Available online at www.sciencedirect.com

Signal Processing: *Image Communication* 19 (2004) 793–849

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.com/locate/image

Video coding using the H.264/MPEG-4 AVC compression standard

Atul Puri[a,*], Xuemin Chen[b], Ajay Luthra[c]

[a] *RealNetworks, Inc., 2601 Elliott Avenue, Seattle, WA 98121, USA*
[b] *Broadcom Corporation, 16215 Alton Parkway, Irvine, CA 92619, USA*
[c] *Motorola, Inc., 6420 Sequence Drive, San Diego, CA 92121, USA*

Abstract

H.264/MPEG-4 AVC is a recently completed video compression standard jointly developed by the ITU-T VCEG and the ISO/IEC MPEG standards committees. The standard promises much higher compression than that possible with earlier standards. It allows coding of non-interlaced and interlaced video very efficiently, and even at high bit rates provides more acceptable visual quality than earlier standards. Further, the standard supports flexibilities in coding as well as organization of coded data that can increase resilience to errors or losses. As might be expected, the increase in coding efficiency and coding flexibility comes at the expense of an increase in complexity with respect to earlier standards.

In this paper, we first briefly introduce the video coding tools that the standard supports and how these tools are organized into profiles. As with earlier standards, the mechanism of profiles allows one to implement only a desired subset of the standard and still be interoperable with applications of interest. Next, we discuss how the various video coding tools of the standard work, as well as the related issue of how to perform encoding using these tools. We then evaluate the coding performance in terms of contribution to overall improvement offered by individual tools, options within these tools, and important combinations of tools, on a representative set of video test sequences and movie clips. Next, we discuss a number of additional elements of the standard such as, tools that provide system support, details of levels of profiles, and the issue of encoder and decoder complexity. Finally, we summarize our overview and analysis of this standard, by identifying, based on their performance, promising tools as well as options within various tools.

Keywords: MPEG; H.264; MPEG-4; AVC; JVT; Video compression; Video coding; Standard

1. Introduction

Earlier MPEG audio and video coding standards such as MPEG-1 and MPEG-2 [9,10] have enabled many familiar consumer products. For instance, these standards enabled video CDs and DVDs allowing video playback on digital

[*] Corresponding author. The author was previously with Apple Computer where much of this work was done.

E-mail addresses: apuri@real.com (A. Puri), schen@broadcom.com (X. Chen), aluthra@motorola.com (A. Luthra).

0923-5965/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2004.06.003

VCRs/set-top-boxes and computers, and digital broadcast video delivered via terrestrial, cable or satellite networks, allowing digital TV and HDTV. While MPEG-1 addressed coding of non-interlaced video at lower resolutions and bit-rates [23] offering VHS-like video quality, MPEG-2 addressed coding of interlaced video at higher resolutions and bit-rates [24] enabling digital TV and HDTV with commensurate video quality. The MPEG-1 standard was completed in 1992 while the MPEG-2 standard was completed in 1994. At the time of their completion they represented a timely as well as practical, state-of-the-art technical solution [3,8,15,19,23,24,27,33,34], consistent with the cost/performance tradeoffs of the products intended within the context of implementation technology available. The actual impact of these standards in terms of inexpensive consumer products and market penetration took at least 5 years from the time of completion of these standards.

MPEG-4 was launched to address a new generation of multimedia applications and services. The core of the MPEG-4 standard [29,11] was developed during a 5-year period (1995–1999), however MPEG-4 is a living standard with new parts added continuously as and when technology exists to address evolving applications. The premise behind MPEG-4 was future interactive multimedia applications and services such as interactive TV, Internet video etc where access to coded audio and video objects might be needed. The MPEG-4 standard consists of many more parts besides the traditional, audio, video systems and conformance parts of earlier standards. Our discussion in this paper is however limited only to video. The MPEG-4 video standard is designed as a toolkit standard with the capability to allow coding and thus access to individual objects, scalability of coded objects, transmission of coded video objects on error prone networks, as well as efficient coding of video objects. Further, MPEG-4 video also allows higher efficiency coding (than MPEG-1 and MPEG-2) of rectangular video without the necessity of dividing a scene into video objects prior to coding. The significant advances in core video standard referred to as MPEG-4 part 2, were achieved in the capability of coding of video objects while at the same time it clearly did improve coding efficiency over earlier standards. From coding efficiency standpoint, MPEG-4 video was evolutionary in nature as it built on coding structure of MPEG-2 and H.263 standards and adding enhanced/new tools but within the same coding structure. Thus, MPEG-4 part 2 offers a modest coding gain but only at the expense of a modest increase in complexity. The expectation was that since object-based video was the main focus, increase in complexity could be only justified for those applications only, not for pure rectangular video applications.

