6.2

Interframe Subband/Wavelet Scalable Video Coding

John W. Woods,
Peisong Chen,
Yongjun Wu, and
Shih-Ta Hsiang
Rensselaer Polytechnic Institute

1 Introduction

MC-EZBC [1] is one of a family of motion-compensated (MC) subband/wavelet coders that exploit temporal correlation but are fully embedded in quality/bit-rate, spatial resolution, and frame rate. The name EZBC stands for embedded zero-block coder. Figure 1 shows the basic structure of the MC-EZBC coder. This chapter will use this coder as an example of the class of fully embedded subband/wavelet transform (SWT)[1] coders that use an MC temporal filter (MCTF).

In Fig. 1, the incoming data is operated on first by the MCTF, followed by spatial subband/wavelet transform (SWT), and then an EZBC entropy coder. By using the MCTF, this system does not suffer the drift problem exhibited by hybrid coders that have feedback loops. The MCTF structure is shown in Fig. 2 for a group of pictures (GOP) of size 16, yielding $4+1$ temporal levels or frame rates. Since the complete motion-compensated (MC) temporal transform comes before any of the spatial transform, this structure has been labeled "$t+2D$." Order matters here due to the nonlinear effects of the motion estimation/compensation on the MCTF.

In MC-EZBC we have used the two-tap Haar filter in the MCTF, but longer filters, such as the LeGall and Tabatabai 5/3 are currently of interest, and can offer a modest boost in performance [8]. In [1] half-pixel accurate MCTF with perfect reconstruction was realized and significant coding

[1]More commonly called discrete wavelet (DWT).

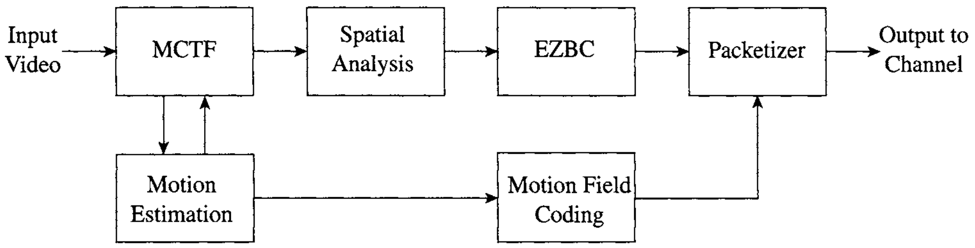


FIGURE 1 Basic structure of MC-EZBC.

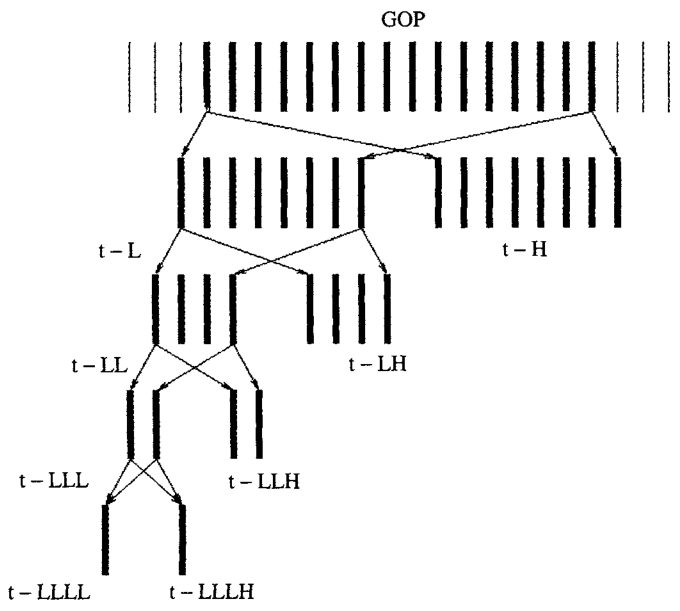


FIGURE 2 Octave based five-band temporal decomposition.

gain was observed. The use of half-pixel accurate motion compensation has also achieved substantial coding gain with respect to the use of pixel accuracy in MPEG2 [2] and H.26L [3], where even 1/8 pixel accurate motion compensation can be utilized. Sub-pixel accurate motion compensation is more useful for low spatial resolution video. There, the spatial sample rate is near to or below the Nyquist rate, so the power spectrum is often flatter, leading to the need for increased motion-vector accuracy.

Both MPEG-2 and H.26L are hybrid coders and the Ohm [4, 39] designed an invertible MCTF with half-pixel accuracy by incorporating the spatial interpolation as part of the temporal subband/wavelet filtering. More recently, lifting implementations of subband/wavelet filters have been used for MC temporal subband decomposition [5–7] to provide invertibility for arbitrary interpolation schemes. In all these approaches, the chosen spatial interpolation is incorporated into the perfect reconstruction subband/wavelet decomposition, effectively making the MC temporal filter somewhat spatial also. The use of a lifting implementation makes this incorporation easy, even for any kind of motion field. However, there is no reason to assume that the resulting motion-compensated SWT should be optimal in any sense [36], even though it is perfectly reconstructible in the absence of quantization errors.

In the sequel, we first cover motion estimation including detection of covered/uncovered pixels and the use of color information in Section 2. This is followed in Section 3 by a discussion of MC temporal filtering including the lifting implementation for invertibility and directional IBLOCKs for intraframe spatial coding. Section 4 introduces the use of overlapped block motion compensation (OBMC) in the context of MCTF coding. We then discuss scalable motion vector coding in Section 5 as it has been found to be necessary to keep efficiency high at low bit rates. Section 6 then presents the lossy entropy coder EZBC and provides some comparison to JPEG 2000. Section 7 discusses the problem of comparing scalable coders and introduces the cross-check method. Scalable coders may need multiple adaptations on the internet and that is presented in Section 8. Section 9 then shows some visual results. Related coders are touched upon in Section 10, and Conclusions follow.

2 Motion Estimation

A more detailed system diagram, focusing on the MCTF, is shown in Fig. 3. The original frames in one GOP serve as the input. The size of the GOP can vary, but usually is in the range 8 to 64, the latter being about 1.1 seconds at 60 fps. A large GOP is only efficient if the motion is largely coherent over the corresponding time interval. On the other hand, the algorithm-inherent transmission delay goes up with GOP size. Next we provide details of the system blocks in Fig. 3, starting with the first block motion estimation.

We can see in Fig. 2, that motion vectors are only needed for successive pairs of input frames for the Haar filter. In fact, the total number of motion vector estimations, totaled down the temporal levels, equals the number of frames [17]. Thus the set of motion vectors V'_{xx} in Fig. 4 is only used for the prediction of motion vectors in the next lower temporal level [8]. For example, we estimate the additional motion vectors V'_{00} but do not code them. Only $V'_{00} + V_{01}$ will be used as the starting point for the estimation of, V_{10} which then can be estimated in a reduced search. Namely if the

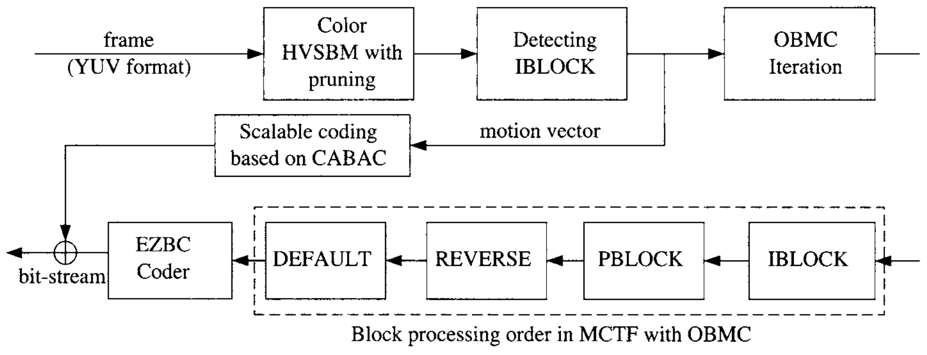


FIGURE 3 Functional block diagram for MC-EZBC.

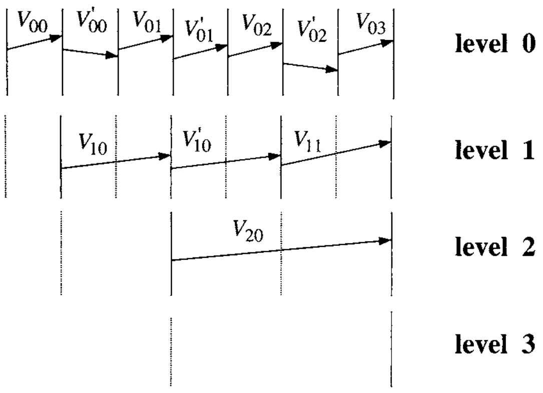


FIGURE 4 Motion estimation in a GOP with size 8 frames. While the V_{xx} are estimated and coded, the V'_{xx} are estimated but not coded, and are just used in the initial estimate at the next lower temporal level.

original HVSBM search range is $\beta \times \beta$ pixels, we start a search from $V'_{00} + V_{01}$ in a reduced range of $\beta/2 \times \beta/2$ pixels.

Motion estimation is first done through hierarchic variable-size block matching (HVSBM), with block sizes varying from 64×64 down to 4×4 for each pair of frames as shown in Fig. 5. Block matching is a good choice for practical applications and the range of block sizes can achieve a fairly accurate motion field portrayal. The hierarchic aspect offers a computational savings and tends to reduce "noise" in the resulting motion vectors.

When deciding whether to split a motion block or not, it is appropriate to augment the spatial error, often chosen as mean absolute difference (MAD) weighted with a term involving the motion vector coding rate,

$$MAD + \lambda_{mv} R_{mv},$$

and then choose whether to split or not based on this criteria. In fact, this optimization can be combined with the motion-vector mode determination [37] treated in Section 3.7.

2.1 Detection of Covered and Uncovered Pixels

When we do motion estimation, we find the motion vector for each pixel of the predicted frame. There are good matches and bad matches based on their displaced frame differences (DFD). If we have temporal filtering between two poorly matched pixels, not only do we get a DFD with high energy, but also we get a lower frame rate video with poor visual quality. In the predicted frame, several pixels may choose the same reference pixel (see Fig. 6). One of the reasons for this multi-connection is the occlusion problem [9]. For two consecutive frames, if we let the second frame A be the reference, those pixels in the first frame B, which will be covered or unknown in the second frame, will be called *to-be-covered* pixels. These to-be-covered pixels are most probably visible in frame A's preceding frame. If we let frame A be the reference, then those pixels in frame B, which are revealed and not present in frame A, are named *uncovered* pixels. For them,

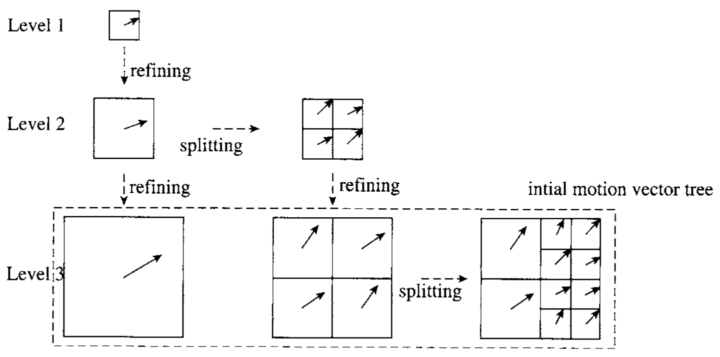


FIGURE 5 A 3 level HVSBM showing 3 subband levels.

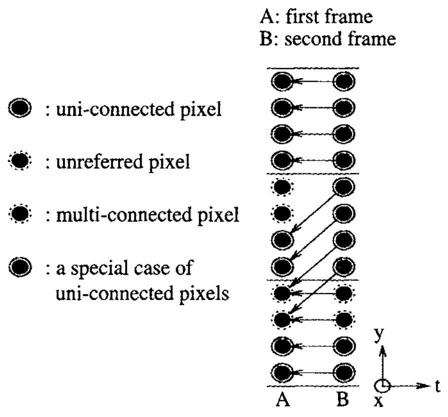


FIGURE 6 Possible connections between pixels in neighboring frames.

frame *B*'s immediately following frame most probably contains their matched regions. Thus sometimes we will have to look in the opposite directions to get the match.

