside the basic block and are therefore ignored in the calculations. These distortions are reduced by exploring the fact that a macroblock in a luminance has four adjacent blocks and hence, when filtering the top left block horizontally, for example, the top right one can also be used as an input to the filter such that no distortion is introduced.

Nevertheless, when considering four adjacent blocks, the shaded boundaries in Figure 12 are still distorted as they lie outside the basic macroblock.

However, the work reported in [9] did not take into account that motion compensated macroblocks can also be adjacent to each other provided that they have the motion vector value. Therefore, adjacent blocks of successive macroblocks can contribute to the vertical and/or horizontal filtering of the underlying block resulting in no distortions at the boundaries. In Figure 13 for instance, the filtering of $\hat{B}_2$ of the first motion compensated macroblock $\hat{MB}_1$ can involve both $\hat{B}_1$ and $\hat{B}_3$ of $\hat{MB}_2$ provided that the mentioned macroblocks were motion compensated with the same motion vector value.

Referring to $\hat{B}_2$ of $\hat{MB}_1$ as $\hat{MB}_1.\hat{B}_2$ and assuming that MB_1 has a half-pixel accurate motion vector in both directions then $\hat{MB}_1.\hat{B}_2$ is calculated as:

$$\hat{MB}_1.\hat{B}_2 =$$

$$F_1^v \times (\hat{MB}_1.\hat{B}_2 \times F_1^h + \hat{MB}_2.\hat{B}_1 \times F_2^h) \ + \ F_2^v \times (\hat{MB}_1.\hat{B}_4 \times F_1^h + \hat{MB}_2.\hat{B}_3 \times F_2^h) \ (19)$$

Where the filter matrices $F_1^h, F_2^h, F_1^v, F_2^v$ (defined in [9]) are the DCT of the matrices $f_1^h, f_2^h, f_1^v, f_2^v$ as defined below.

$$f_1^h = \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix} \quad f_2^h = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} (20)$$

$$f_1^v = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \quad f_2^v = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} (21)$$

Figure 14 compares the results of the two techniques. The reviewed technique is referred to as "MB-filtering", whereas the proposed one is referred to as "Enhanced MB-filtering". As shown in the figure, making use of the successive motion compensated macroblocks helps in reducing the boundaries distortions and hence enhancing the overall picture quality.

Despite the enhancement achieved, the proposed technique is not guaranteed to entirely eliminate the distortions since motion vectors differ on region or even macroblock basis and therefore the resultant quality is inferior to that generated by the brute-force method of Equation 18.

Note that since matrix multiplication is distributive, Equation 18 can be rewritten as:

$$\hat{B} = \frac{1}{4} \; [\; (D_h + D_{h-1}) \, B_1 (D_w + D_{w-1}) + (D_h + D_{h-1}) B_2 \, (D_{8-w} + D_{8-w-1}) \qquad +$$
$$(D_{8-h} + D_{8-h+1}) B_3 (D_w + D_{w-1}) + (D_{8-h} + D_{8-h+1}) B_4 (D_{8-w} + D_{8-w+1}) \;] \qquad (22)$$

Hence, the number of matrix multiplication involved in *MC* a DCT block with half-pixel accurate motion vectors becomes equal to that of motion compensating for a pixel-accurate motion vector. Moreover, since Equation 22 is mathematically equivalent to the brute-force method of Equation 18 it follows that the resultant picture quality is identical to the top curve of Figure 14.

However, common to all half pixel *MC-DCT* methods is that, the *DCT* coefficients participating in the above equations are floating point values and therefore the dividing term i.e. multiplying by ¼ for a motion vector with horizontal and vertical half pixel accuracy, becomes a potential source of motion compensation loop mismatch between the transcoder and the destination decoder. This is because the *MC*-loop at the latter uses integer rather than floating point representations and hence different values for the divisions and rounding.

In the pixel domain *MC*, to average and round two pixels, the sum of the pixels is added to 1 prior to dividing it by 2. Equivalently, if the averaging is performed on block basis, the involved blocks are added to a third block composed entirely of ones. In the DCT domain this corresponds to scaling the *dc* value resulting from the sum of the two DCT blocks by adding it to 8. Since the *MC-DCT* employs floating-point arithmetic, the pixel domain rounding is performed by adding 0.5 rather than 1. Hence in the DCT domain this corresponds to scaling the *dc* value of the sum by adding it to 4.0. Likewise if four DCT blocks are averaged, the *dc* value of the sum shall be added

to 8.0 prior to dividing the resultant block by 4. Therefore, rounding in the DCT domain is realised through appropriate scaling of the *dc* value.

To validate this statement, Figure 15 shows that by treating the DCT rounding as if it is in the pixel domain i.e. by adding 0.5 to the sum of each two corresponding coefficients, a *MC*-loop mismatch occurs that results in a severe picture-drift. Likewise, ignoring the rounding altogether results in a higher picture-drift. On the other hand, the figure shows that by employing proper *dc* scaling, *MC*-loop mismatch is eliminated therefore, no picture drift shall occur.

8   Experimental Results

In the previous sections the optimisation of the *MC-DCT* was categorised into two broad categories of lossless and lossy methods. It was pointed out that large computational savings are associated with the lossy methods. For instance in [9] it was shown that by approximating the displacement matrices to substitute the floating point arithmetic by simple integer shifts and adds, 81% of the computational load is reduced compared to the pixel domain and associated *DCT* and *IDCT* pairs.

Likewise partial processing of the *DCT* coefficients results in 40.4% savings of computational load for the case of the motion compensating for the top left *4x4* coefficients [14]. Lastly, the utilisation of the shared information (*USI*) is 44% less complex than the brute-force method of Equation 5 as shown in [25]. Nevertheless, when considering the 4:2:0 format, this saving is only true for the luminance part whereas no computational savings are achieved for the chrominance part. On the other hand, for the proposed (*UISI*) technique that makes use of the successive aligned *MC* macroblocks, the number of matrix multiplications for a chrominance *MC* block in Equation 5 is reduced from 8 to 6 as explained in section 6.1-a. This results in up to 25% computational savings for each colour component. Moreover, combining the *USI* with the matrix approximation technique, further 13.5% computational saving is achieved as reported in [25].

This section shows and compares the results of all the discussed *MC-DCT* methods with respect to the application of heterogeneous video transcoding. The aim is to reach a fair trade-off between quality and performance for each of the proposed transcoding architectures.