In the meantime, while highly interactive multimedia applications appear farther into the future than anticipated, there seems to be an inexhaustible demand for much higher compression to enable with as best video quality as possible, practical applications such as internet multimedia, wireless video, personal video recorders, video-on-demand, and videoconferencing. The H.264/MPEG-4 AVC standard [14] is a new state-of-the-art video coding standard that addresses aforementioned applications. The core of this standard was completed in the form of final draft international standard (FDIS) in June 2003 while an extension for professional applications is currently in progress. It promises significantly higher compression than earlier standards. The standard evolved from the original very promising work performed by ITU-T VCEG in their H.26L project over the period of 1999–2001, and with MPEG joining the effort in late 2001, a joint team of ITU-T VCEG and ISO MPEG experts was established for co-developing the standard. The resulting joint standard is called H.264 by VCEG and MPEG-4 part 10 by MPEG (as mentioned earlier, the original MPEG-4 video is referred to as MPEG-4 part 2). Another name for this standard is MPEG-4 advanced video coding (AVC) standard. Further, informally it is also referred to as Joint Video Team (JVT) standard as it resulted from collaborative effort of the VCEG and MPEG standards committees. Regardless of the preferred name, the standard achieves clearly higher compression efficiency, often quoted as, up to a factor of 2 [38], over the MPEG-2 video standard. As one would expect, the increase in compression efficiency comes at the cost of substantial increase in complexity, often quoted as factor of 4 for the decoder, while encoding complexity may be as high as factor of 9 over MPEG-2. Further, the exact improvement as well as the resulting complexity depends on the profile (subset) of the standard implemented, the choice of which is application dependent. It is worth noting that the standard uses the familiar motion compensated coding structure of earlier standards, a number of refinements to existing tools in earlier standards, as well as key new tools and coding optimization. Also, the coding performance benefits of individual tools are much more scene and bit-rate dependent, and different tools differ significantly in performance/complexity tradeoffs they offer. While the standard is discussed in a number of excellent papers [39,17], in this paper we address issues of practical video coding in an effort to help make informed selection of coding tools/profiles and parameters in coder design. The rest of the paper covers the various aspects as follows.

Section 2 of this paper presents an overview of the H.264/MPEG-4 AVC standard. It presents the coding structure, lists tools compared to earlier standards and discusses the operation of encoding and decoding compliant to this standard. Section 3 introduces prediction modes such as intra prediction, motion compensated prediction including multiple frames and multiple block sizes. Section 4 introduces, the key concepts of transform used, the quantization process and the loop filter to reduce blockiness artifacts. Section 5, focuses on entropy coding techniques such as context adaptive VLC (CAVLC) and context adaptive arithmetic (CABAC) coding. Section 6, addresses core issues in design of encoders for this standard, while many of the issues are similar to earlier standards, there are also several new issues to ensure high coding efficiency from this standard. Section 7 presents an experimental evaluation and analysis of performance of various tools included in the standard. Section 8 discusses special tools that do not impact coding efficiency but provide a supporting role to allow adaptation of the standard to various applications. Section 9 discusses the current profiles and levels structure and the motivation behind mapping of tools to profiles. Section 10 summarizes the findings of the paper.

2. H.264/MPEG-4 AVC codec overview

We now present an overview of coding as per the H.264/AVC standard.

2.1. Coding structure

The basic coding structure of this standard is similar to that of earlier standards and is commonly referred to as motion-compensated—transform coding structure. Coding of video is performed picture by picture. Each picture to be coded is first partitioned into a number of slices (it is possible to have one slice per picture also). Slices are individual coding units in this standard as compared to earlier standards as each slice is coded independently.

As in earlier standards, a slice consists of a sequence of macroblocks with each macroblock (MB) consisting of 16×16 luminance (Y) and associated two chrominance (Cb and Cr) components. In rest of the paper, the terms macroblock or MB will be used interchangeably.

Each macroblock's 16×16 luminance is partitioned into 16×16, 16×8, 8×16, and 8×8, and further, each 8×8 luminance can be sub-partitioned into 8×8, 8×4, 4×8 and 4×4. The 4×4 sub-macroblock partition is called a block. The hierarchy of video data organization is as follows:

picture [slices {macroblocks (sub-macroblocks (blocks (pixels)))}]

Currently, only 4:2:0 chroma format and 8-bit sample precision for luma and chroma pixel values is supported in the standard. In 4:2:0 chroma format, each macroblock associates two 8×8 chroma components with 16×16 luminance. Work is in progress to extend the standard to 4:2:2 and 4:4:4 chroma formats and higher than 8-bits resolution.

As mentioned earlier, slices are individually coded and are the coding units, while pictures plus associated data can be considered as being the access units. There are three basic slices types:

I—(Intra), P—(Predictive), and B—(Bi-predictive) slices. This is basically a nomenclature as well as functionality extension of the I-, P-, and B-picture concept of earlier standards [9–11]. As a side note, work on original B-pictures in MPEG was based on ideas in [21,22,25,28], and subsequent refinements. In H.264/MPEG-4 AVC standard, I-slice macroblocks are compressed without using any motion prediction (also true of all earlier standards as well) from the slices in other pictures. A special type of picture containing I-slices only called instantaneous decoder refresh (IDR) picture is defined such that any picture following an IDR picture does not use pictures prior to IDR picture as references for motion prediction. Thus after decoding an IDR all following coded pictures in decoding order can be decoded without the need to reference to any decoded picture prior to the IDR picture. IDR pictures can be used for random access or as entry points in a coded sequence. P-slices consist of macroblocks that can be compressed by using motion prediction, but P-slices can also have intra macroblocks. Macroblocks of a P-slice when using motion prediction must use one prediction only (uni-prediction). Unlike previous standards, the pixels used as reference for motion compensation can either be in past or in future in the display order. Also, both I- and P-slices may or may not be marked as used for reference. B-slices also consist of macroblocks that can be compressed by using motion prediction and like P-slices can also have intra macroblocks. Macroblocks of a B-slice when using motion prediction can use two predictions (bi-prediction). Like earlier standards, one of the motion predictions can be in past and the other in future in the display order, but unlike earlier standards, it is also possible to have both motion predictions from past, or both motion predictions from future. Also, unlike earlier standards B-slices can also be used as reference for motion prediction by other slices in the future or in the past. Such B-slices that are used as reference for motion prediction, are informally called *stored B-slices* due to the need for storing them unlike traditional B-slices. Besides I-, P-, B- slices, there are two derived slice types called SI- (switching I-) and SP- (switching P-) slices. The SI- and SP- slices allow switching between multiple coded streams such as different bit-rate encoded versions of the same content, as might be needed by some streaming applications.

2.2. *Overview of coding tools*

The H.264/MPEG-4 AVC standard, while bringing new coding tools and concepts, still builds on the proven and familiar framework of motion compensated transform coding used by earlier standards. Thus, while the details such as exact prediction/coding modes, transform, or entropy coding method may be different, the overall coding structure is still the same. The primary goal of H.264/MPEG-4 AVC standard is significantly higher coding efficiency although it also includes tools to allow error resilient coding in certain applications. Unlike MPEG-2, MPEG-4 part 2 or H.263, it currently does not support layered scalable coding. Further, unlike MPEG-4 part 2, it does not support object-based video- or object-based scalable coding. The focus of the standard is on achieving higher coding efficiency not only for progressive but also for interlaced video. The standard consists of a large number of tools designed to address efficient coding over a wide variety of video material. Similar to MPEG-2 or MPEG-4 part 2, it includes the concepts of profiles and levels and while there are many tools included in the standard, only the tools supported by a profile of interest need to be implemented. In order to best introduce the standard, a comparison of tools of this standard with respect to tools in MPEG-2 and MPEG-4 part 2, is in order; Table 1 presents such a comparison.

2.3. *Overview of profiles*

While H.264/MPEG-4 AVC standard contains a rich set of video coding tools, not all the coding tools are required for all applications. For example, error resilience tools may not be needed for video stored on a compact disk or on networks with very few errors. If every decoder was forced to implement all the tools, it would make such a decoder unnecessarily too complex and thus not very practical. On the other hand, interoperability between applications requires that

Table 1
Comparison of main coding tools in MPEG-2, MPEG-4 Part 2, and H.264/MPEG-4 AVC

Tools	MPEG-2	MPEG-4 Part 2	H.264/MPEG-4 AVC
I-, P- and B-pictures	Yes	Yes	Yes, and, I-, P- and B-slices
Flexible picture prediction structure and stored B picture	Basic, no stored B-picture	Basic, no stored B-picture	Yes, allowed
Transform	8×8 DCT	8×8 DCT	Approximation of 4×4 DCT (a bit-exact transform)
Intra prediction in blocks of intra MB	Fixed prediction of DC coefficient	Adaptive prediction of DC coefficient, and first row/ column of AC coefficients	Adaptive spatial prediction of 4×4 or 16×16 pixel blocks
MC prediction 16×16, 16×8	16×16; interlace only 16×8	16×16; interlace only 16×8	Yes, 16×16, 16×8, 8×16
MC prediction 8×8	No	Yes	Yes
MC Prediction sub8×8	No	No	Yes, 8×4, 4×8, 4×4
MC prediction with $\frac{1}{4}$ pel	No, $\frac{1}{2}$ pel only	Yes, $\frac{1}{2}$ pel and $\frac{1}{4}$ pel	Yes, $\frac{1}{4}$ pel only
Multi reference prediction	No	No	Yes
Direct prediction mode in B pictures	No	1 Mode only: temporal direct with mv update	2 Modes: temporal direct no mv update, spatial direct
Global MC	No	Yes	No
Unrestricted MVs	No	Yes	Yes
Motion vector prediction	Simple	Better, uses median	Uses median, and segmented
Intra DC nonlinear quant, Intra AC directional scans and improved chroma quant	No	Special nonlinear quant, MB level adaptive directional scan, improved chroma quant	Nonlinear DC quant, horizontal and vertical scans, improved chroma quant
Quantizer weighting matrices	Yes	Yes	No
Efficient quantizer overhead	No	Yes	Yes
Block artifact reduction	Postprocessing often used but no suggested filter	Postprocessing suggested with provided optional filter	Mandatory in-loop filter, postprocessing may also be used
Adaptive VLC coding	No	Yes, uses 2 tables	Yes, very content adaptive
Adaptive arithmetic coding	No	No, not for DCT coefficients	Yes, very content adaptive
Weighted prediction in P/B	Usual ($\frac{1}{2}$, $\frac{1}{2}$) weighting of forward and backward prediction in B-pictures	Usual ($\frac{1}{2}$, $\frac{1}{2}$) weighting of forward and backward prediction in B-pictures	Yes, very flexible
Arbitrary slice order and flexible macroblock ordering	No	No	Yes
Error resilient coding support	Concealment motion vectors for intra MB, very basic data partitioning	Resynch marker and header extension, reversible VLC, data Partitioning, new pred	Ref selection, data partitioning, arbitrary slice order, flexible macroblock order
Arbitrary object shape coding support	No	Yes, gray level or binary shapes and related motion and texture, sprite coding	No
Scalable coding support	Yes, layered picture spatial, SNR, temporal scalability	Yes, layered picture/object spatial and temporal scalability	With some support on temporal and SNR scalability
Interlace video coding support	Yes, field picture, MB adaptive frame/field, frame/ field scan	Yes, field picture, MB adaptive frame/field, frame/ field scan	Yes, frame pictures, field pictures, picture adaptive frame/field, MB adaptive frame/field, frame/field scan
Stream switching, splicing and random access	Basic, intra pictures	Basic, intra pictures	Intra pictures/slices, SI/SP switching pictures/slices
Division-free decoding capability	Yes	No	Yes