The motion estimation in MCTF coders is most often bidirectional, using one previous reference frame and one future reference frame. In MC-EZBC, after we form the full motion vector quad-tree, if we detect there are more than 50% "unconnected" pixels with the algorithm in [10] in a given block, the block is classified as a REVERSE block and a good match for it is searched for in the past frame. We also search for good matches in the past frame for those blocks with high distortion after motion compensation from the future frame, and they are classified as PBLOCKs and IBLOCKs after HVSBM. The better of the two matches is chosen as the reference block for the PBLOCKs, and the block mode is changed accordingly. When there is no good match, then IBLOCK mode is chosen. When this matching results in too many "unconnected" pixels, we decide to discontinue further temporal analysis for that frame pair based on a threshold value. A large number of "unconnected" pixels can be used to trigger the *adaptive GOP size,* meaning that the GOP terminates with this frame. An alternative is to do another level of temporal decomposition, but at a reduced spatial resolution [10].

It is possible to have complete temporal frames that have not been updated, called unconstrained or UMCTF in [11]. This corresponds to omitting the update step in the lifting implementation, i.e., leaving the temporal low frame unchanged from the next higher level. This method can be employed to reduce the delay of MCTF coders, but generally suffers a penalty in PSNR [12], especially at high qualities and bit rates.

2.2 Using Color Components for Motion Estimation

In the MC-EZBC coder, we have begun using chrominance data *U* and *V* in addition to the luminance data *Y* in order to get a more stable motion field. For this *color HVSBM* [14], we use original or sub-sampled *U* and *V* frames. The computation time for color HVSBM is 1.5 times that of HVSBM running on the luminance data alone. Each dimension of *U* and *V* is half that of *Y* in the chosen 4:2:0 color space and we need sub-pixel accuracy motion vectors, but we keep the full accuracy for the *U* and *V* motion vectors, and saturate this accuracy at one eighth pixel.

After we form the full motion vector quad-tree with bidirectional color HVSBM, we prune it back in the sense of MV rate–error distortion optimization using a fixed λ. We introduced an elementary type of scalable motion vector coder [13] based on context-based adaptive binary arithmetic coder (CABAC) [15] with respect to temporal, SNR and resolution as will be described in the following.

3 MC Temporal Filtering

The MCTF plays an essential role in motion compensated 3D SWT coding. It will influence the coding efficiency and temporal scalability. Since MCTF is a kind of SWT, we can implement MCTF in the conventional (transversal) way or by using the lifting scheme [20]. The lowpass and highpass filtering in the conventional implementation can be looked at as a parallel computation, but the lowpass and highpass filtering in the lifting implementation of this transform is a serial computation. For integer-pixel accurate MCTF, these two schemes are equivalent, but for sub-pixel accurate MCTF, the lifting scheme can provide some nice properties. In the following, we will compare these two schemes in sub-pixel accurate MCTF. Before that, we have to define connected pixels in the case of sub-pixel motion vectors. When a motion displacement (d_m, d_n) from the second frame *B* points to a sub-pixel position in the first frame *A*, we can say $B[m,n]$ is connected to $A[m-\bar{d}_m, n-\bar{d}_n]$, where $\bar{d}_m$ and $\bar{d}_n$ are defined as following. If (d_m, d_n) points to a sub-pixel position, $(\bar{d}_m, \bar{d}_n) = ([d_m], [d_n])$ where $[[]]$ denotes the nearest integer function (nint). All the pixels in uncovered blocks will be left for later.

3.1 Noninvertible Approach

This approach was proposed by Ohm [16] and extended by Choi and Woods [17]. In this approach, the subband analysis pair for connected pixels is

$$H[m,n] = \left(B[m,n] - \tilde{A}[m-d_m, n-d_n]\right)/\sqrt{2} \qquad (1)$$

$$\begin{aligned} L[m-\bar{d}_m, n-\bar{d}_n] = \Big(&\tilde{B}[m-\bar{d}_m+d_m, n-\bar{d}_n+d_n]. \\ &+A[m-\bar{d}_m, n-\bar{d}_n]\Big)/\sqrt{2} \end{aligned} \qquad (2)$$

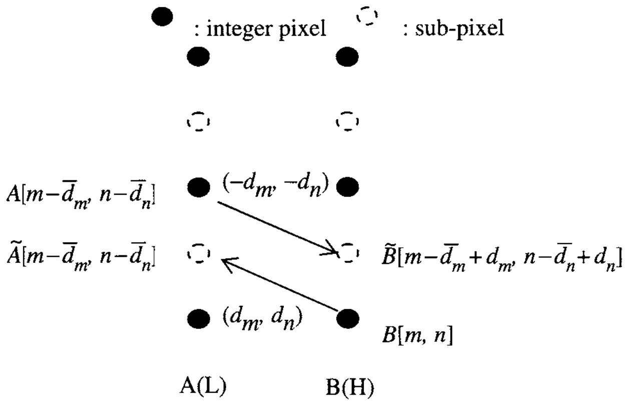

: integer pixel
: sub-pixel
$A[m-\bar{d}_m, n-\bar{d}_n]$
$\tilde{A}[m-\bar{d}_m, n-\bar{d}_n]$
$(-d_m, -d_n)$
$\tilde{B}[m-\bar{d}_m+d_m, n-\bar{d}_n+d_n]$
(d_m, d_n)
$B[m, n]$
A(L)
B(H)

FIGURE 7 Sub-pixel accurate MCTF (Choi and Woods' scheme).

The highpass coefficient comes from the filtering of $B[m,n]$ and the interpolated reference pixel $\tilde{A}[m-d_m, n-d_n]$. Since we want the lowpass coefficients to be at integer pixel positions, we do the lowpass filtering between $\tilde{A}[m-d_m, n-d_n]$'s closest integer pixel $A\left[m-\bar{d}_m, n-\bar{d}_n\right]$ and the interpolated pixel $\tilde{B}\left[m-\bar{d}_m+d_m, n-\bar{d}_n+d_n\right]$. Here we use the inverse (reverse) of $B[m,n]$'s backward motion vector as $A\left[m-\bar{d}_m, n-\bar{d}_n\right]$'s forward motion vector. In general, this scheme is not invertible [18]. Since we need to use H and L to reconstruct A, but H only contains the information of interpolated pixels in A, if this interpolation itself is not invertible, we cannot reconstruct frame A exactly.

Unconnected pixels in B are processed like (1), and unconnected (unreferred) pixels in A are processed as

$$L[m,n] = \sqrt{2}A[m,n]. \tag{3}$$

This open loop prediction is not good, but fortunately there are not a lot of unconnected pixels present, at least at the first few stages of temporal decomposition.

3.2 Invertible Half-Pixel Accurate MCTF

Hsiang, Woods, and Ohm [4, 39] designed an invertible half-pixel accurate MCTF. In this approach a composite block is constructed by merging a pair of linked motion blocks when the motion vector points to a half-pixel position. Then the composite block is decomposed by a two-channel subband analysis filter bank and the lowpass output and the highpass output are put into the temporal low subband and the temporal high subband respectively. So perfect reconstruction can be realized. Effectively the desired spatial interpolation has been incorporated into the subband/wavelet temporal filter.

3.3 Lifting Implementation

The so-called lifting scheme is a new realization for SWT [20]. This implementation has been introduced into sub-pixel MCTF to realize perfect reconstruction [5–7], as mentioned above. In the following, we first take a look at this implementation for connected pixels. If motion vectors have sub-pixel accuracy, the lifting scheme gets $H[m,n]$ in the same way as the conventional method [17],

$$H[m,n] = \left(B[m,n] - \tilde{A}[m-d_m, n-d_n]\right)/\sqrt{2}. \tag{4}$$

For the lowpass filtering, there are two current ways to get the forward motion vectors. One way in [7] used forward motion estimation, thereby requiring two motion fields needed to be transmitted. The other way, followed in [5] and [13] is used here,

$$\begin{aligned} L\left[m-\bar{d}_m, n-\bar{d}_n\right] = \tilde{H}\left[m-\bar{d}_m+d_m, n-\bar{d}_n+d_n\right] \\ + \sqrt{2}A\left[m-\bar{d}_m, n-\bar{d}_n\right]. \end{aligned} \tag{5}$$

At the decoder, by using L and H, we can do the same interpolation on H and reconstruct A exactly if there is no quantization error,

$$\begin{aligned} A\left[m-\bar{d}_m, n-\bar{d}_n\right] = \Big(L\left[m-\bar{d}_m, n-\bar{d}_n\right] \\ -\tilde{H}\left[m-\bar{d}_m+d_m, n-\bar{d}_n+d_n\right]\Big)/\sqrt{2}. \end{aligned} \tag{6}$$

After A is available, we can do the same interpolation on A as the encoder, and reconstruct B exactly as

$$B[m,n] = \sqrt{2}H[m,n] + \tilde{A}[m-d_m, n-d_n]. \tag{7}$$

So no matter how we interpolate those subpixels, if we interpolate pixels in the same way at the encoder and the decoder, we can realize perfect reconstruction. This should come as no surprise by now, since we have put the desired spatial interpolation into the guaranteed reconstructible lifting filters. In (6), we see L and H are still necessary for the reconstruction of A, and H still only contains the information of interpolated pixels in A. But this interpolated information is also available in L. So it is canceled out in (6). Thus the interpolation algorithm has no influence on the perfect reconstruction. Of course, there is still the question of which interpolation is best to use.

Unconnected pixels in B are processed like (4), and unconnected (unreferred) pixels in A are processed as in (3).

3.3.1 Sub-Pixel Interpolation

We saw that interpolated temporal highpass band $\tilde{H}$ was used to update A. Then we have a question: can we make $L\left[m-\bar{d}_m, n-\bar{d}_n\right]$ the same as (2)? To answer this question, we use A and B to represent L

$$\begin{aligned}L[m-\bar{d}_m,n-\bar{d}_n] &= \tilde{H}[m-\bar{d}_m+d_m,n-\bar{d}_n+d_n] \\ &\quad+\sqrt{2}A[m-\bar{d}_m,n-\bar{d}_n] \\ &=(\tilde{B}[m-\bar{d}_m+d_m,n-\bar{d}_n+d_n] \\ &\quad-\tilde{\tilde{A}}[m-\bar{d}_m,n-\bar{d}_n])/\sqrt{2} \\ &\quad+\sqrt{2}A[m-\bar{d}_m,n-\bar{d}_n] \\ &=\frac{1}{\sqrt{2}}(2A[m-\bar{d}_m,n-\bar{d}_n] \\ &\quad-\tilde{\tilde{A}}[m-\bar{d}_m,n-\bar{d}_n]) \\ &\quad+\frac{1}{\sqrt{2}}\tilde{B}[m-\bar{d}_m+d_m,n-\bar{d}_n+d_n]\end{aligned} \tag{8}$$

where $\tilde{\tilde{A}}[m-\bar{d}_m,n-\bar{d}_n]$ results from two interpolations: the first interpolation is using integer pixels in A to obtain interpolated subpixels, which as seen in (4) are stored in the integer positions of H; the second interpolation happens in (5) when we use integer pixels in H (subpixels of A in effect) to interpolate subpixels in H (integer pixels of A in effect). If we can achieve

$$A[m-\bar{d}_m,n-\bar{d}_n]=\tilde{\tilde{A}}[m-\bar{d}_m,n-\bar{d}_n],$$

then

$$\begin{aligned}L[m-\bar{d}_m,n-\bar{d}_n] &= \frac{1}{\sqrt{2}}(A[m-\bar{d}_m,n-\bar{d}_n] \\ &\quad+\tilde{B}[m-\bar{d}_m+d_m,n-\bar{d}_n+d_n]),\end{aligned} \tag{9}$$

which is the same as (2). Since the pixels in $\tilde{\tilde{A}}[m-\bar{d}_m,n-\bar{d}_n]$ undergo an interpolation from integer pixels to subpixels and then from subpixels back to integer pixels, the necessary condition is *sinc* function interpolation, and a constant motion vector within the support region of the interpolation filter. For different s at ¼, ½, and ¾, we can get three different interpolation filters

$$f(n+s)=\sum_m f(m)\frac{\sin\pi(n+s-m)}{\pi(n+s-m)}, 0<s<1. \tag{10}$$

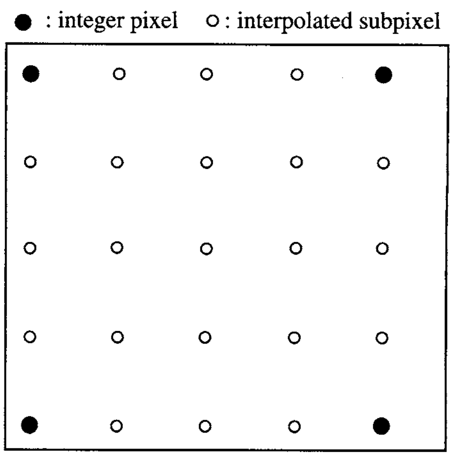


FIGURE 8 Subpixel interpolation.