Common to the following experiments, the SALESMAN sequence is MPEG coded at 1.5Mbit/s, with a spatial and a temporal resolution of 352x288 and 25f/s respectively. The sequence is transcoded into quarter spatial resolution whilst

retaining the temporal resolution. The output sequence is coded without intermediate I-frames i.e. (N=∞, M=1) at a bitrate of 250Kbit/s.

For a fair comparison between different transcoding architectures integrating different image decimation techniques, transcoded images are interpolated to the original size using an inverse technique of that used for the decimation. Thereafter, the interpolated images are compared against the original non-compressed ones. In contrast, if a result is shown for only one transcoding architecture then the original image is decimated using a similar technique to that used in the respective transcoder. The transcoded images are then compared against the non-compressed and down-sampled original input.

Since the *PAD* decimation method is derived directly from pixel averaging and down sampling, it follows that its interpolation can be realised through bilinear interpolation in the pixel domain. On the other hand, the interpolation of the *MIT* method is realised through padding the coefficients obtained by transforming a *NxN* block by (M^2-N^2) zeros to get an *MxM* block of coefficients, where *M/N* is the interpolation factor. The interpolated block is obtained by inverse transforming the *MxM* block after scaling the *NxN* coefficients by a factor of *M/N* [26, 27]. The block diagram of this method is shown in Figure 17.

To illustrate the superiority of the *MIT* decimation method, Figure 16 compares it against down sampling in the pixel domain using both low-pass filtering and pixel averaging. As mentioned above, the down sampled images are interpolated and compared against the original non-compressed input pictures. In the figure, the first picture of seven different video sequences is used; the figure shows that the *MIT* decimation and interpolation outperforms the pixel domain low-pass filtering and the pixel domain pixel averaging by up to 1 and 3 dB respectively.

8.1 *DCT-to-DCT* transcoding

As pointed out earlier, the matrix approximation technique has a more severe impact when employed for motion compensating a whole picture rather than its transcoding error. To illustrate this, Figure 18 plots the results of transcoding the SALESMAN sequence with and without employing the approximation matrices. Part-a of the figure shows that the approximation causes a *MC*-loop mismatch between the *MC*-loop of the encoder part of the transcoder and that of the final decoder. This is because the latter decoder does not employ any matrix approximations for its motion compensation process. If no periodic I-pictures are present in the incoming video

stream then the approximation matrices cause a more severe effect as shown in part-b of the figure. In the figure the incoming sequence has a GoP structure of (N=∞ and M=1) hence, the error caused by the approximation keeps on accumulating in the frame-buffers of the decoder part without being flushed out or replaced by intraframe pictures as the case with a GoP structure of (N=12, M=3).

8.2 *DCT-to-PEL* transcoding

In this transcoding architecture, the image decimation is realised through the modified inverse transformation of the low frequency coefficients. This makes it feasible for the *MC-DCT* loop at the decoding side of the transcoder to motion compensate for the low frequency coefficients whilst ignoring the rest as explained in section 6.1-b and 6.2-a. For instance when *MC-DCT* the top-left *4x4* coefficients (or *4x4* coefficients for short) for the reconstruction of the B-pictures, the resultant picture quality remains unaffected in the sense that B-pictures are not used as a source of prediction. Also note that the image decimator will only use the *4x4* coefficients and throw away the rest regardless. On the other hand, *MC-DCT* for the *2x2* coefficients will cause the picture to be decimated with less low-frequency components and hence lower quality. As for the reference pictures i.e. P-pictures, reconstructing them with only *4x4* frequency components affects the reconstruction of successive pictures and hence the quality will drop.

In part-a of Figure 19, the incoming video has a GoP structure of (N=∞, M=1), the sequence is transcoded with partial *MC-DCT* for the top-left *4x4, 3x3, 2x2* and *1x1* coefficients. The figure shows that the partial *MC-DCT* of the top-left *4x4* coefficients of P-pictures results in moderate quality degradation, whereas the partial *MC-DCT* of the *dc* coefficients on their own results in a 5dB drop in quality. The subjective degradation is shown in part-b of the figure. While the *4x4* partial *MC-DCT* of P-pictures did not result in visual artifacts, the blurring effects of *MC-DCT* of the *dc* value are evident. Note that the static parts of the picture are coded without *MC* i.e. with a MV value of nil, hence the respective DCT blocks are copied directly from the previous reference picture without partial motion compensation processing.

Clearly, if the incoming video is coded with periodic B and I pictures i.e. (N=12, M=3) then the quality degradation associated with the partial *MC-DCT* becomes acceptable. For a GoP of a given size, the larger the ratio between anchor to non-anchor pictures the lower is the error propagation. In addition since I-pictures replace

the contents of the frame-buffers flushing out any error accumulation, it follows that lossless partial *MC-DCT* with an incoming GoP structure of (N=12, M=3) results in an acceptable quality in comparison to the case of (N=∞, M=1). For instance, Figure 20 shows that different combinations of partial *MC-DCT* for B and P pictures result in moderate quality degradation when compared to the plots of Figure 19.

As for combining the partial *MC-DCT* with the array approximation technique in the *DCT-to-PEL* transcoder for an incoming GoP structure of (N=12, M=3), the latter technique will barely have any effect on the picture quality for two reasons. First, the *MC-DCT* is employed in the decoder part of the transcoder only, hence no approximation arrays are used at the encoder part. Second, as only few frequency components are motion compensated in the partial *MC-DCT* then the approximation matrices will not have a full effect as in the case of full *MC-DCT*. In part-a of Figure 21 the result of the partial *MC-DCT* is plotted with and without the use of approximation matrices, as shown the difference is marginal.

Repeating the experiment with an incoming GoP structure of (N=∞, M=1), part-b of the figure shows that combining the lossy partial processing with the matrix approximation technique results in unacceptable quality degradation.

8.3   Comparing against the *PEL-to-PEL* transcoder.

It was pointed out earlier that to compare the results of different transcoding architectures, the transcoded pictures are interpolated to the original size and compared against the original images. PSNR traces for the three transcoding architectures are shown in Figure 22. As expected, the results generated by the *DCT-to-PEL* architecture are the highest due to the *DCT* decimation method. On the other hand, the lowest quality is achieved by the *DCT-to-DCT* architecture where the decimation technique used is drawn directly from pixel averaging and down sampling without proper low-pass filtering.