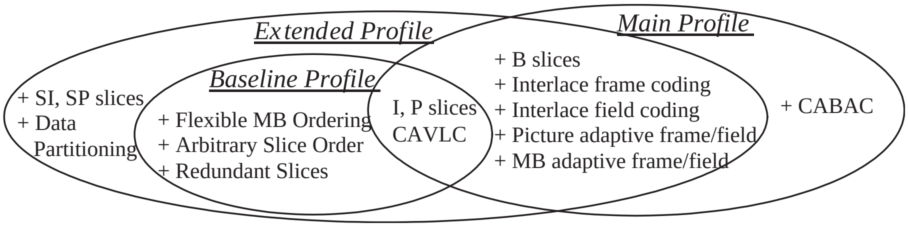

Fig. 1. Current Profile structure of H.264.

certain bit-streams be decodable by not only class of decoders that address that application, but related applications as well. Therefore, the standard defines subsets of coding tools intended for different classes of applications. These subsets are called Profiles. A decoder may choose to implement only one subset (profile) of tools. Currently, the following three profiles are defined but more may be added as deemed necessary.

- Baseline profile,
- Main profile,
- Extended profile.

Fig. 1 provides a pictorial view of the current organization of the standard into the three aforementioned profiles.

The Baseline profile includes I- and P-slice coding, enhanced error resilience tools (flexible macroblock ordering (FMO), arbitrary slices and redundant slices), and CAVLC. It does not include B-slices, SI- or SP-slices, interlace coding tools, and entropy coding with arithmetic coding (CABAC). It was designed for low delay applications, as well as for applications that run on platforms with low processing power and in high packet loss environment. Among the three profiles, it offers the least coding efficiency.

The Extended profile is a superset of the Baseline profile. Besides tools of the Baseline profile it includes B-, SP- and SI-slices, data partitioning, and interlace coding tools. It however does not include CABAC. It is thus more complex but also provides better coding efficiency. Its intended applications were streaming video.

The Main profile includes I-, P- and B-slices, interlace coding, CAVLC and CABAC. It does not include some error resilience tools (e.g. FMO), data partitioning, or SI and SP slices. It shares common tools such as I- and P-slices, and CAVLC with both the Baseline and Extended profiles. In addition it shares B-slices and interlaced coding tools with the Extended-profile. The Main profile was designed to provide the highest possible coding efficiency.

Additional details of profiles as well as semantic constraints (levels) are described in Section 9.1.

2.4. H.264/MPEG-4 AVC codec

Similar to earlier standards, the H.264/MPEG-4 AVC standard specifies the syntax for a compliant bitstream as well as a set of decoding semantics that describe how to interpret the syntax elements in the bitstream to produce decoded pictures. Thus the decoding operations and thus the decoder is fully specified but there is considerable flexibility in design of the encoder. However, as in earlier standards, even encoders must follow a standard set of operations for the resulting picture quality to be good, although algorithmic shortcuts are often taken at the encoder to tradeoff picture quality performance for speed. In fact, due to a plethora of prediction modes in H.264/MPEG-4 AVC, considerable effort was put by the JVT committee in demonstrating good coding quality (although optimized encoding is not standardized) to show value of many of the available prediction modes.