The FIR interpolation can be designed as separable and using Hamming window. For relevant values of the shift s, our filter has the following coefficients:

$$\begin{aligned}s=\frac{1}{4}&:[-0.0110\quad 0.0452\quad -0.1437\quad 0.8950\quad 0.2777\quad -0.0812\quad 0.0233\quad -0.0053]\\ s=\frac{1}{2}&:[-0.0053\quad 0.0233\quad -0.0812\quad 0.2777\quad 0.8950\quad -0.1437\quad 0.0452\quad -0.0110]\\ s=\frac{3}{4}&:[-0.0105\quad 0.0465\quad -0.1525\quad 0.6165\quad 0.6165\quad -0.1525\quad 0.0465\quad -0.0105]\end{aligned}$$

With these filters, we can interpolate ¼, ½, and ¾ pixels. We first interpolate subpixel at integer columns and integer rows, where only 1D filtering of integer pixels is used. Then at each subpixel column, we will use those already generated subpixels to interpolate remaining subpixels (see Fig. 8). We can also implement by each subpixel row.

3.4 Comparison of Different Interpolation Filters

We use bilinear filter and our designed 8 tap FIR filters for ½ pixel accurate MC-EZBC. From the coding results on the *Mobile* test sequence as seen in Fig. 9, we can see that the 8-tap FIR filter gives better PSNR than bilinear filter.

3.5 Comparison of Integer-Pixel, 1/2 Pixel and 1/4 Pixel Accuracy

Figure 10 shows the rate-distortion curves of MC-EZBC on *Mobile* again with different motion accuracies. The eighth pixel accurate case is omitted, but it would lie just slightly above the ¼ pixel accurate case. We can see that the MCTF coding can benefit from quite an accurate motion

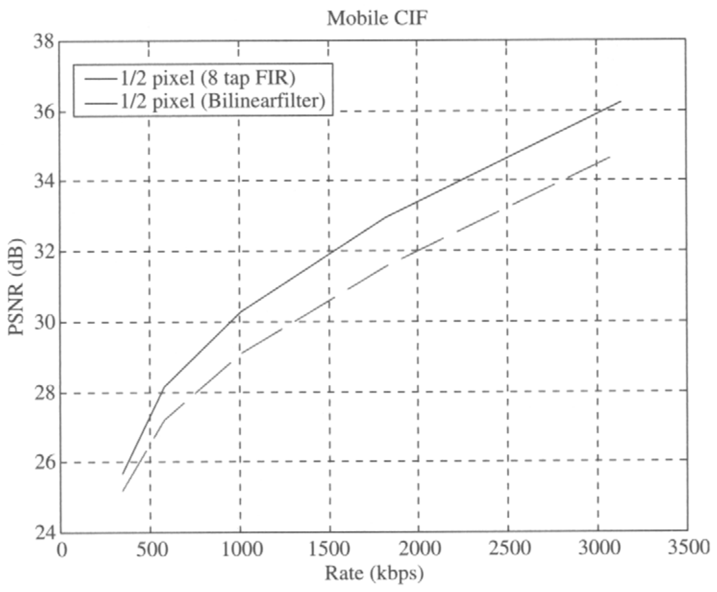


FIGURE 9 Comparison of 8-tap FIR filter versus bilinear filter.

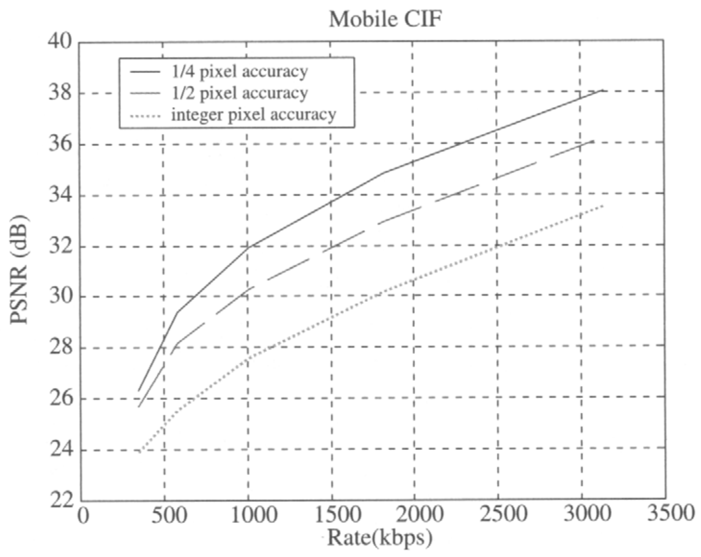


FIGURE 10 PSNR variation versus subpixel accuracy.

compensation, with a gain of 4 dB occurring at 2.5 Mbps for this CIF test sequence at 30 fps.

3.6 Bidirectional MCTF

We have already seen that because of the covered and uncovered pixels, for the temporal high frequency frame, we need to use frames on either side as an additional reference, no matter which frame we choose as reference. Ohm introduced the idea of bidirectional MCTF in [16], where the temporal high subband was in the position of the first frame, but the motion estimation direction was backward. The unreferred pixels in the first frame were then reasonably looked at as *to-be-covered* pixels, but the determination of the uncovered pixels in the second frame was done according to the scan order. So this scheme has the problem of "accidentally" classifying pixels as uncovered pixels. For the to-be-covered pixels, forward MCP was utilized. Then DFDs based upon the first frame's immediately preceding reconstructed frame were substituted. These to-be-covered pixels were assumed to have the same motion as their neighboring positions, termed the "homogeneous motion" assumption [16], so the

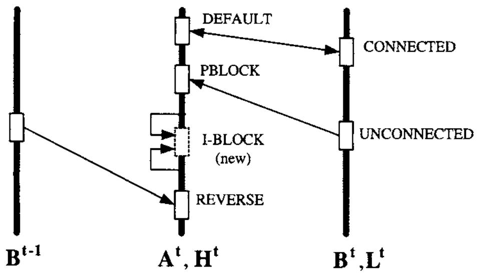


FIGURE 11 The block modes in high temporal frame for MC-EZBC. Frame H^t is temporally aligned with A^t, and L^t is temporally aligned with B^t. The IBLOCK employs spatial interpolation/prediction.

displacements at the adjacent connected positions were used. This might cause some problem, since there were different motions around the covered pixels.

3.7 Directional IBLOCKs

Since there are inevitable areas with occlusions and irregular motion, we classify the pruned blocks into three categories (or modes) in MC-EZBC as shown in Fig. 11: DEFAULT blocks with motion vector between current frame A^t and next frame B^t, where we do the lifting predict step in A^t and the update step for "connected" blocks in B^t. REVERSE blocks with motion vector between A^t and B^{t-1}, where we do motion compensation from a corresponding block in B^{t-1}. The PBLOCK has a motion vector between A^t and B^t, we do a predict step in A^t, and omit an update step for "unconnected" pixels in B^t. This PBLOCK originates from multiply connected pixels as well as from singly connected pixels with too large a prediction error, as determined by a threshold $\gamma = 0.5$ between the original block's variance and the motion-compensated block's MSE.

Since PBLOCKs are classified partially from DEFAULT blocks with a threshold $\gamma = 0.5$, they can include poorly matched blocks between frames A^t and B^t. We regard such blocks of size 8×8 or 4×4 as potential candidates for IBLOCKs. Similarly we also look for IBLOCK candidates from the set of REVERSE blocks. In our coder, we detect IBLOCKs from the candidates and do spatial interpolation/prediction for them [21]. An interesting alternative approach is presented in [37], where the authors use Lagrange optimization to jointly determine motion-vector block size and the mode classification. They only use VSBM with no resolution hierarchy for motion estimation though.

4 Overlapped Block Motion Compensation

In MC-EZBC, after IBLOCK detection, we apply overlapped block motion compensation (OBMC) to the variable size blocks as shown in Fig. 3. In our OBMC framework, we view data received by the decoder prior to the decoding of frame k as a source of information about the true prediction scheme finally employed. For simplicity, we limit the relevant information for each block to be the two horizontal, two vertical nearest neighbors and itself [22], and we assume stationarity of the image and block motion field.

We use a modified 2D bilinear window, whose performance is only a little worse than the iterated optimal window design (OWD) presented by Orchard and Sullivan in [23]. Each block size has its corresponding weighting window. Since a large block probably has a different motion vector from its small neighbor, a *shrinking* scheme is introduced between the different sized blocks, in order to reduce smoothing at a motion discontinuity. We do OBMC for all the types of blocks classified above, but we use a different weighting window for IBLOCKs to emphasize the continuity between motion compensation and spatial interpolation/prediction.

To further optimize OBMC, we employ an iterative procedure and minimize mean absolute difference (MAD). Since bi-directional color HVSBM runs on both luminance and chrominance data, it follows naturally that the OBMC iterations should be applied to *YUV* simultaneously. We know *U* and *V* are sub-sampled frame data after the color space transform from *RGB* data, so the weighting windows used for *U* and *V* are also sub-sampled versions of those used for *Y* [24].

4.1 Motion Compensated Temporal Filter with OBMC

As shown in Fig. 11, we do OBMC in a lifting implementation for DEFAULT blocks, i.e., with the prediction and update steps in normal order to reduce the noise in the area of good motion. The specific equations for OBMC in our lifting implementation are as follows [24],

$$H[m,n] = \frac{1}{\sqrt{2}}B[m,n] - \frac{1}{\sqrt{2}}\sum_k h_k[m,n]\tilde{A}[m - d_{mk}, n - d_{nk}], \tag{11}$$

$$\begin{aligned} L[m - \bar{d}_m, n - \bar{d}_n] = {} & \tilde{H}[m - \bar{d}_m + d_m, n - \bar{d}_n + d_n] \\ & + \sqrt{2}A[m - \bar{d}_m, n - \bar{d}_n]. \end{aligned} \tag{12}$$

The OBMC approach treats the displacement vector (d_m, d_n) as random, meaning that pixel $B[m,n]$ in frame B has motion vector (d_{mk}, d_{nk}) with probability $h_k[m,n]$ as determined by its corresponding probability weighting window. So the prediction is the weighted average of the self and nearest neighbor predictions. As above, in these equations $(\bar{d}_m, \bar{d}_n)$ is the nearest integer to (d_m, d_n). Although the form of the low temporal frame (12) seems the same as that without OBMC, actually OBMC affects both high and low temporal frames. To see this, note that in this lifting implementation, the H value in (11) is shifted and interpolated and then becomes the $\tilde{H}$ value in (12). Also, importantly, the low temporal frames from OBMC are visually preferred and more suitable for further stages of MCTF (i.e., much more nearly block-artifact free).