Since the highest result is obtained by the *DCT-to-PEL* architecture that tolerates fast *MC-DCT* as shown in Figures 20-22, it is desirable to integrate fast *MC-DCT* in the mentioned architecture and compare it against the *PEL-to-PEL* transcoder. Again, this is shown in the same figure under the legend 'Fast *DCT-to-PEL'*. In this configuration, the transcoder is accelerated by employing the approximation matrices and the 4x4 partial *MC-DCT* for both B and P pictures. From the figure it is shown

that the *DCT-to-PEL* architecture generates the best results even with the fast *MC-DCT*.

One can conclude that the *DCT-to-PEL* architecture is more suitable for heterogeneous video transcoding for the following reasons:

1. *MC-DCT* is only used in the decoder part and hence lossy motion compensation accelerations have a less significant effect.

2. Inherent to implementing the encoder part in the pixel domain is the flexibility of integrating more functionality into the transcoder such as *MC* for selecting the best candidate post-processed motion vector and motion estimation refinement. As mentioned earlier, such functionality is less complex in the pixel domain.

3. Since the input to the image decimator is in the *DCT* domain, it is quite feasible to retain most of the image energy by employing the modified inverse transformation of the low frequencies whilst cutting off the rest.

9 Conclusions

In this paper, the pixel domain MPEG into H.263 heterogeneous video transcoder is implemented in the DCT domain. The superiority of the DCT decimation utilising the modified inverse transformation led to a unique transcoding architecture. The decoder's *MC*-loop outputs DCT domain images to the decimator, which in turn outputs a decimated pixel domain image. The pixel domain image is then motion compensated in the pixel domain. It was shown that such a *DCT-to-PEL* transcoding architecture facilitates partial prioritised DCT processing whilst tolerating lossy *MC-DCT*. In contrast, the *DCT-to-DCT* architecture proved to be more sensitive to lossy accelerations. Namely, the use of the approximate integer matrices in the encoder's *MC*-loop caused a *MC*-loop mismatch and a severe picture drift. Finally, common to both transcoding architectures it was shown that for proper *MC-DCT* with half-pixel accurate motion vectors, the *dc* component of the motion compensated blocks is scaled according to the directionality of the underlying motion vector.

References:

[1] T. Shanableh and M. Ghanbari, "Heterogeneous video Transcoding to lower spatio-temporal resolutions and different encoding formats", *IEEE Trans. on Multimedia*, 2(2), pp. 101-110 June 2000.

[2] A. Eleftheriadis and D. Anastassiou, "Constrained and general dynamic rate shaping of compressed digital video," *Proc. IEEE International Conference on Image Processing, ICIP'95*, Washington, DC, October 1995

[3] A. Eleftheriadis and D. Anastassiou, "Meeting arbitrary QoS constraints using dynamic rate shaping of coded digital video," *Proc. IEEE Int. Workshop on Network and Operating Systems for Digital Audio and Video*, pp. 95-106, Durham – USA, April 1995

[4] Y. Nakajima, H. Hori and T. Kanoh, "Rate conversion of MPEG coded video by re-quantization process," *Proc. of IEEE International Conference on Image Processing, ICIP'95,* vol. 3, pp. 408-411, Washington DC – USA, October 1995.

[5] D. G. Morrison, M. E. Nilsson and M. Ghanbari, "Reduction of bitrate of compressed video while in its coded form," *Proc. of sixth international workshop on Packet Video*, Portland-USA, September 1994

[6] H. Sun, W. Kwok and J. W. Zdepski, "Architectures for MPEG compressed bitstream scaling," *IEEE Trans. Circuits and Systems for Video Technology*, 6(2), pp. 191-199, April 1996

[7] J. Youn and M. Sun, "Motion vector refinement for high-performance transcoding," *IEEE Trans. Multimedia*, 1 (1) pp. 30-40, March 1999

[8] T. Warabino, S. Ota, D. Morikawa, M. Chanshi, H. Nakamura, H. Iwashita and F. Watanabe, "Video transcoding proxy for 3G wireless Mobile Internet Access," *IEEE Communication Magazine*, pp. 66-71, October 2000

[9] P. A. A. Assuncao and M. Ghanabari, "A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams*," IEEE Trans. Circuits and Systems for Video Technology*, 8(8), pp. 953-967, December 1998

[10] R. H. J. M. Plompen, B. F. Schuurink and J. Biemond, "A new motion compensated transform domain coding scheme," *Proc. of ICASSP '85*, Vol. 1, pp. 371-374, 1985

[11] R. H. J. M. Plompen, J. G. P. Groenveld, D. E. Boekee and F. Booman, "The performance of a hybrid video-conferencing coder using displacement estimation in the transform domain*," Proc. of ICASSP '86*, pp. 4.8.1-4.8.4, 1986

[12] W. Kou and T. Fjallbarant, "A direct computation of DCT coefficients for a single block taken from two adjacent blocks*," IEEE Trans. Signal Processing*, vol. 39, pp.1692-1695, July 1991

[13] S-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video*," IEEE Journal on Selected Areas in Communications*, 13(1), pp. 1-11, January 1995.

[14] N. Merhav and V. Bhaskaran, "Fast inverse motion compensation algorithms for MPEG-2 and partial DCT information," *HPL Technical Report* # HPL-96-53 , 1996

[15] B-L. Yeo and B. Liu, "On the extraction of a dc sequence from mpeg compressed video," *Proc. of the International Conference on Image Processing, ICIP'95,* Washington D.C., October 1995

[16] B. Natarajan and V. Bhaskaran, "A fast approximate algorithm for scaling down digital images in the DCT domain," *Proc. of the International Conference on Image Processing, ICIP*, Washington D.C., October 1995

[17] S-F. Chang and D. G. Messerschmitt, "A new approach to decoding and compositing motion-compensated DCT based images," *Proc. of ICASSP '93*, pp. V.421-V.424, Minneapolos, MN, April 1993

[18] N. Merhav and V. Bhaskaran, "Fast algorithm for DCT domain image down sampling and for inverse motion compensation*," IEEE Trans. Circuits and Systems for Video Technology*, 7(3), pp. 468-476, June 1997

[19] N. Merhav, "Multiplication-free approximate algorithm for compressed-domain linear operations on images," *IEEE Tran. Image Processing*, 8(2), pp.247-254, February 1999