As mentioned earlier, this standard follows a similar coding structure as earlier video coding standards but with many important enhancements. Fig. 2 shows the block diagram of an example encoder. The encoder follows the classic DPCM encoding loop of motion compensated

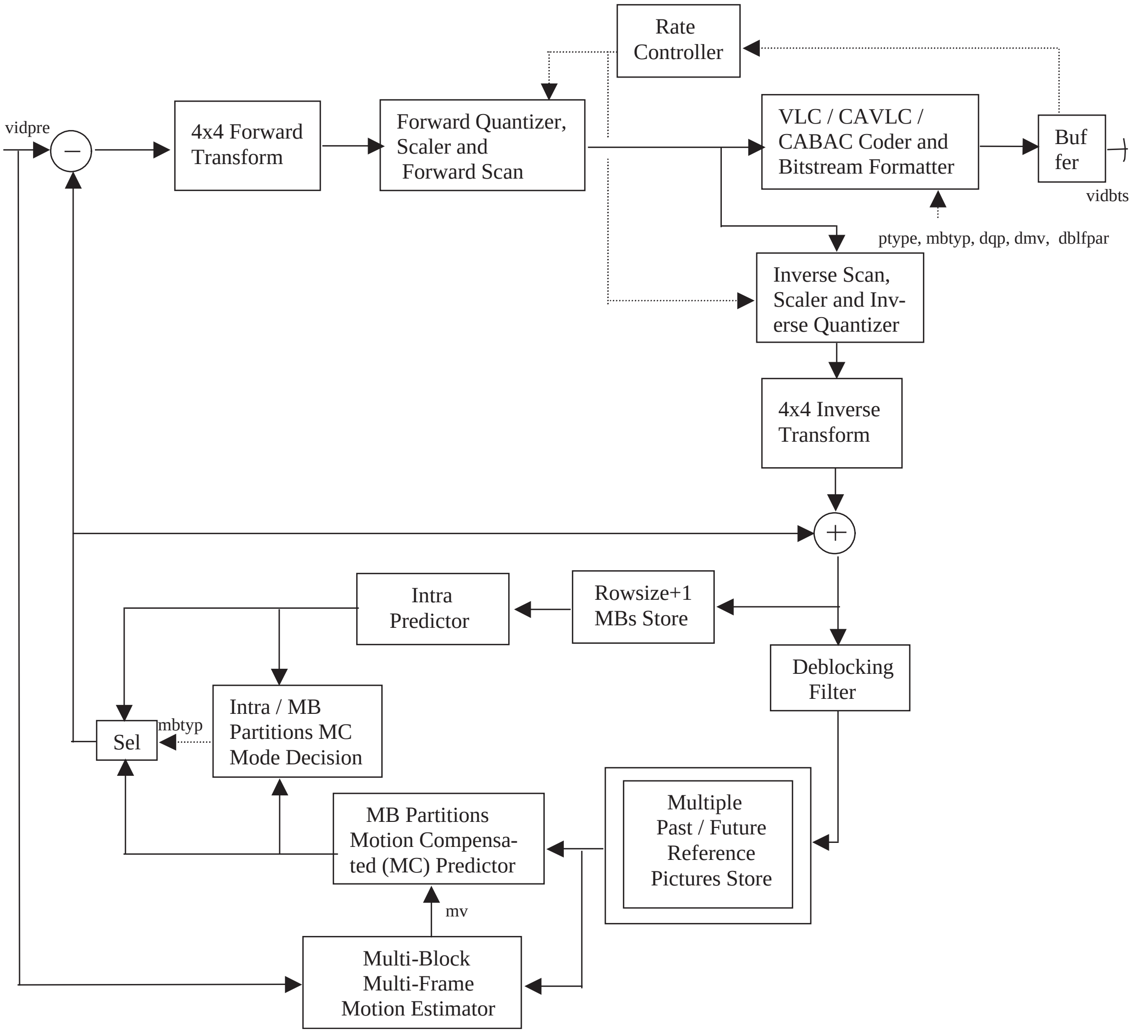
Rate Controller
vidpre
4x4 Forward Transform
Forward Quantizer, Scaler and Forward Scan
VLC / CAVLC / CABAC Coder and Bitstream Formatter
Buf fer
vidbts
ptype, mbtyp, dqp, dmv, dblfpar
Inverse Scan, Scaler and Inverse Quantizer
4x4 Inverse Transform
Intra Predictor
Rowsize+1 MBs Store
Deblocking Filter
Intra / MB Partitions MC Mode Decision
Sel
mbtyp
MB Partitions Motion Compensated (MC) Predictor
Multiple Past / Future Reference Pictures Store
mv
Multi-Block Multi-Frame Motion Estimator

Fig. 2. H.264/MPEG-4 AVC encoder block diagram.

transform coding as in earlier standards although the details are somewhat different. H.264/MPEG-4 AVC includes a number of motion compensated prediction modes requiring multi-frame, variable block-size [20,32] (also known as multi-partition MB motion estimation) as well as intra prediction modes. Each slice is coded a macroblock at a time (except in the case of interlace coding) and from it, its prediction signal is subtracted; the prediction signal is generated using best of the prediction from many possible candidate modes. The residual difference signal is coded with 4×4 transform and quantized and scaled, and scanned prior to entropy coding by CAVLC or CABAC. The rest of the block diagram represents the local decoder in the encoder including the inverse scan, scaler and inverse quantization, inverse transform, deblocking filter, and motion compensated prediction and intra prediction. The key encoder-only operations consist of motion estimation, macroblock type mode decision, and rate control; to a large extent these operations are similar, in principle, but much more complex than that in earlier standards.