For PBLOCK, there is only a prediction step (1) from a future frame, and for REVERSE block, there is only a prediction step (1) from a previous frame. The IBLOCK is predicted with the weighted average of spatial prediction/interpolation from spatial neighbors in the corresponding frame.

5 Scalable Motion Vector Coding

After careful arrangement of the bitstream for motion vectors, namely grouping the motion vectors in each temporal layer with the frame data in that temporal level, temporal scalability of motion vectors follows naturally. But the bitstream for motion vectors was still not scalable with respect to SNR and resolution. Here we describe a scalable motion vector coder based on context-based adaptive binary arithmetic coder (CABAC) [15] for MC-EZBC. A layered structure for motion vector coding and alphabet general partition (AGP) of motion vector symbols are employed for SNR and resolution scalability of the motion vector bitstream [25]. With these two features and the careful arrangement of the motion vector bitstream in MC-EZBC, we can have elementary forms of temporal, SNR and resolution scalability for motion vectors, and improve significantly both visual and objective results for low rates and low resolution with slight PSNR loss (about 0.05 dB) and unnoticeable visual loss at high rates. Initial results on scalable motion vector coding for MC-EZBC were obtained in [27].

5.1 Scan/Spatial Prediction for Motion Vectors

As shown in Fig.12, now the motion vector *mv* in the current block is predicted from its three nearest previously processed spatial neighbors *mv*1, *mv*2, and *mv*3 similar to the approach in [8]. The spatial prediction scheme can predict the motion vector more efficiently as shown in our previous work. The motion vectors in DEFAULT and PBLOCK are between current and next frames (defined as normal motion vectors), those in REVERSE blocks are between current and previous frames (defined as reverse motion vectors). There are no motion vectors in IBLOCKs, as they use spatial prediction/interpolation only.

We found experimentally that the characteristics of normal motion (forward) vectors and reverse ones are quite different. So we predict and code the two sets of motion vectors separately, which can often improve the prediction. Then there are two loops for motion vector coding. The first loop is for normal motion vector coding, and the second loop is for reverse motion vector coding.

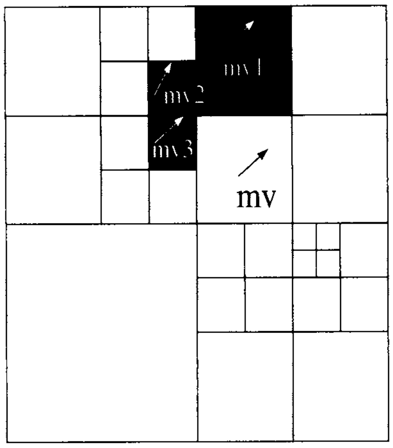


FIGURE 12 Spatial prediction from three direct neighbors for the motion vector in some block.

For the motion vector *mv* in a target block, if there is at least one motion vector in *mv*1, *mv*2, and *mv*3 with the same type (normal or reverse) as that in this block, we predict from the spatial neighbors. If there is no motion vector with the same type in these three neighbors, we predict it from the previous motion vector with the same type in quad-tree scan order. We consistently use this combined spatial and scan-order prediction, and the prediction residual is coded by CABAC [15].

5.2 Alphabet General Partition of Motion Vector Symbols

In MC-EZBC, motion estimation is typically done at 1/8 pixel accuracy. Although it can reduce MSE after motion compensation, the quarter and eighth pixel bits of motion vectors are quite random due in part to the camera noise and quantization noise. This is because the MSE after motion compensation is already near the total noise variance after quarter or eighth pixel accuracy motion compensation. We can thus model the motion vector as follows (without loss of generality we present the one-dimensional case although a motion vector has two components)

$$r_k = s_k + n_k$$

Here r_k the estimated k^{th} motion vector, s_k is the true k^{th} motion vector, and n^k is the noisy motion vector due to the inaccuracies in the frame data. All of the three components are at 1/8 pixel accuracy. Since the noises in frame data are quite small, we believe they can only contaminate the quarter pixel and eighth pixel accuracy of the estimated motion vector during motion estimation. Here we divide the estimated motion vector r_k into two symbols,

$$r_k = r_{k1} + r_{k2}$$

where r_{k1} is the major symbol at 1/2 pixel accuracy, and r_{k2} is the sub-symbol for quarter and eighth pixel accuracy. For example, if the estimated motion vector $r_k = -1.625$, then $r_{k1} = -1.5$ and $r_{k2} = -0.125$. We predict the major symbol with the above scheme in section and code the prediction residual with CABAC, but code the sub-symbol simply as a binary number.

Actually we do not need to code the sign for sub-symbols because we can find the sign from the major symbol. The only exception is for those motion vectors in the range $[-0.375, +0.375]$, for which the major symbol $r_{k1} = 0$. Then we don't know the sign for sub-symbol r_{k2}, so we need one additional bit to indicate whether the sub-symbol is positive or negative, in this case.

At high bit rates and full resolution, we transmit all the three parts of the motion-vector bitstream. At the decoder we receive the lossless motion vectors to reconstruct the frames. But at low bit rates, we can discard the sub-symbol and sign parts, and get more room for frame (SWT) data. Although the motion vectors are lossy, this can be compensated by the increased accuracy of the frame data, and thus, the total performance can be improved. Also at low resolution, since in MC-EZBC the motion vectors will be scaled down, obviously we do not need the same accuracy for motion vectors as that in full resolution. So the sub-symbol and sign parts can also be discarded, making more room for the SWT coefficients/data.

One can develop an experimental R-D type model for scaling the motion vectors. We also plan to partition the motion vector symbols further, i.e., not only partition quarter and eighth accuracy but also half or integer accuracy. Also, a context model can be introduced for the sub-symbols instead as a substitute for simple binary coding. This may be necessary in fact if more levels of scalability for motion vector accuracy are needed.

5.3 Layered Structure of Motion Vector Coding

After one spatial resolution level down, the block size is halved, i.e., 16×16, 8×8 and 4×4 blocks become 8×8, 4×4 and 2×2 blocks, respectively. Thus, after one or two spatial levels down, we do not need the same number of motion vectors as at full resolution. Moreover, the motion vectors are also scaled down by a factor of 2 after each spatial level downwards, since if two adjacent motion vectors differ by less than 2 pixels at full resolution, then at half resolution they will differ by less than 1 pixel.

Assume four grouped blocks (children) in the quad-tree structure have similar motion vectors. We can replace the four motion vectors by one representative at one spatial level down. Currently we just choose the first motion vector in quad-tree scan order as the representative for the four motion vectors in the children.

As mentioned above, the normal and the reverse motion vectors can have different characteristics. So we reserve up to two representatives for the four children, one for normal motion vectors and the other for reverse motion vectors in the four child blocks.

We use a sub-sample selection scheme from bottom-up to form the layered structure for motion vector coding. In each layer, we use the scan/spatial prediction as stated above. We also continue using the context model from layer to layer. Since we simply choose the first normal and reverse motion vector in quad-tree scan as the representative, the coded number of motion vectors is the same as in non-layered coding. Then we code the prediction residuals in larger blocks in the base layer of the motion vectors, with successively smaller blocks as the enhancement layers. When coding enhancement layers, one should use all the information up to that layer, i.e., updated motion field and updated context models from previous layers.

This coarse layer structure for motion vector coding shows good results [25]. As an alternative, the local gradient of the compensated frame data has been used with success by Secker and Taubman [38]. If an image frame area is very smooth, then the difference among the motion vectors will not affect the reconstructed frame's distortion very much. However, if the area is full of texture, then a little difference among the motion vectors may cause a large distortion. In [38] a local average of the energy in the gradient is used to modulate the portion of the total bit assignment going to the motion vectors, determined over a triangular mesh, which are also coded with a lossless SWT coder.

6 The EZBC Coder

After MCTF analysis with OBMC, a set of high temporal frames and typically one low temporal frame as shown in Fig. 2 will be input to EZBC coder. The entropy coder EZBC then codes the frame data with scalability respect to SNR, temporal and resolution, by interleaving the bitstreams of the spatial and temporal subbands. Please refer to [1] for the details about the EZBC coder for fine scalable video compression. Unlike JPEG2000 and EBCOT [29] on which it was based, there is no rate-distortion optimized bit assignment over code blocks, as the entire subbands are coded together. This results in a complexity advantage for EZBC.

6.1 Definitions

- $c(i, j, t)$: integer part of subband/wavelet coefficients at position (i, j, t) after proper scaling
- $\mathrm{QT}_k[l](i, j, t)$: quadtree representation of SWT coefficients for subband k, at quadtree level l, and spatiotemporal position (i, j, t)
  - $\mathrm{QT}_k[0](i, j, t) \triangleq |c_k(i, j, t)|$

- $QT_k[l](i,j,t) \triangleq \max\{QT_k[l-1](2i,2j,t), QT_k[l-1](2i,2j+1,t), QT_k[l-1](2i+1,2j,t), QT_k[l-1](2i+1,2j+1,t)\}$

- $m(i,j,t)$: MSB (most significant bit) of quadtree node (i,j,t)
- D_k: depth of the quadtree of subband k
- QTR_k: collection of all quadtree roots of subband k
- $D_{\max}$: $\max_k\{D_k\}$
- K: total number of subbands
- n: index of the current bitplane pass, corresponding to quantization threshold 2^n
- $S_n(i,j,t)$: significance of quadtree node (i,j,t) against threshold 2^n,

$$S_n(i,j,t) \triangleq \left\{\begin{array}{ll} 1, & n \le m(i,j,t), \\ 0, & \text{otherwise.} \end{array}\right\}$$

Node (or pixel or coeff.) is significant iff $S_n(i,j,t) = 1$.
- $LIN_k[l]$: list of insignificant nodes from quadtree level l of subband k
- LSP_k: list of significant pixels from subband k
- $CodeLIN(k,l)$: function for processing insignificant nodes in $LIN_k[l]$
- $CodeLSP(k)$: function for coefficient refinement
- $CodeDescendentNodes(k,l,i,j,t)$: function for coding significance of all descendent nodes of $QT_k[l]$ (i,j,t), which just tested significant against the current threshold.

1. Coder

$$LIN_k[l] = \left\{\begin{array}{ll} QTR_k, & l = D_k \\ \phi, & \text{otherwise,} \end{array}\right\}$$

$$LSP_k = \phi,$$

$$n = \left\lfloor \log_2\left(\max_{(k,i,j,t)}\{|c_k(i,j,t)|\}\right)\right\rfloor.$$

```
step 1: for $l = 0 : D_{\max}$
            for $k = 0 : K-1$
                codeLIN$(k,l)$
            end
            for $k = 0 : K-1$
                codeLSP$(k)$
            end
        end
```

$n \to n-1$ and return to step 1, stopping when target bitrate is reached.

```
2. CodeLIN$(k,l)$ {
     for each $(i,j,t)$ in $LIN_k[l]$
       code $S_n(i,j,t)$
       if ($S_n(i,j,t) = 0$, $(i,j,t)$ remains in $LIN_k[l]$
       else
         if($l = 0$), then output sign bit of $c_k(i,j,t)$ and add
           $(i,j,t,l)$ to $LSP_k$
         else CodeDescendentNodes$(i,j,t,l)$
   }

3. CodeLSP {
     for each pixel $(i,j,t)$ in $LSP_k$, code bit $n$ of $|c_k(i,j,t)|$
     }

4. CodeDescendentNodes ($k$, $l$, $i$, $j$, $t$) {
      for each node $(x,y,t)$ in $\{(2i,2j,t),(2i,2j+1,t),$
        $(2i+1,2j,t),\ (2i+1,2j+1,t)\}$ of quadtree
     level $l-1$ and subband $k$
       output $S_n(x,y,t)$
       if ($S_n(x,y,t) = 0$), add $(x,y,t,l-1)$ to $LIN_k[l-1]$
       else
         if($l = 1$) output the sign bit of $c_k(x,y,t)$ and add
           $(x,y,t)$ to $LSP_k$
         else CodeDescendentNodes$(k, l-1, x, y, t)$
   }
```

EZBC begins with establishment of the quadtree representations of individual subbands. The value of each quad-tree node is just equal to the maximum magnitude of the SWT coefficients in the block region corresponding to this node. In contrast with the conventional pixel-wise bitplane coding algorithm, EZBC also needs to deal with bitplanes of nodes at individual quadtree levels. Nevertheless, only nodes in lists LIN and LSP need to be processed in each bitplane coding pass.