[20] L. R. Rabiner, R. W. Schafer and C. M. Rader, "The chrip z-transform algorithm and its application," *The Bell Sys. Technical Journal* 48(5), pp. 1249-1292, May-June 1969

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

[22] K. H. Tan and M. Ghanbari, "Layered image coding using the DCT pyramid," *IEEE Tans. Image Processing*, 4(4), pp.512-516, April 1995

[23] K. H. Tzou and T.C. Chen, "An embedded HDTV coding scheme using decimation in frequency domain method," *Proc. of Picture Coding Symposium, PCS' 88,* pp. 14.4-1 to 14.4-2, Torino, Italy, September 1988

[24] R. Mathew and J. F. Arnold, "Layered coding using Bitstream Decomposition with Drift correction*," IEEE Trans. Circuits and Systems for Video Technology*, 7(6), December 1997

[25] J. Song and B-L. Yeo, "A fast algorithm for DCT domain inverse motion compensation based on shared information in a macroblock*," IEEE Trans. Circuits and Systems for Video Technology*, 10(5), pp. 767-775, August 2000

[26] R.W. Schafer and L. R. Rabinar, "A digital signal processing approach to interpolation," *Proceedings of IEEE*, vol. 61, pp. 692-702, 1973

[27] Y-F. Hsu and Y-C. Chen, "Digital TV video format conversion by extendible inverse DCT and its recursive algorithm for VLSI implementation*," IEEE Trans. Consumer Electronics*, 44(2), pp. 240-245, May 1998

FIGURES:
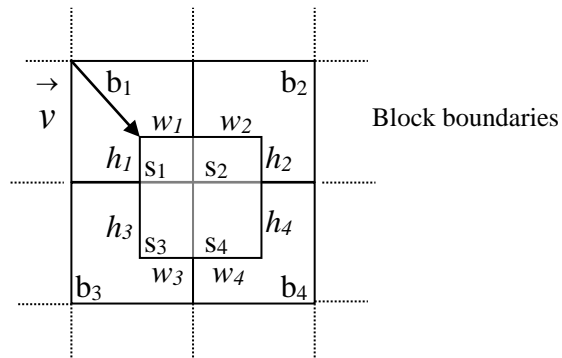
FIGURE 1. Vector $\vec{v}$ pointing at a *MC* block overlapping with 4 blocks in the reference frame.
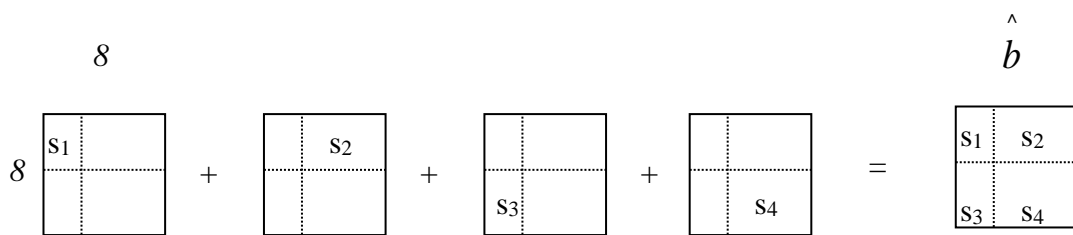
FIGURE 2. Addition of shifted versions of the *MC* sub-blocks to calculate $\hat{b}$.
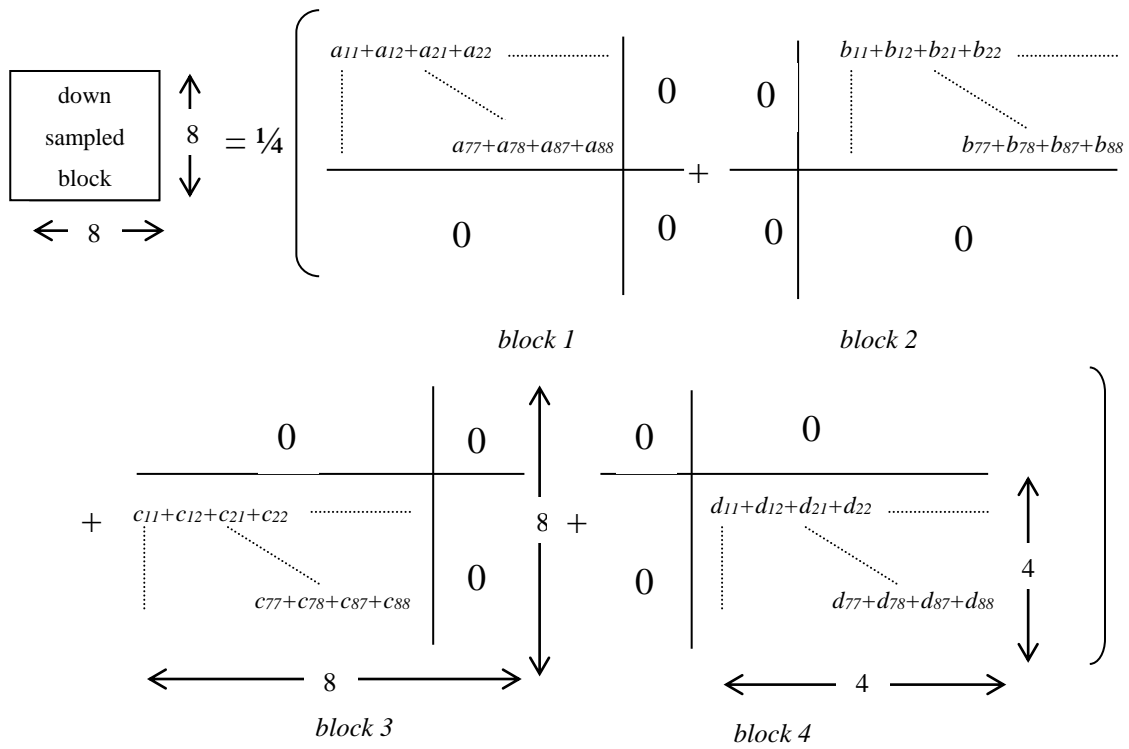
FIGURE 3. Pixel averaging and down sampling performed on 8x8 block basis.
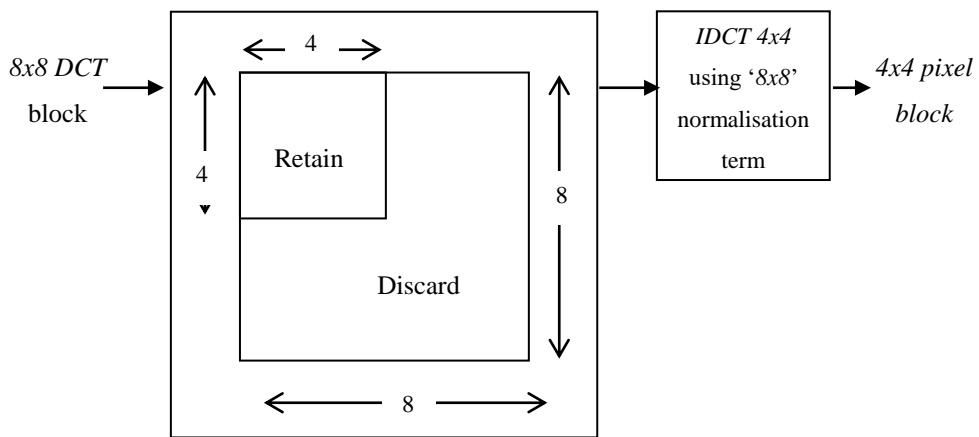
FIGURE 4. low-pass filtering and decimation into quarter spatial resolution.