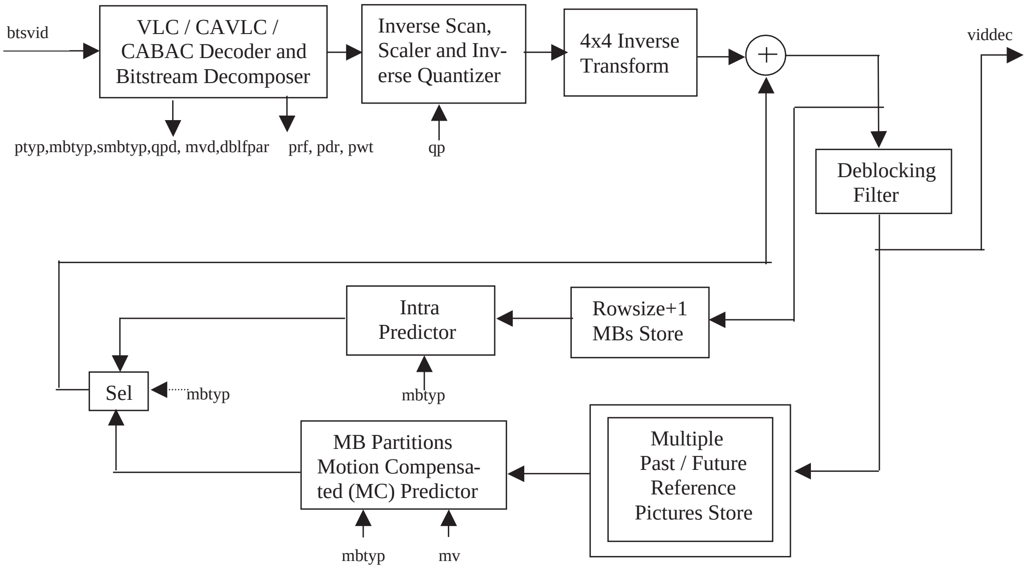
btsvid
VLC / CAVLC / CABAC Decoder and Bitstream Decomposer
Inverse Scan, Scaler and Inverse Quantizer
4x4 Inverse Transform
+
viddec
ptyp,mbtyp,smbtyp,qpd, mvd,dblfpar
prf, pdr, pwt
qp
Deblocking Filter
Intra Predictor
Rowsize+1 MBs Store
Sel
mbtyp
mbtyp
MB Partitions Motion Compensated (MC) Predictor
Multiple Past / Future Reference Pictures Store
mbtyp
mv

Fig. 3. H.264/MPEG-4 AVC decoder block diagram.

Motion compensated prediction can use a number of block sizes such as 16 × 16, 16 × 8, 8 × 16, 8 × 8, 8 × 4, 4 × 8, and 4 × 4. Further, $\frac{1}{4}$ pixel motion compensation uses 6 tap filters in horizontal and vertical direction for $\frac{1}{2}$ pixel positions, and a 2-tap horizontal, vertical or diagonal filter for $\frac{1}{4}$ pixel refinement. Intra prediction can be performed on spatial blocks of 16 × 16 or 4 × 4 size and uses previous decoded pixels. The number of reference frames depends on the constraints or levels of a profile. The residual signal after prediction is transform coded with 4 × 4 block size. To avoid blocking artifacts, a deblocking filter is employed in the loop, which means that decoder must use the exact filter in the same way. Entropy coding uses three different qmethods: Exp-Golomb codes, context adaptive variable length coding (CAVLC), and context adaptive binary arithmetic coding (CABAC). Interlace is handled somewhat differently than earlier standards; four coding modes are available such as frame pictures, field pictures, frame pictures with picture adaptive frame/field (PicAFF), and frame pictures with MB adaptive frame/field (MBAFF).

The result of the encoding process is an H.264/MPEG-4 AVC compliant bitstream. This bitstream may be raw or formatted for storage or delivery over specific network and is eventually input to H.264 decoder.

We now discuss the operation of decoder, to some extent already briefly discussed. Fig. 3 shows the block diagram of a general H.264/MPEG-4 AVC decoder. It includes all the control information such as picture or slice type, macroblock types and subtypes, reference frames index, motion vectors, loop filter control, quantizer step size etc, as well as coded data comprising of quantized transform coefficients.

The decoder of Fig. 3 works similar to the local decoder at the encoder; a simplified description is as follows. After entropy (CABAC or CAVLC) decoding, the transform coefficients are inverse scanned and inverse quantized prior to being inverse transformed. To the resulting 4 × 4 blocks of residual signal, an appropriate prediction signal (intra or motion compensated inter) is added depending on the macroblock type *mbtyp* (and *submbtype*) mode, the reference frame, the motion vector/s, and decoded pictures store, or in intra mode. The reconstructed video frames undergo deblock filtering prior to being stored for future use for prediction. The frames at the output of deblocking filter may need to undergo reordering prior to display.

2.5. Components of the codec

We now describe components of Figs. 2 and 3 in a bit more detail.