The coding procedure in EZBC is similar to that of SPECK [28] which also adopted the same quadtree splitting method. The major difference lies in context modeling schemes for coding decisions of quad tree splitting, which will be described in the next section. The other important difference is that the lists in EZBC are all separately established for different quadtree levels and subbands. Therefore, all nodes in a list come from the same quadtree level and subband. With context models also separately built for the individual lists, different statistical characteristics of the individual quadtree levels and subbands can be more accurately modeled. The other added benefit of this modification is the algorithm can be applied for resolution-scalable coding applications once a proper bitstream parsing method is performed.

6.2 Context Modeling

To code the significance of the quadtree nodes, we include 8 neighbor nodes of the *same* quadtree level in the spatial context, as illustrated in Fig. 13. This spatial context has

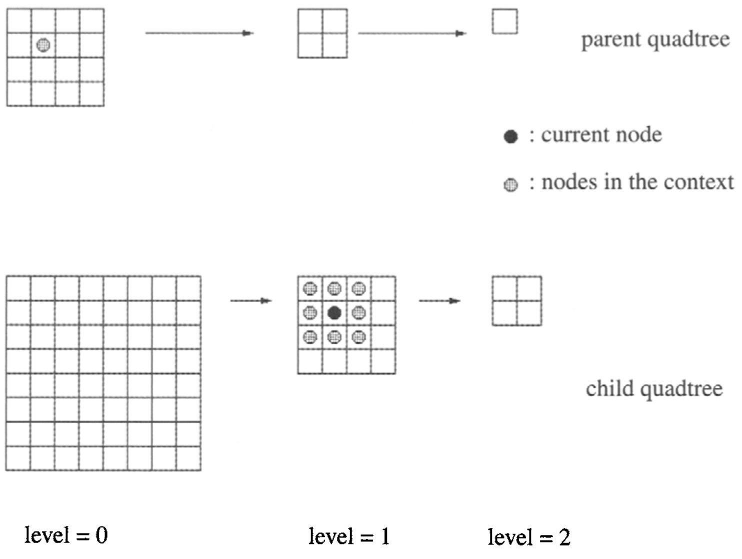


FIGURE 13 Illustration of the context models of a quadtree node.

been widely adopted for coding of the significance of *SWT* samples/coefficients. However, the information across scale is given by the node of the parent subband at the *next lower* quadtree level, as shown in Fig. 13. This choice is based on the fact that at the same quadtree level the dimension of the region a node corresponds to in the input image is halved in the parent subband as a result of subsampling at the transform stage. The model selection of the arithmetic coding is just based on a 9-bit string with each bit indicating the significant status of nodes in this context. To lower model cost, instead of including all context states (2^9 totally), we adopted a similar method to EBCOT [29] for context reduction. The sign coding scheme of EBCOT is also employed in our algorithm.

Coding comparisons of EZBC versus SPIHT, SPECK, and JPEG2000 are contained in [19] and show a general slight advantage in PSNR for EZBC in the coding of natural image data. This is surprising since EZBC omits the rate-distortion optimization of JPEG2000, and simply interleaves the bitstream passes from the various subbands. This amounts to relying on the near orthogonality of the transform to produce a near constant MSE in the data. An additional simplification with JPEG2000 is that EZBC revisits the pixels less often, amounting to about a 50% savings for bit rates less than about 1 bit per pixel as an image coder [26].

6.3 Scalability

The layout of the bitstream for a GOP is shown in Fig. 14(a). The 3D SWT coefficients are coded from the most significant bit (MSB). The quantizer step size is halved after each bit coding pass. The quality (or the quantizer step size) of the video can be controlled by just dropping the corresponding subbitstreams for the remaining bits of subband coefficients. The bitstream is therefore scalable in SNR. Since the bitstream

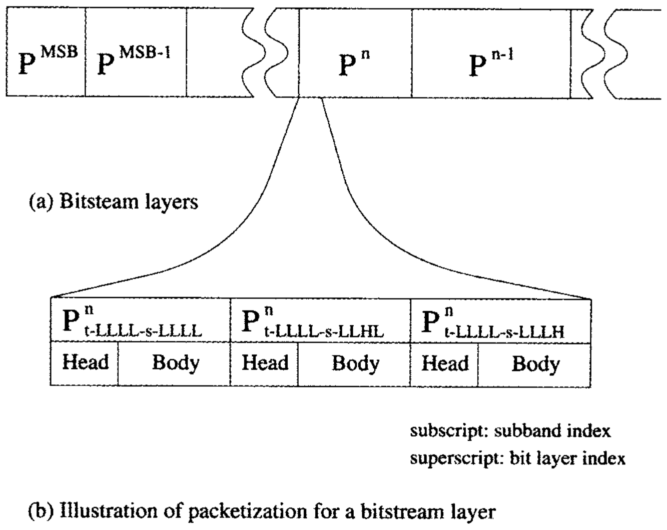

P^{MSB}
P^{MSB-1}
P^{n}
P^{n-1}
(a) Bitsteam layers
$P^{n}_{t\text{-}LLLL\text{-}s\text{-}LLLL}$
$P^{n}_{t\text{-}LLLL\text{-}s\text{-}LLHL}$
$P^{n}_{t\text{-}LLLL\text{-}s\text{-}LLLH}$
Head
Body
Head
Body
Head
Body
subscript: subband index
superscript: bit layer index
(b) Illustration of packetization for a bitstream layer

FIGURE 14 The coded bitstream for a GOP in EZBC.

itself is embedded, it can be truncated at any point to meet desired video quality or rate constraint. Hence, it is also scalable in coding rate.

In MC-EZBC, context models and lists are separately established for individual subbands. Any subband can be separately decoded from its corresponding subbitstream once its *dependent subband* has been processed up to the current bit coding pass. This restriction in the decoding order is due to the fact that inter-subband models are employed for context-based arithmetic coding. The resolution and frame-rate scalabilities can be achieved by discarding the subbitstreams for irrelevant spatial/temporal subbands. For examples, the video can be reconstructed at half resolution and at half frame rate by throwing away subbitstreams for s-LH, s-HL, s-HH, and t-H subbands. For ease of extraction of subbitstreams without transcoding, the subbitstreams for individual subbands are packetized as illustrated in Fig. 14(b). The actual decoding/transmission order for individual subbands is not precise. We may, for instance, transmit packets for all spatial low-frequency subbands first. Again, the principle is that each subband has to be decoded after its dependent subband.

For multiresolution coding, we adopt a single prediction loop to reduce the temporal redundancy. Slight quality degradation from energy leakage may occur due to the fact that the SWT is not shift-invariant. In contrast to the multiple-loop scheme by [30], this choice is made with the goal to optimize the coding performance for the full-resolution video.

6.4 Packetization

As the last step of MC-EZBC, packetization realizes quality control. In our coding system with GOP structure, we try to realize constant quality both inside a GOP and across the many GOPs of the video clip. Since we use a near orthonormal transformation both in the spatial and temporal domains, corresponding bit planes of different subbands have the same importance. In bit plane scanning, we proposed to interleave spatial subbands of all temporal subband frames of a GOP and interleave their coding passes further [10] to realize constant quality inside a GOP.

To realize constant quality across GOPs, we proposed a two-step coding scheme for the long-term constant quality problem [10, 13]. In this scheme, we first encode the entire sequence and stop the bit plane encoding at some bit plane, which will ensure the needed quality. Effectively, we have just created an archive. Our strategy of bit allocation over GOPs is to make the bit plane coding of all the GOPs stop at the same bit plane and realize approximately constant quality. While this type of constant quality does not optimize the total average performance, it comes close, and may well provide the visually better performance.

7 Comparing Scalable Coders—Cross-Check Method

Since MCTF coders, and in fact all scalable coders, produce the low resolution and low frame rate video, they in effect create their own *references*.[2] Since each scalable coder is expected to generate different references, there is the problem of doing an objective (PSNR say) comparison. This is a new problem that arises in comparing scalable coders. It has not arisen in the past since the references were always generated in the same way, using for example, standard downsampling filters and frame skipping.

In the cross-check as shown in Fig. 15, we compare two scalable coders not only on their own reference, but also on the other coder's reference. If one of the coders performs best on both references, then it is said to be the better coder. While "coding" the natural reference (i.e., the scalable coders own reference), we assume that the coding does not have to be explicitly done, only the partial decoding or so-called pull function. While coding the other coder's reference though, explicit coding must be performed. On that reference, the coder is not scalable (most likely). If the scalable coder with the best performance also obtains that performance on its reference, then we say it has won the test as a scalable coder. Otherwise, it could be best on the other coder's reference, and be a better coder, but not a better scalable coder. Neither coder may win the test, and that is perhaps the main difficulty with the cross-check method! However, this tie outcome has not occurred as yet in our limited experiments with four common test clips, two of which are reported below.

When we cross-check a scalable coder with a nonscalable one, the nonscalable one can win the test, but *not as a scalable coder*. If the scalable coder only ties with the nonscalable one, it has effectively won as a scalable coder. Below we use the cross-check method with the nonscalable coder H.26L.

Of course, there is always the question of the quality of the reference videos. We assume that this is judged elsewhere and is not a part of the cross-check test. Nevertheless, it is very important that both references have passed subjective tests as to their suitability for use. Otherwise, there is not much sense in doing the cross-check in the first place. To ensure this in the experimental comparisons below, we used the low value of $\lambda_{mv} = 6$ in the MCTF output rather than $\lambda_{mv} = 24$ for the Lagrange multiplier in HVSBM tree pruning in MC-EZBC [17]. At $\lambda_{mv} = 6$ we did not see any artifacts in the lower frame-rate videos. Since decreasing λ_{mv} will cost more motion bits, at low bitrates and full frame rate we observe a performance drop.

Note that we never compute PSNR with respect to the reference that was not coded in that trial. Such a comparison would not make sense in our opinion. For each temporal level we get four results, by varying the two coders and the two references, but always using the chosen reference for the PSNR calculation. The following general comments can be made. The MCTF low frame-rate reference will probably have less temporal aliasing and less sensor noise, possibly making it easier to code, but occasionally there will be some motion artifacts. On the other hand, the conventional frame-skipped reference will have more temporal aliasing along with the full amount of camera or sensor noise, but no motion artifacts. It may be harder to code.

[2]A reference is the "original" to be used for calculation of PSNR at these lower resolutions and frame rates. It amounts to the coder output at infinite bit rate, for these resolutions and frame rates.

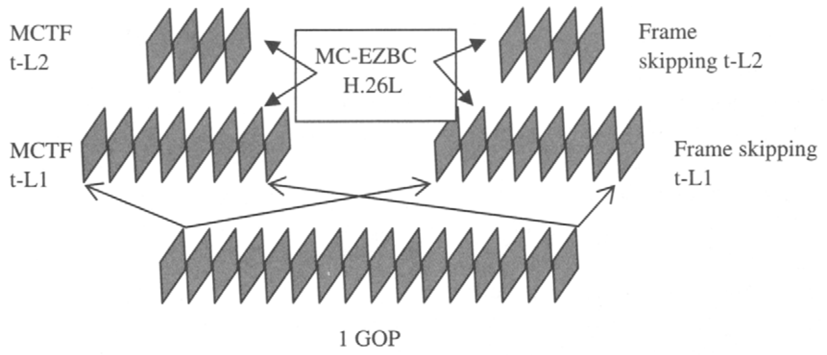


FIGURE 15 Cross-check method.