26
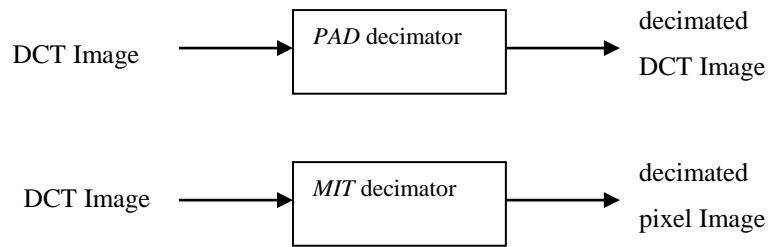
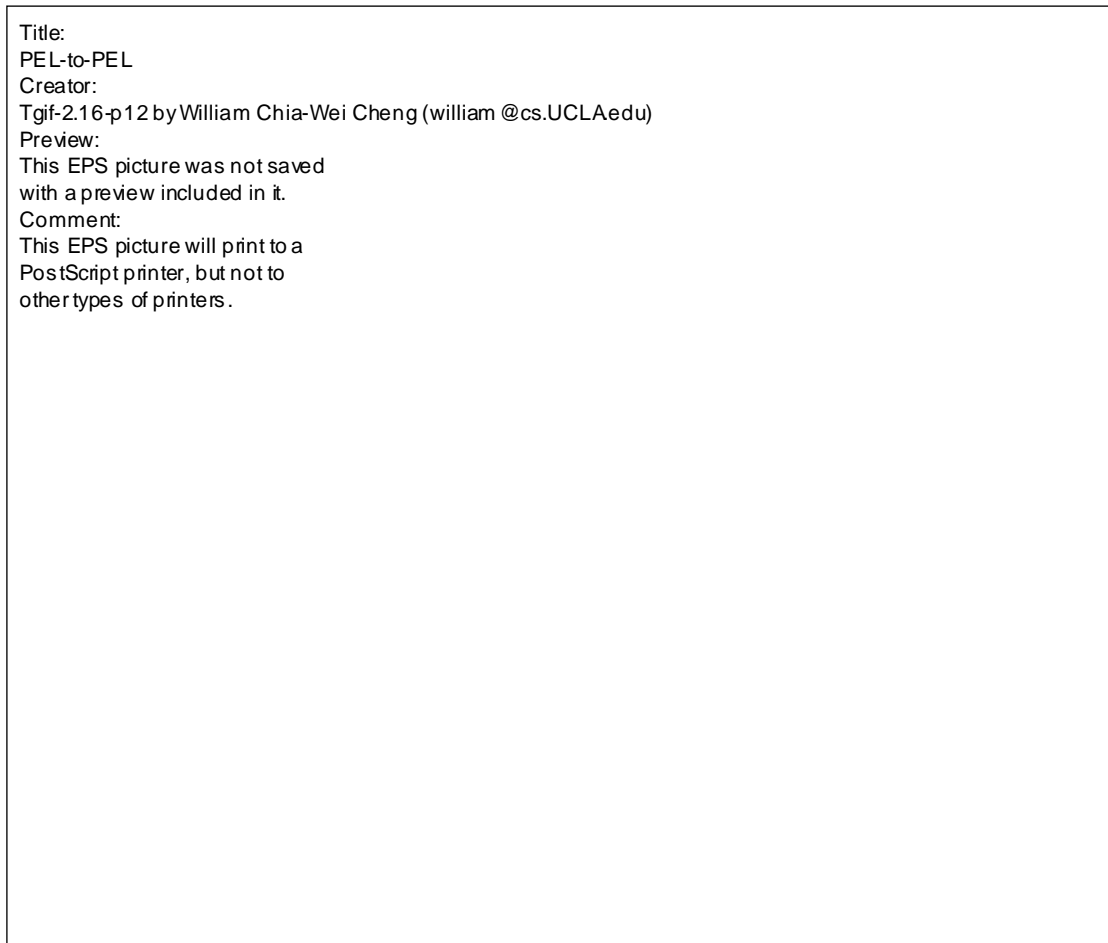FIGURE 5. Input and output domains of two decimation methods.

Title:
PEL-to-PEL
Creator:
Tgif-2.16-p12 by William Chia-Wei Cheng (william @cs.UCLAedu)
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

FIGURE 6. *PEL-to-PEL* transcoding architecture.

Title:
DCT-to-DCT
Creator:
Tgif-2.16-p12 by William Chia-Wei Cheng (william@cs.UCLA.edu)
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

FIGURE 7. *DCT-to-DCT* transcoding architecture.