2.5.1. Transform

A 4 × 4 integer transform (rather than the 8 × 8 floating point transform in MPEG-2 or MPEG-4 Part 2) whose transform coefficients are explicitly specified is used in AVC and allows it to be perfectly invertible. In AVC, the transform coding always utilizes predictions to construct the residuals, even in the case of Intra MBs. That is, the pixel values in a MB are always predicted, either from neighboring pixels in the same picture (in the case of Intra MBs), or from pixels in one or two previously decoded reference pictures (in the case of Inter MBs).

2.5.2. Quantization and scan

The standard specifies the mathematical formulae of the quantization process. Unlike MPEG-2, the current version of AVC does not support downloadable quantization matrices. Quantization is also called "scaling" in the standard. The scale factor for each element in each sub-block varies as a function of the quantization parameter associated with the MB that contains the sub-block, and as a function of the position of the element within the sub-block. The rate-control algorithm in the encoder controls the value of quantization parameter.

Two scan patterns for 4 × 4 blocks are used in this standard — one for frame coded MBs and one for field coded MBs.

2.5.3. CAVLC and CABAC entropy coders

VLC encoding of syntax elements for the compressed stream is performed using Exp-Golomb codes. For transform coefficient coding AVC includes two different entropy coding methods for coding quantized coefficients of the transform. Both methods are permitted in Main profile. The entropy coding method can change as often as every picture. These methods are CAVLC and CABAC.

2.5.4. Loop filter

The AVC loop filter, also called the deblocking filter, operates on a MB after motion compensation and residual coding, or on a MB after intra prediction and residual coding, depending whether the MB is inter coded or intra coded. The loop filter is specified to operate on the MBs in raster scan order. The result of the loop filtering operation is stored as a reference picture (except of course for pictures that are not used as reference pictures). Loop filtering operates on the edges of both MB and 4 × 4 sub-blocks. The operations are somewhat different at the MB edges than they are at the inner edges. The loop filter operation is adaptive in response to several factors, among them the quantization parameter of the current and neighboring MBs; the magnitude of the MV; and the MB coding type; as well as the values of the pixels to be filtered in both the current and neighboring blocks and MBs.

2.5.5. Mode decision

It determines the coding mode for each MB. Mode decision to achieve high efficiency may use rate/distortion optimization; however such mode decision can also be quite complex. Mode decision may at times need to work with rate control algorithm also. The outcome of mode decision is the best-selected coding mode for a macroblock.

2.5.6. Intra prediction

Prediction for intra MBs is called intra prediction and is done in pixel-domain in this standard. By comparison, in MPEG-2 only a simple intra prediction is performed on DC coefficients, and in MPEG-4 both DC and several AC coefficients of 8 × 8 DCT coefficients can be predicted; both of these predictions are in transform-domain [29,34,33]. In this standard, intra prediction forms predictions of pixel values as linear interpolations of pixels from the adjacent edges of neighboring MBs (or 4 × 4 blocks) that are decoded before the current MB (or 4 × 4 block), i.e. MBs that are above and/or to the left. The interpolations are directional in nature, with multiple modes, each implying a spatial direction of prediction. For luminance pixels with 4 × 4 partitions, 9 intra-prediction modes are defined. Four intra prediction modes are defined when a 16 × 16 partition is used. For chrominance pixels, 4 different modes are defined, that are similar in nature to the 4 modes of 16 × 16 luma intra-prediction process. Both chroma blocks, Cb and Cr, use the same prediction mode.

2.5.7. *Inter prediction*

This block includes both motion estimation (ME) and motion compensation (MC). The ME/MC process performs prediction. It generates a predicted version of a rectangular array of pixels, by choosing another similarly sized rectangular array of pixels from a previously decoded reference picture and translating the reference array to the position of the current rectangular array. In AVC, the rectangular arrays of pixels that are predicted using MC can have the following sizes: 4 × 4, 4 × 8, 8 × 4, 8 × 8, 16 × 8, 8 × 16, and 16 × 16 pixels. The translation from other positions of the array in the reference picture is specified with quarter pixel precision. The filter to perform the translation uses 6 taps in the x and y dimensions, plus another step that uses 2 taps. The foregoing is primarily concerned with luma values; motion compensation is applied to chroma values in a slightly different way, with smaller arrays of samples due to the 4:2:0 sampling. Chroma MVs at $\frac{1}{8}$ pixel resolution are derived from transmitted luma MVs of $\frac{1}{4}$ pixel resolution, and simpler filters are used for chroma as compared to luma.

2.6. *Encoding process*

Next, Fig. 4 provides an overview of the major encoding functions and decisions that must be performed starting at the access unit level. The individual steps involved in encoding as per this figure are shown in Figs. 5–7.

2.6.1. *Slice header and MB syntax encoding*

Different elements in the slice header and MB syntax are coded using different code types. Fig. 5(a) illustrates how an encoder must switch between Exp-Golomb coding and fixed length coding when encoding these syntax elements.