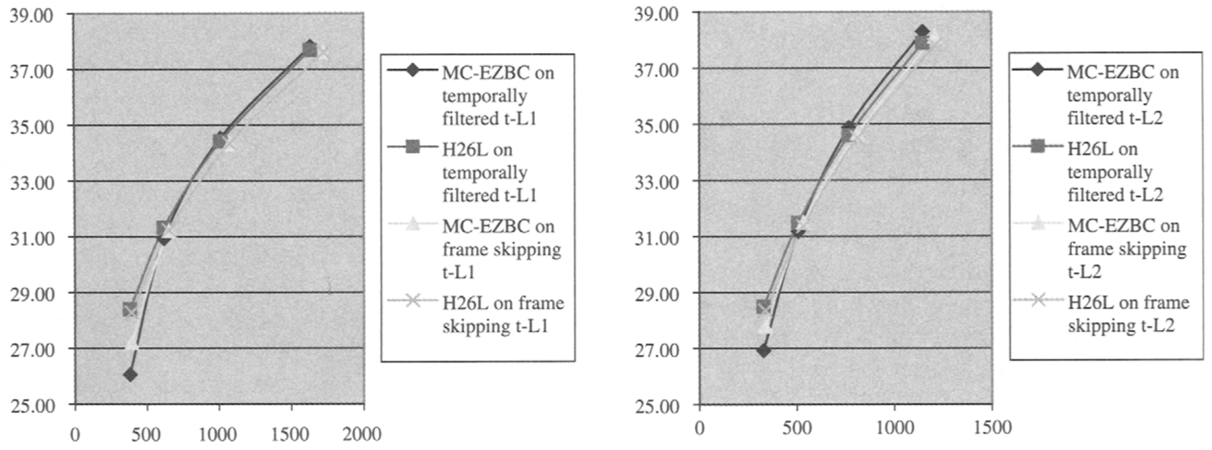


FIGURE 16 Comparison of MC-EZBC and H.26L on decimated and MCTF sequences for *Mobile*.

7.1 Cross-Check Experiments

We compare MC-EZBC to H.26L on lower frame-rate videos at 15 Hz (t-L1) and 7.5 Hz (t-L2). We use $\lambda_{mv} = 6$ for the MCTF. We treat H.26L as not scalable, even in frame rate. So all various frame rates are encoded by this coder directly. We extract lower frame-rate videos at 15Hz and 7.5 Hz from a MC-EZBC bitstream at high bitrates and use them as the references for both MC-EZBC and H.26L. Then we encode these two lower frame-rate videos with H.26L at different QP levels. We also use MC-EZBC and H.26L to encode lower frame-rate videos generated by frame skipping. When we run MC-EZBC on frame skipping t-L1 and t-L2 videos, we use GOP size of 8 and 4, respectively, to match the lower frame rate GOP sizes chosen for H.26L. The PSNR results are shown, four to a plot, in Figs. 16 and 17. In these figures the left plot is for 1/2 frame rate and the right plot is for 1/4 frame rate.

We can see scalable MC-EZBC is comparable to nonscalable H.26L at medium and high bitrates in lower frame-rate video *Mobile*, but not for *Foreman*. These results at lower frame rates are consistent with those of the full frame rate. For *Bus* and *Foreman*, in both MCTF low frame-rate videos and frame-skipping videos, H.26L has the higher PSNR performance. Two other clips tested, but not with PSNR shown here, were *Flower Garden*, *Bus*, and *Coastguard*.

Interestingly, we find the PSNR performance for the MCTF low frame-rate reference is never lower than that of the frame-skipping reference, and almost always significantly higher. We conclude that MCTF output is actually easier to code than the decimated or frame-skipped sequences. Now the H.26L coder is scalable in frame rate but not in quality like MC-EZBC. Based on tests across the five video clips, it turned out that, if the test were for frame-rate scalability only, then H.26L wins the test for two out of these five test videos, with MC-EZBC winning the test on the other three. Interestingly though, H.26L could have even better performance on the MCTF reference outputs. Conversely, if the test is for both frame-rate and quality scalability, then there is no way that H.26L wins the test, because it is not eligible. An FGS extension of H.26L or of H.264 could be used for that test though.

For full frame-rate sequences where MC-EZBC was either better or worse than H.26L, the same ranking was observed at lower frame-rate videos either coming from MCTF or frame

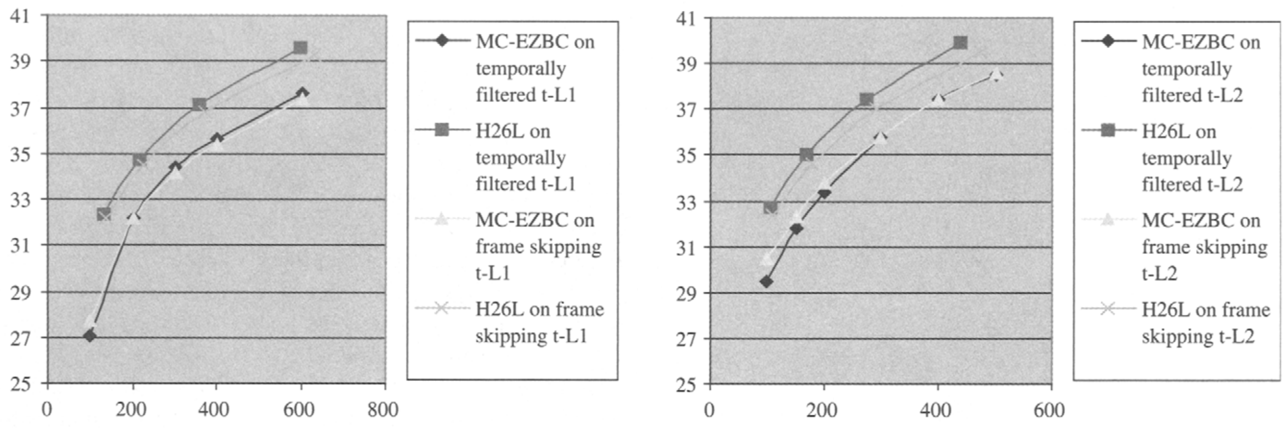


FIGURE 17 Comparison of MC-EZBC and H.26L on decimated and MCTF sequences for *Foreman*.

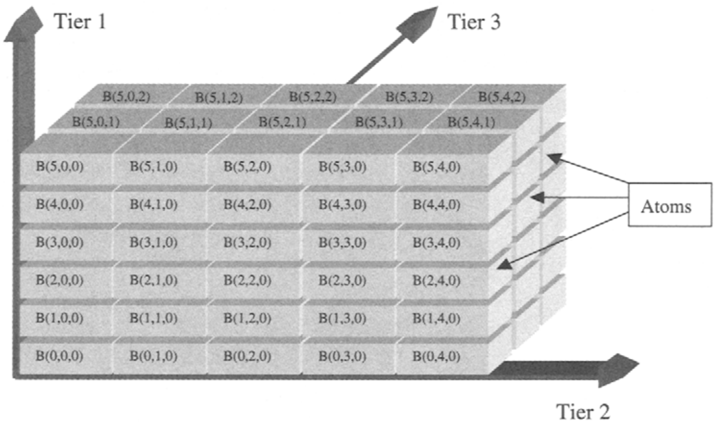

Tier 1
Tier 3
Tier 2
Atoms
B(5,0,2) B(5,1,2) B(5,2,2) B(5,3,2) B(5,4,2)
B(5,0,1) B(5,1,1) B(5,2,1) B(5,3,1) B(5,4,1)
B(5,0,0) B(5,1,0) B(5,2,0) B(5,3,0) B(5,4,0)
B(4,0,0) B(4,1,0) B(4,2,0) B(4,3,0) B(4,4,0)
B(3,0,0) B(3,1,0) B(3,2,0) B(3,3,0) B(3,4,0)
B(2,0,0) B(2,1,0) B(2,2,0) B(2,3,0) B(2,4,0)
B(1,0,0) B(1,1,0) B(1,2,0) B(1,3,0) B(1,4,0)
B(0,0,0) B(0,1,0) B(0,2,0) B(0,3,0) B(0,4,0)

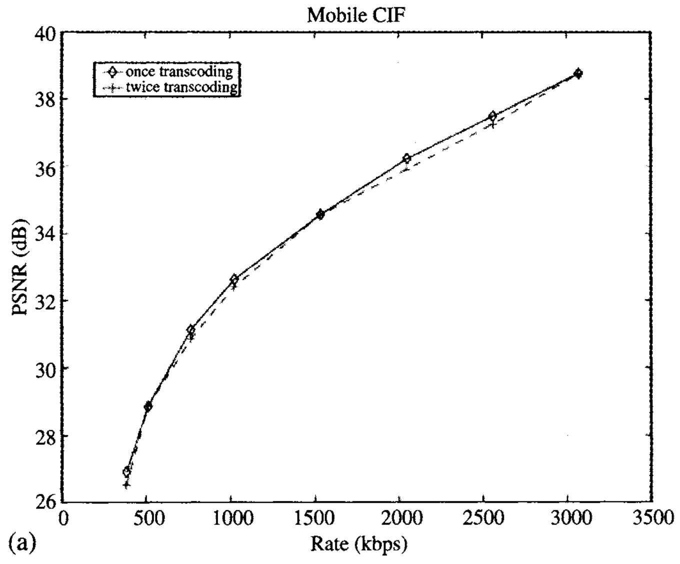
FIGURE 18 3D scalable data strucure [31].

skipping. We also found MCTF output was easier to code than the decimated sequences. This may give MCTF referenced coders an edge.

8 Multiple Adaptations

So far we have presented scalable coding as a method for compressing the video once at the source and placing it onto a video server, from which various qualities, bit rates, spatial resolutions, and frame rates can be extracted. The individual spatiotemporal subbands are stored in embedded form, and the context modeling is based on lower resolution data. A useful abstraction of the packetized data is the 3D data structure shown in Fig. 18 (borrowed from [31]), where each packet is termed an *atom*. The three dimensions are called tiers, and in our case would correspond to bitrate, resolution, and frame rate. So this is the data structure present at the video server, but in this figure, quantized down to a small number of atoms for display purposes. Because of the context dependence of the arithmetic coding, there is a dependency here, with each atom depending on its lower resolution atoms. We can scan the cube in various directions, always in increasing order in each dimension to satisfy the context dependencies, to perform the required scalabilities, as also indicated with a different notation in Section 6.3. This is the so-called "pull function" of scalable coders, which creates the actual (one-dimensional) bitstream to be sent out over the channel or network link.

In a network application though, we may have to *adapt* the data further inside the network, where by adapt we mean reduce one or more of the dimensions: bitrate, spatial resolution, and/or frame rate. Adaptation may be required multiple times inside a network, due to the use of multicast and broadcast to serve a number of users, with their heterogeneous set of displays and compute capability. Another motivation is the need to respond to congestion and packet loss inside the network. The real practical case would combine these two. For a non-scalable coder, one would have to adapt the data by employing transcoding, with its attendant computational



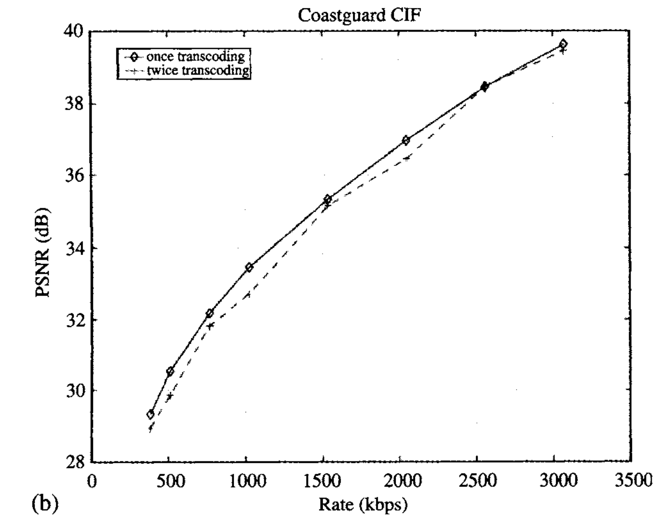


FIGURE 19 (a) PSNR performance in multiple transcoding *Mobile*. (b) PSNR performance in multiple transcoding *Coastguard*.

load and quality loss. A transcoding solution would also require the network node processors to understand the coding algorithm used. In contrast, a scalable coder merely sheds unneeded atoms, making data adaptation much easier. In order to do this, some additional small amount of information has to be packetized and transmitted with the coded data, to indicate the locations in the bitstream where the various spatiotemporal subbands are located (Fig. 14(b)). Such adaptations would not be performed in the network itself, but on a content delivery network (CDN) superimposed on the underlying network.