Title:
DCT-to-PEL
Creator:
Tgif-2.16-p12 by William Chia-Wei Cheng (william@cs.UCLAedu)
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

FIGURE 8. *DCT-to-PEL* transcoding architecture.
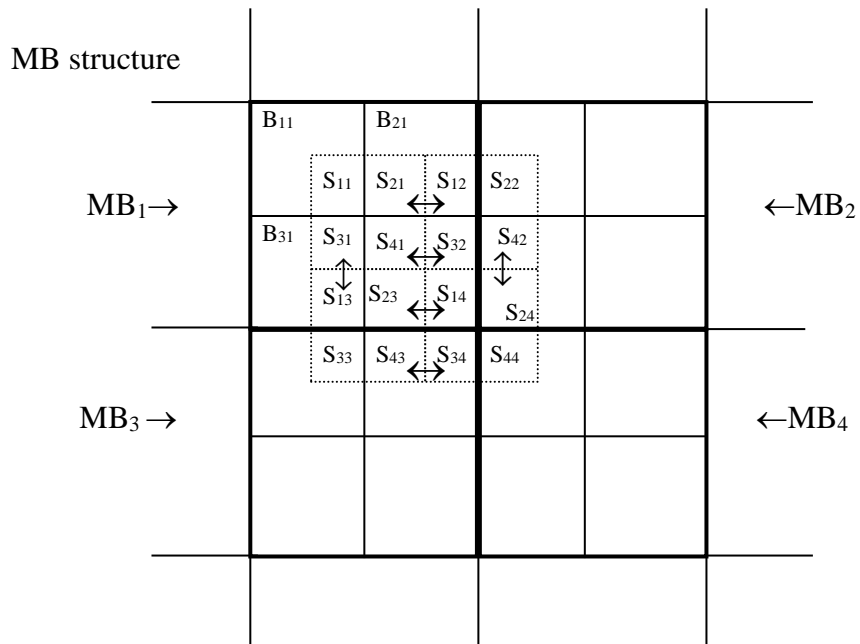
FIGURE 9. *MC* of four blocks utilising the shared information in a *MC* macroblock.
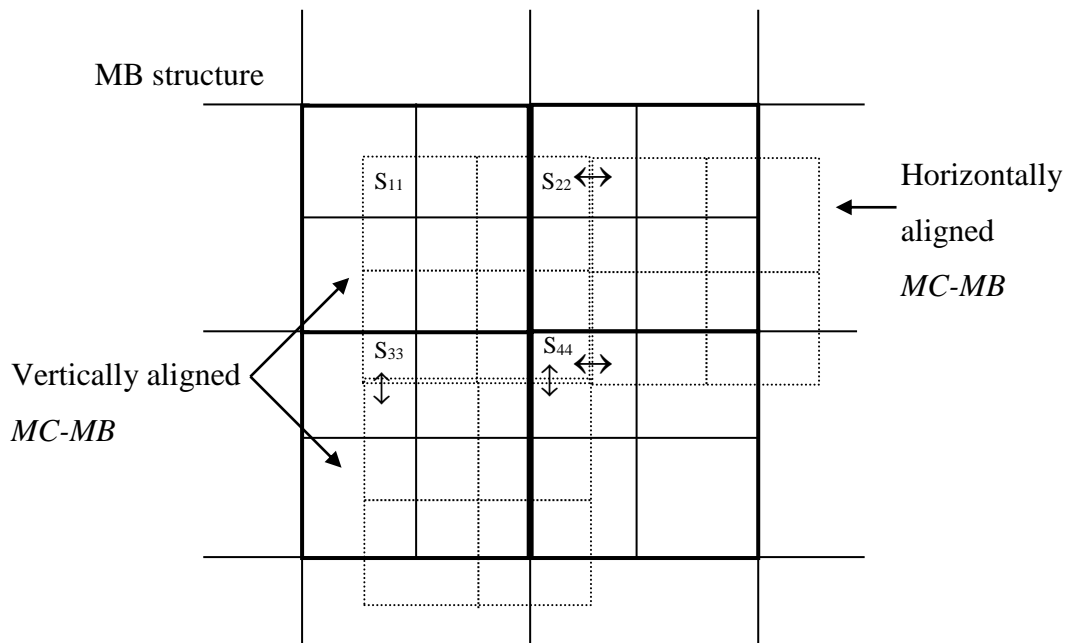
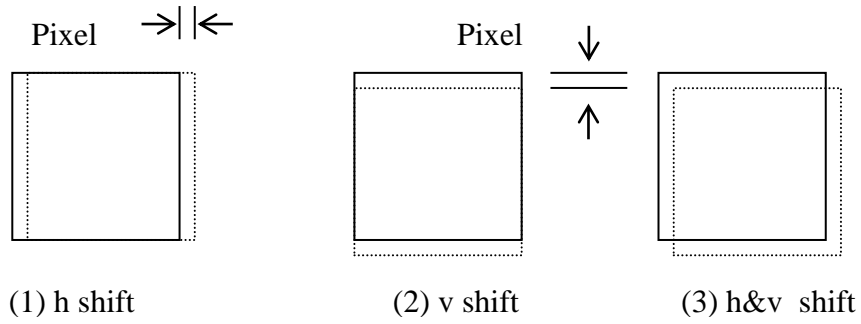FIGURE 10. Utilising shared information among aligned *MC-MBs*.

Pixel →||←

(1) h shift

Pixel ↓
↑

(2) v shift

(3) h&v shift

FIGURE 11. Shifted blocks involved in the basic block prediction.
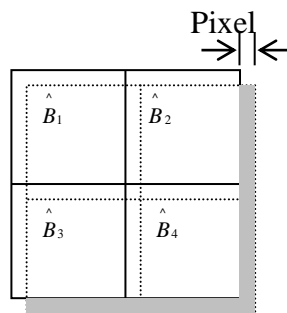
$\hat{B}_1$ $\hat{B}_2$

$\hat{B}_3$ $\hat{B}_4$

Pixel →||←

FIGURE 12. Distortion introduced at the right and bottom boundaries due to horizontal and vertical filtering on macroblock basis.
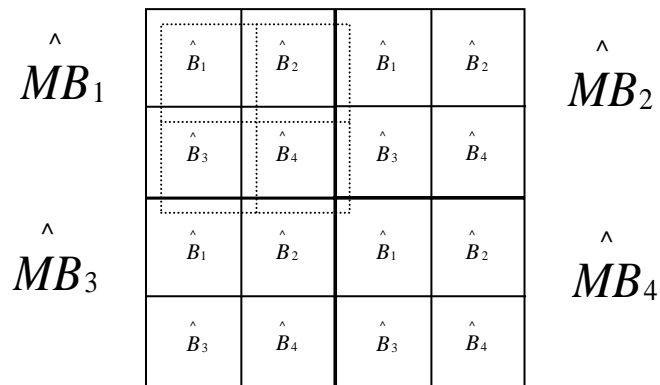
$\hat{MB}_1$