Below we give some illustrative results using MC-EZBC with a $5 \times 6 \times 7$ scalability structure, meaning 5 temporal or frame-rate layers, six spatial resolution layers, and seven bitrate or quality layers, all rate controlled on a GOP basis. Each GOP will then have a header that contains information as to how many temporal and spatial layers are included. Further, each spatiotemporal layer is proceeded by a length field. When an adaptation is made on a GOP, not only should appropriate parts of the bitstream be removed, but the information in the header and length fields must be updated appropriately to enable correct decoding. Since we downsize the description file to seven quality layers to save on bits for the resource description file, at the following adaptations we cannot access the bitstreams at the fractional bitplane level. Figure 19a shows the PSNR performance after a second adaptation (transcoding) for the *Mobile* clip. We see the dashed curve (a second adaptation) is nearly coincident with the solid curve (first adaptation). The data in these curves is at the indicated rate points, which are $7+1$ in number. Figure 19b shows a similar performance for the *coastguard* clip, where slightly more loss is evident but mostly under 0.5 dB. For any fully embedded coder, the loss at the indicated rate points is solely due to the overhead of carrying the additional header information needed for the multiple adaptations. Visually there is no apparent degradation of either clip after two adaptations. Further adaptations should suffer no further loss, since the header information is the same. In fact further adaptations would truncate the resource description file, deleting unneeded parts, and of course, reducing the slight overhead it causes.

9 Some Visual Results

Here, we show some visual results for MC-EZBC coding of the *Foreman* clip in CIF resolution at frame rates of 30 fps and 15 fps.

Visual improvement can be significant, as seen in the Fig. 20a and b, which show comparisons for full frame rate, with OBMC (20a) and without OBMC (20b). We see a clear decrease in blocking. The results after one adaptation to half frame rate are shown in Figs. 21a–b.

10 Other Improvements and Related Coders

In [32] two proposals are presented for further improvement of this interframe codec. They involve spatial transition filtering at block edges where there is a motion mode change and an approach to optimize the quantization error difference between DEFAULT, prediction, and intra blocks. These non-default blocks will have a higher quantization error on decoding from the already decoded DEFAULT blocks. There is also the possibility of applying a reconstruction or de-blocking filter in the decoder.

The MC-EZBC coder performs the temporal transform first followed by the spatial transform and is denoted $t+2D$.

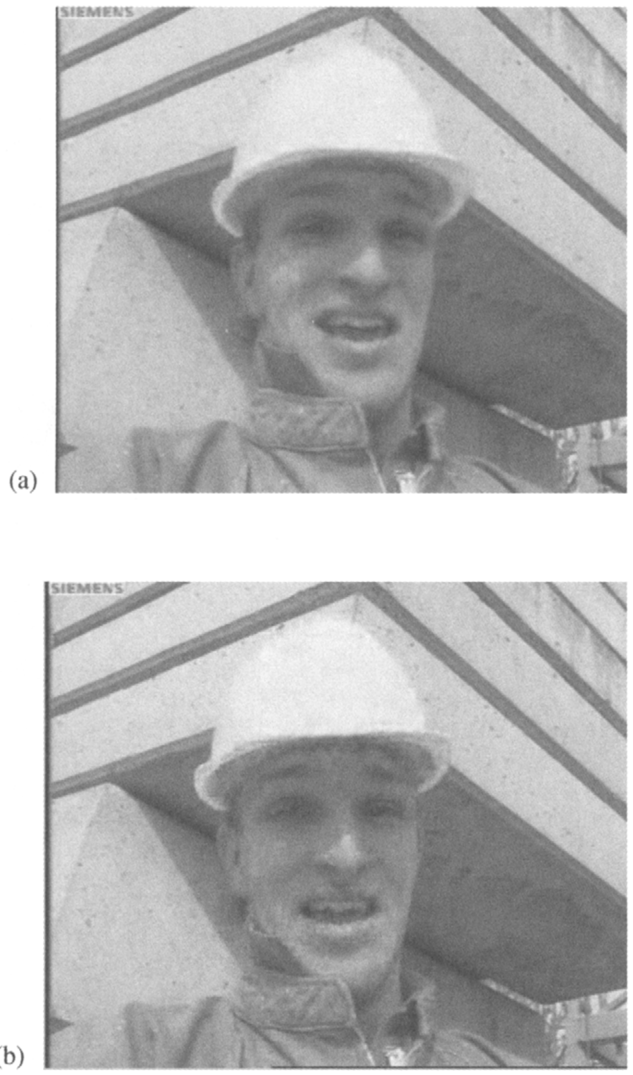


FIGURE 20 (a) Frame 141 in *Foreman* OBMC, IBLOCK and color HVSBM at rate 512 Kbps @ 30 fps. (b) Frame 141 in *Foreman* OBMC and IBLOCK at rate 512 Kbps @ 30 fps (see color insert).

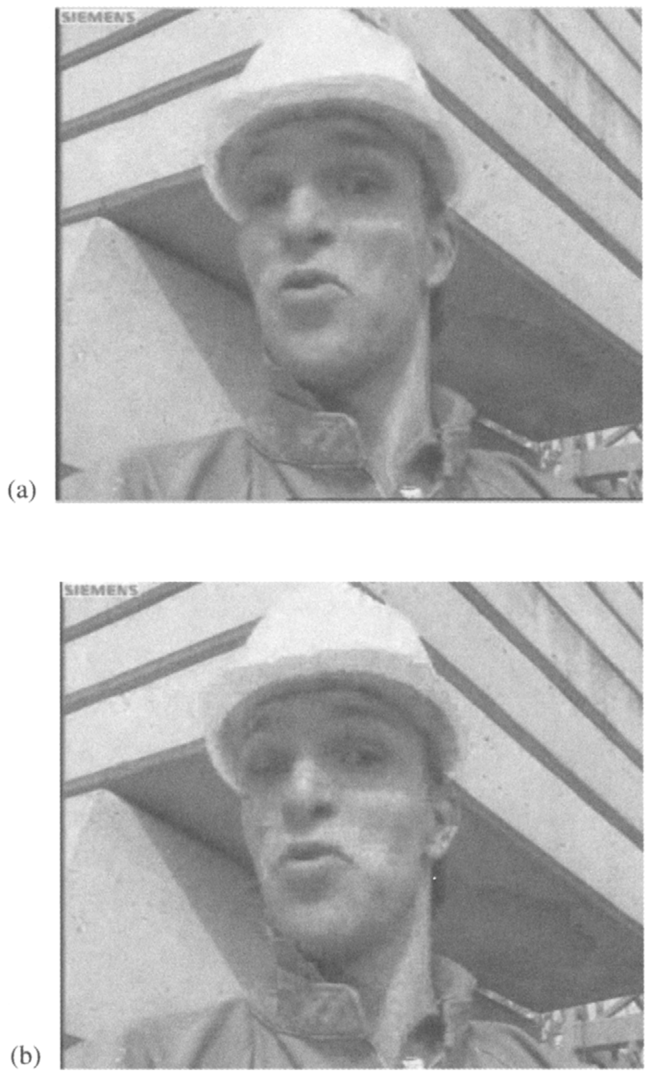


Figure 21 (a) Frame 43 in *Foreman* OBMC, IBLOCK and color HVSBM (corresponding to original frame 86) at rate 256 Kbps @ 15 fps, half the original frame rate. (b) Frame 43 in *Foreman* OBMC and IBLOCK, but WITH old YUV HVSBM (corresponding to original frame 86) at rate 256 Kbps @ 15 fps, half the original frame rate (see color insert).

Another class of interframe SWT coders, denoted 2D + t, performs the spatial transform first, followed by motion compensation of the subbands, often involving a complete to overcomplete SWT. A good example coder form this category is [33], whose performance is reportedly similar to that of MC-EZBC. Another t + 2D coder with performance similar to MC-EZBC is contained in [34]. A full presentation of these and other interframe/wavelet coders is contained in the recent special issue [35].

11 Conclusions

In this chapter we have considered highly scalable video coders producing embedded bit streams that can be adapted to produce reduced bit rates, resolutions, and frame rates. Of course, the trick is to do this with very little loss in performance over that of a state-of-art coder at each comparison point. The classic hybrid coders are not capable of doing this well, due to the coder state implied by their internal temporal DPCM loop. The newer class of MCTF coders have an inherent advantage in this respect, and compare very well to the best non-scalable coders. Many important design issues were reviewed in the context of the popular MC-EZBC video coder.

Finally we should mention a possibility of using the MCTF in such coders to preprocess the data prior to encoding, as well as postprocess the data at decoding. Due to the high accuracy of the MCTF motion compensation, such filtering can be very beneficial to reduce sensor and other noise in the input data, as is generally a problem with both HD and film inputs.

References

[1] S.-T. Hsiang and J. W. Woods, "Embedded video coding using invertible motion compensated 3D subband/wavelet filter bank," *Signal Processing: Image Communication*, 16, May 2001, 705–724.

[2] B. Haskell, A. Netravali, and A. Puri, *Digital Video: An Introduction to MPEG-2*, Kluwer Academic Pub., 1996.

[3] ITU-T, Video Coding Expert Group (VCEG), *H.26L test model long term number 9 (TML-9)*. draft 0, Dec. 21, 2001.

[4] S.-T. Hsiang and J. W. Woods, "Invertible three-dimensional analysis/synthesis system for video coding with half-pixel-accurate motion compensation," *Proc. Visual Comm. and Image Process. (VCIP)*, SPIE, 3653, Jan. 1999.

[5] B. Pesquet-Popescu and V. Bottreau, "Three-dimensional lifting schemes for motion compensated video compression," *Proc. Int. Conf. Accoust, Speech, and Signal Process. (ICASSP)*, IEEE, 1793–1796, May 2001.

[6] L. Luo, J. Li, S. Li, Z. Zhuang, and Y.-Q. Zhang, "Motion compensated lifting wavelet and its application in video coding," *Intl. Conf. on Multimedia and Expo (ICME)*, IEEE, Tokyo, Japan, August 2001.

[7] A. Secker and D. Taubman, "Motion-compensated highly scalable video compression using an adaptive 3D wavelet transform based on lifting," *Proc. Int. Conf. Image Process. (ICIP)*, IEEE, October 2001.

[8] A. Golwelkar and J. W. Woods *Improved Motion Vector Coding for the Sliding Window (SW-) EZBC Video Coder*, JTC1/SC29/WG11, MPEG2003/M10415, Dec. 2003, Hawaii, USA.

[9] A. M. Tekalp, *Digital Video Processing*, Prentice-Hall, Englewood cliffs, NJ, 1995.

[10] P. Chen and J. W. Woods, *Comparison of MC-EZBC and H.26L TML 8 on Digital Cinema Test Sequences*, ISO/IEC JTC1/SC29/WG11, MPEG2002/8130, Cheju Island, March 2002.

[11] D. S. Turaga and M. v.d. Schaar, *Unconstrained Motion Compensated Temporal Filtering*, ISO/IEC JTC1/SC29/WG11, MPEG2002/M8520, lagenfurt, AT, July 2002.

[12] D. Taubman, "Successive refinement of video: fundamental issues, past efforts and new directions," *Proc. Visual Comm. and Image Process. (VCIP)*, SPIE, 5150, Lugano, Italy, July 2003.

[13] P. Chen and J. W. Woods "Bi-directional MC-EZBC with lifting implementation," *IEEE Trans. Circ. and Sys. for Video Technology*, 14, 1183–1194, October 2004.

[14] Y. Wu and J. W. Woods, *Recent Improvements in the MC-EZBC Video Coder*, ISO/IEC/JTC1/SC29/WG11, MPEG2003/M10158, Hawaii, USA, Dec. 2003.

[15] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. on Circ. and Sys. for Video Technology*, 13, 620–636, July 2003.

[16] J.-R. Ohm, "Three dimensional subband coding with motion compensation," *IEEE Trans. on Image Process.*, 3, 559–571, Sept. 1994.

[17] S.-J. Choi and J. W. Woods, "Motion-compensated 3D Subband Coding of Video," *IEEE Trans. on Image Process.*, 8, 155–167, Feb. 1999.

[18] S.-J. Choi, *Three-dimensional Subband/Wavelet Video of Video with Motion Compensation*, Ph.D Thesis, ECSE Department, Rensselaer Polytechnic Institute, Troy, NY, 1996.

[19] S.-T. Hsiang and J. W. Woods, "Embedded image coding using zero-blocks of subband/wavelet coefficients and context modeling," *Proc. Int. Sympos. Cir. and Sys. (ISCAS)*, IEEE, 662–665, Geneva, 2000.

[20] W. Sweldens, "The lifting scheme: a new philosophy in biorthogonal wavelet constructions," *Wavelet Appl. Signal and Image Process. III*, SPIE, 2569, 68–79, 1995.

[21] Y. Wu and J. W. Woods, "Directional spatial IBLOCK for the MC-EZBC video coder," *Proc. Int. Conf. Accoust, Speech, and Signal Process. (ICASSP)*, IEEE, Montreal, May 2004.

[22] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications*, Prentice Hall, Englewood cliffs, NJ, 296–300, 2002.

[23] M. T. Orchard and G. J. Sullivan, "Overlapped block motion compensation: an estimation-theoretical approach," *IEEE Trans. on Image Process.*, 3, 693–699, Sept. 1994.

[24] Y. Wu, R. A. Cohen, and J. W. Woods, *An Overlapped Block Motion Estimation for MC-EZBC*, ISO/IEC/JTC1/SC29/WG11, MPEG2003/M10158, Brisbane, AU, Oct. 2003.

[25] Y. Wu, A. Golwelkar, and J. W. Woods, *MC-EZBC video proposal from Rensselaer Polytechnic Institute*, ISO/IEC JTC1/SC29/WG11, MPEG04/M10569/S15, Munich, March 2004.

[26] S.-T. Hsiang, *Highly Scalable Subband/Wavelet Image and Video Coding*, PhD Thesis, ECSE Department, Rensselaer Polytechnic Institute, Troy, NY, May 2002.

[27] S. S. Tsai, H.-M. Hang, and T. Chiang, *Motion Information Scalability for MC-EZBC: Response to Call for Evidence on Scalable Video Coding*, ISO/IEC JTC1/SC29/WG11, MPEG03/M9756, July 2003.

[28] A. Islam and W. A. Pearlman, "An embedded and efficient low-complexity hierarchical image coder," *Proc. Visual Comm. and Image Process. (VCIP)*, SPIE, 3653, 294–305, Jan. 1999.

[29] D. Taubman, *EBCOT (Embedded Block Coding with Optimized Truncation): A Complete Reference*, ISO/IEC JTC1/SC29/WG1, JPEG1998/N988, Sept. 1998.

[30] J. W. Woods and G. Lilienfield, "Resolution and frame-rate scalable video coding," *IEEE Trans. Video Technology*, 11, 1035–1044, Sept. 2001.

[31] D. Mukerjee and A. Said, *Structured Content Independent Scalable Meta-formats (SCISM) for Media Type Agnostic Transcoding: Response to CfP on DIA / MPEG-21*, ISO/IEC JTC1/SC29/WG11, MPEG2002/M8689, Klagenfurt, Austid, July 2002.

[32] T. Rusert, K. Hanke, and J. Ohm, "Transition filtering and optimized quantization in interframe wavelet video coding," *Proc. Visual Comm. and Image Process. (VCIP)*, SPIE, 5150, 682–693, Lugano, July 2003.

[33] Y. Andreopoulos, A. Munteanu, J. Barbarien, M. v.d. Schaar, J. Cornelis, and P. Schelkens, "In-band motion compensated temporal filtering," *Signal Processing: Image Communication*, 19, August 2004.

[34] L. Luo, F. Wu, S. Li, and Z. Zhuang, "Advanced lifting-based motion-threading (MTh) technique for the 3D wavelet video coding," *Proc. VCIP 2003*, SPIE, 5150, Lugano, Italy, July 2003.

[35] J. W. Woods and J.-R. Ohm, Eds., "Special issue on interframe subband/wavelet video coding," *Signal Process.: Image Communication*, 19, August 2004.

[36] J. Konrad, "Transversal versus lifting approach to motion-compensated temporal discrete wavelet transform of image sequences: equivalence and tradeoffs," *Proc. Visual Comm. and Image Process.* (*VCIP*), SPIE, San Jose, Jan. 2004.

[37] R. Xiong, F. Wu, S. Li, Z. Xiong, and Y.-Q. Zhang, "Exploiting temporal correlation with adaptive block-size motion alignment for 3D wavelet coding," *Proc. Visual Comm. and Image Process.* (*VCIP*), SPIE, San Jose, Jan. 2004.

[38] A. Secker and D. Taubman, "Highly scalable video compression with scalable motion vector coding," *IEEE Trans. on Image Process.*, 13, 1029–1041, August 2004.

[39] S.-T. Hsiang, J. W. Woods, and J.-R. Ohm, "Invertible temporal subband/wavelet filter banks with half-pixel-accurate motion compensation," *IEEE Trans. on Image Process.*, 13, 1018–1128, August 2004.

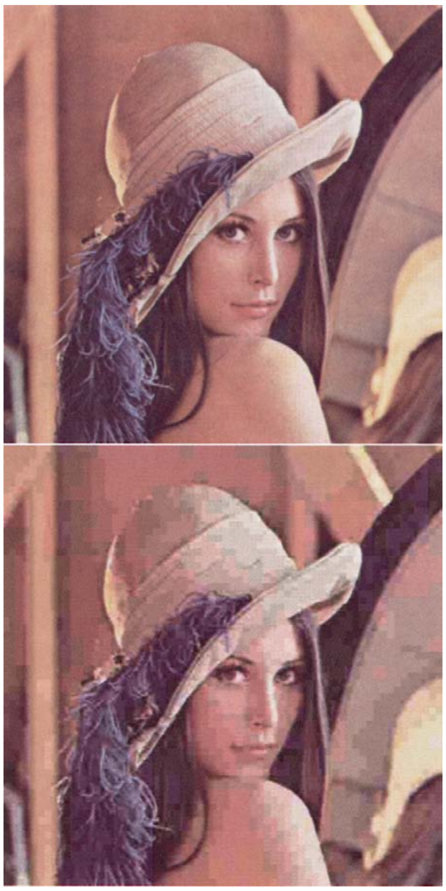


FIGURE 5.5.13 *Lena* image at 24-to-1 (top) and 96-to-1 (bottom) compression ratios.

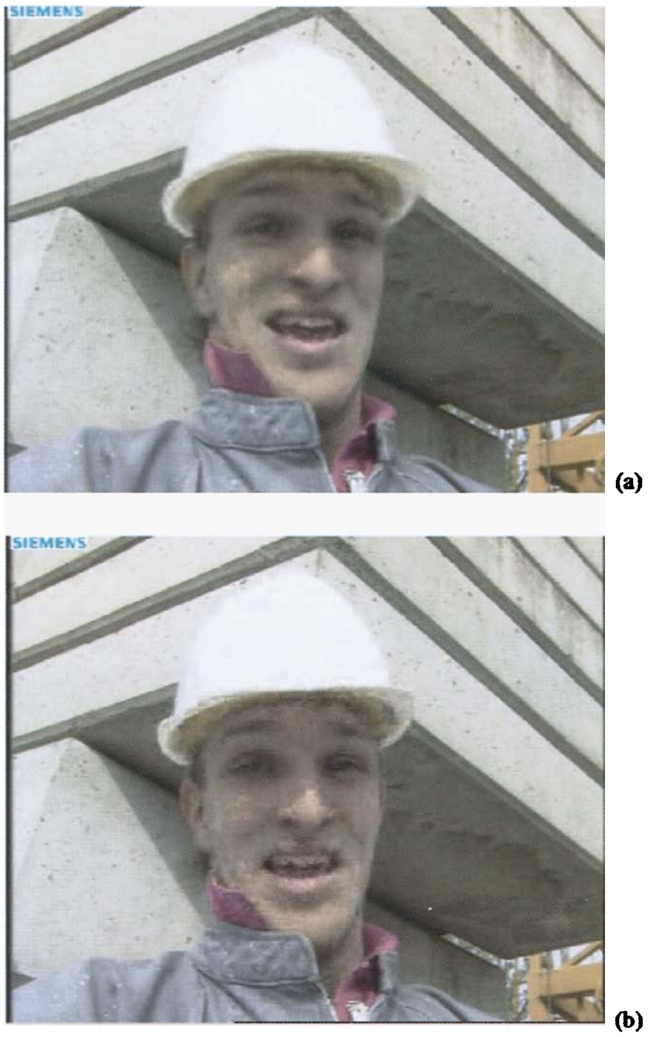


FIGURE 6.2.20 (a) Frame 141 in *Foreman* with OBMC, IBLOCK and color HVSBM at rate 512 Kbps @ 30 fps. (b) Frame 141 in *Foreman* without OBMC and IBLOCK at rate 512 Kbps @ 30 fps.

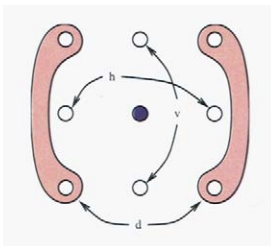


FIGURE 5.5.17 Neighbors involved in the contexts formation.

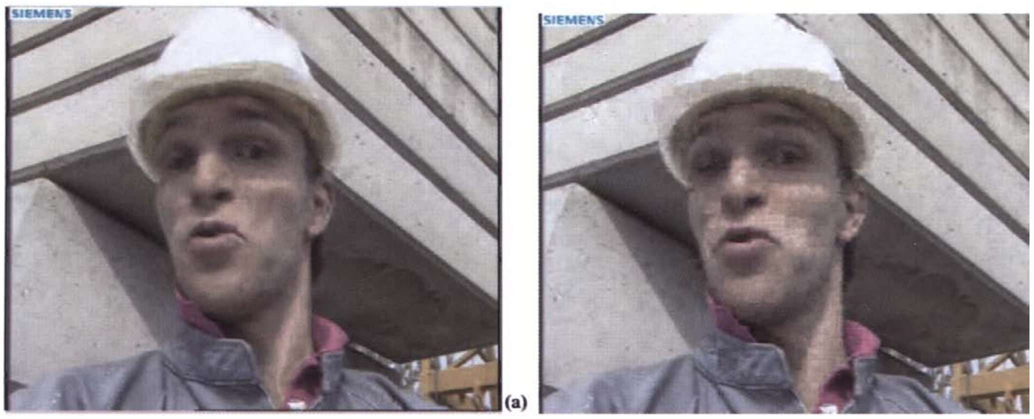


FIGURE 6.2.21 (a) Frame 43 in *Foreman* with OBMC, IBLOCK and color HVSBM (corresponding to original frame 86) at rate 256 Kbps @ 15 fps, half the original frame rate. (b) Frame 43 in *Foreman* without OBMC and IBLOCK, but with old YUV HVSBM (corresponding to original frame 86) at rate 256 Kbps @ 15 fps, half the original frame rate.