$\hat{B}_1$ $\hat{B}_2$ $\hat{B}_1$ $\hat{B}_2$

$\hat{B}_3$ $\hat{B}_4$ $\hat{B}_3$ $\hat{B}_4$

$\hat{MB}_2$

$\hat{MB}_3$

$\hat{B}_1$ $\hat{B}_2$ $\hat{B}_1$ $\hat{B}_2$

$\hat{B}_3$ $\hat{B}_4$ $\hat{B}_3$ $\hat{B}_4$

$\hat{MB}_4$

FIGURE 13. Involving adjacent motion compensated macroblocks in the filtering of the current motion compensated block.
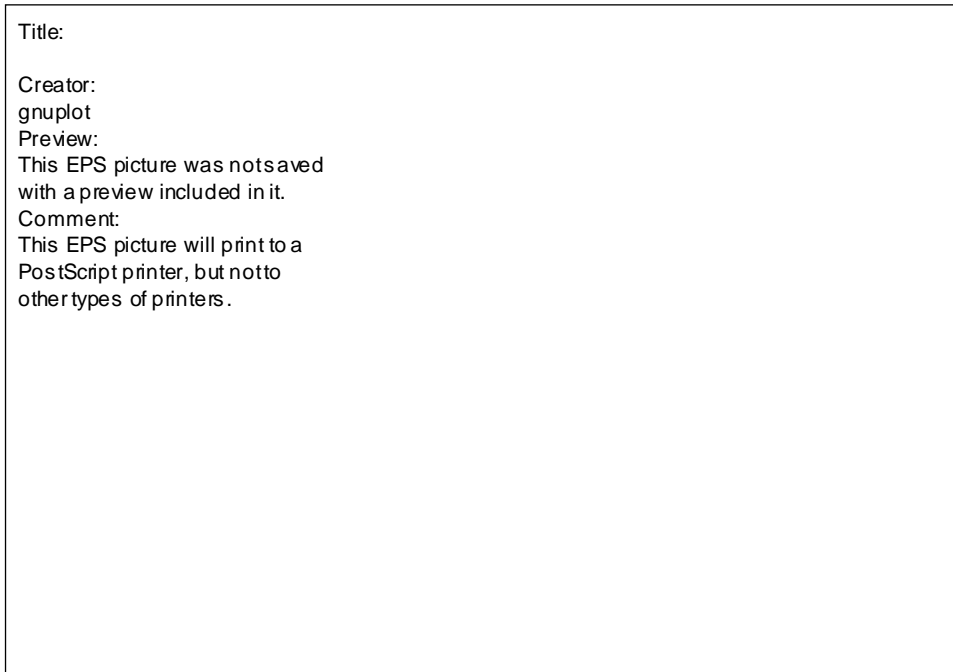
FIGURE 14. Enhancing Macroblock filtering for half-pixel *MC*. SALESMAN sequence encoded at 1.5Mbit/s, 25Hz (352x288), (N=12,M=3) transcoded into quarter spatial resolution at 250kbit/s, 25Hz (IPPPP…).

FIGURE 15. Rounding in the DCT-domain for half-pixel accurate *MC*. SALESMAN sequence, same parameters as Fig 14.

Title:
decimations
Creator:
Tgif-2.16-p12 by William Chia-Wei Cheng (william@cs.UCLA.edu)
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

FIGURE 16. The superiority of the *MIT* decimation and associated interpolation.
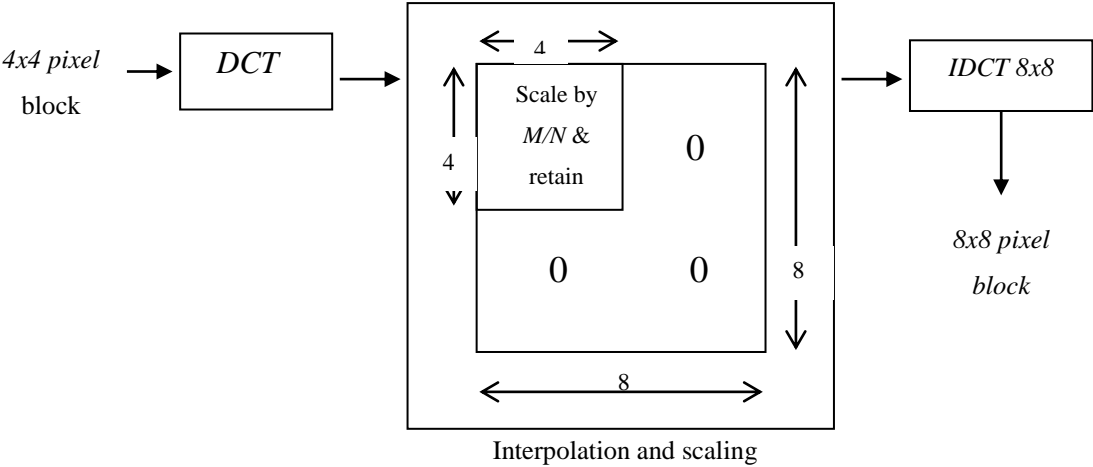

4x4 pixel block → DCT → [Scale by *M/N* & retain | 0 | 0 | 0] Interpolation and scaling → IDCT 8x8 → *8x8 pixel block*
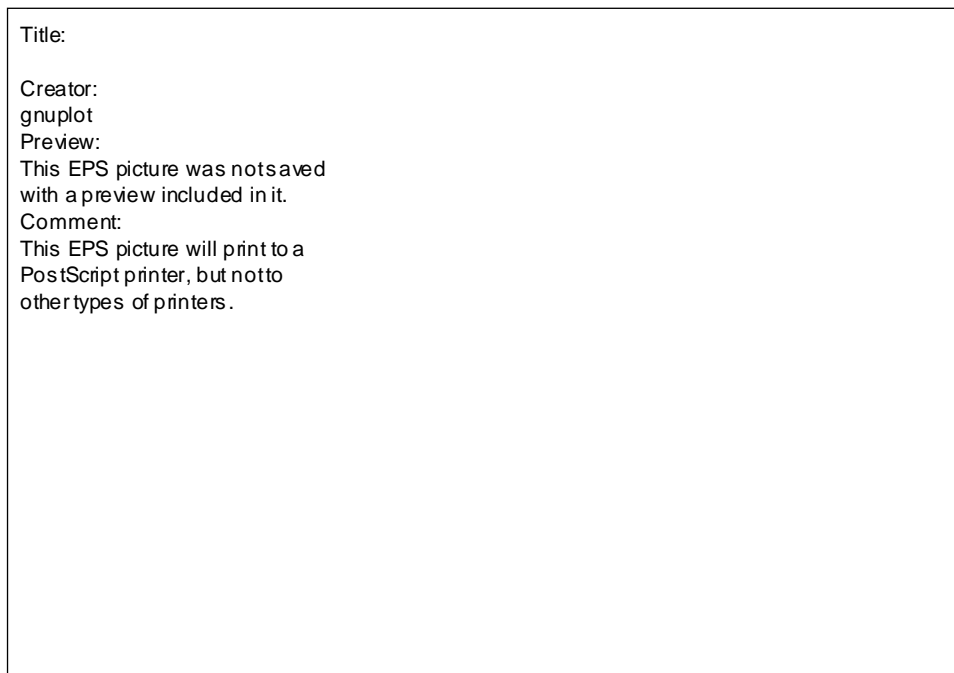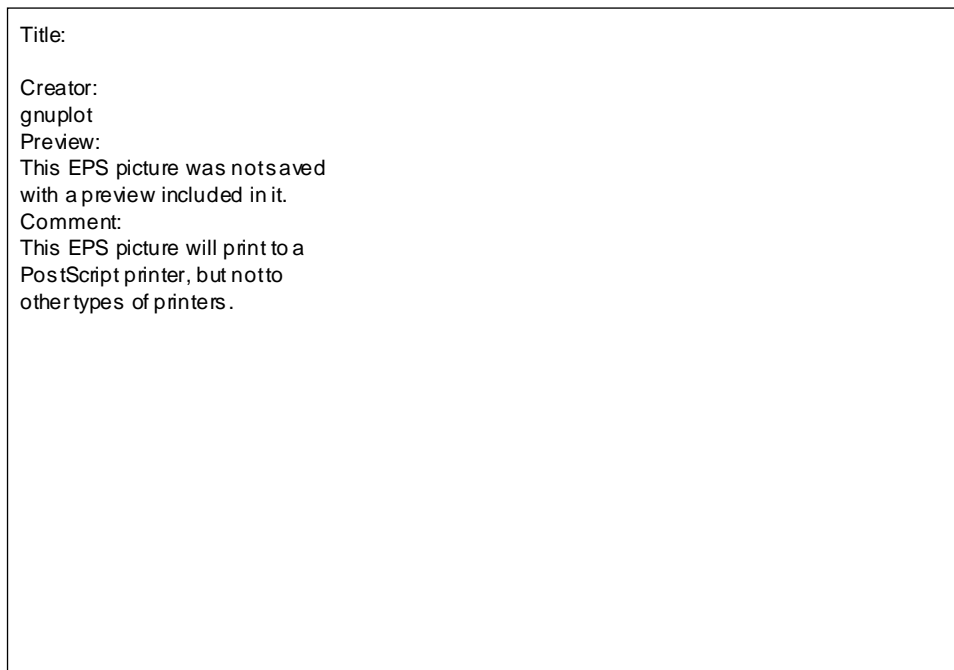
FIGURE 17. Interpolation of the *MIT* decimation method.

(a)

(b)

FIGURE 18. The effect of the approximating the *MC-DCT* matrices whilst transcoding a GoP structure of (a) N=12, M=3 (b) N=∞ M=1.

Title:

Creator:
gnuplot
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

(a)


| without partial processing | Top-left 4x4 | Top-left 1x1 (DC) |

(b)

FIGURE 19. Partial *MC-DCT* for P-pictures with an incoming GoP structure of (N=∞, M=1) (a)Objective PSNR plots (b) associated visual artifacts for picture number 90.

FIGURE 20. Partial *MC-DCT* for B and P pictures with an incoming GoP structure of (N=12, M=3)

(a)

Title:

Creator:
gnuplot
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.
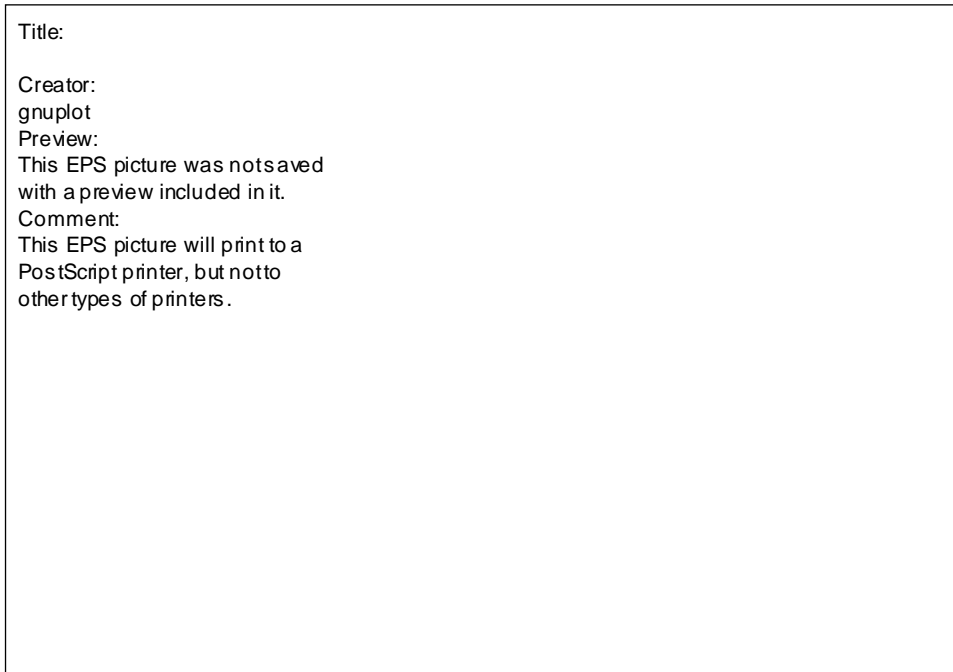
(b)

FIGURE 21. The effect of matrix approximation combined with partial *MC-DCT*.
Incoming GoP structure of (a) N=12, M=3 (b) N=∞, M=1

Title:

Creator:
gnuplot
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

FIGURE 22. Comparing the picture-quality of the three transcoding architectures.