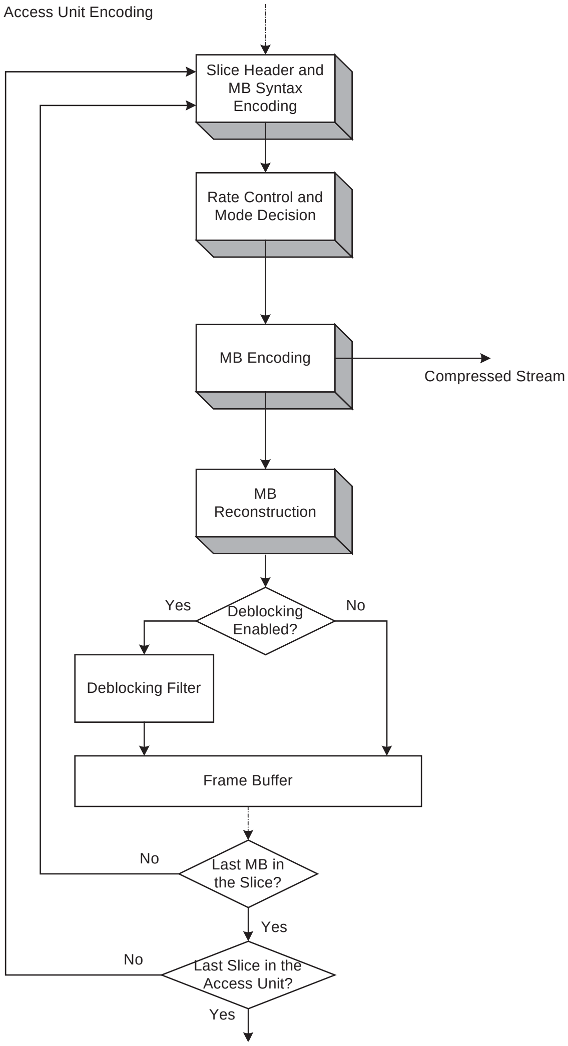

Fig. 4. Encoding flow diagram for access units.

At the macroblock level, the main decision is whether to code the MB as an intra-MB or an inter-MB as shown in Fig. 5(b).

2.6.2. *Rate control and mode decision*

This block is responsible for bit allocation and rate control by controlling how each macroblock is coded. These issues are addressed in encoding discussion in Section 6 of this paper.

2.6.3. *Intra-MB and inter-MB encoding*

Fig. 6(a) shows a detailed diagram of the process for intra-MB encoding. The encoding

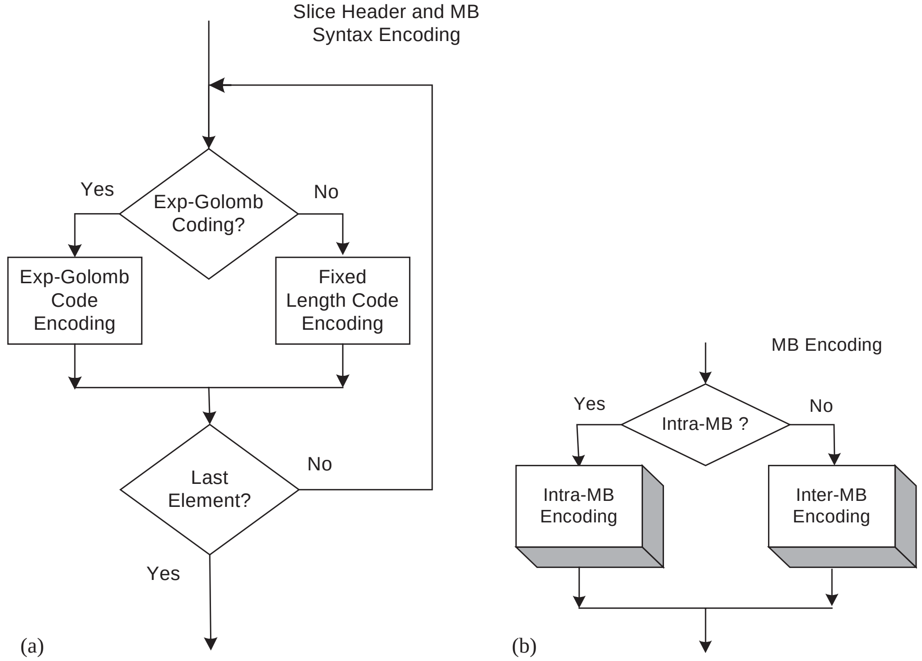

Fig. 5. Encoding flow diagrams for (a) slice header and MB syntax, (b) MB encoding.

tasks performed here depend first on whether I-PCM encoding is performed. If I-PCM encoding is performed, the MB can be coded directly. If I-PCM encoding is not performed, the intra-prediction, transform and quantization operations are performed on the block and then either CABAC or CAVLC encoding is utilized to generate the compressed stream.

Fig. 6(b) shows a detailed diagram of the process for inter-MB encoding. The encoding tasks performed here depend first on whether the MB is a skipped MB. If the MB is a skipped MB, the MB_16 × 16 MV or the direct-mode MV must be calculated depending on whether the MB is in a P-slice or a B-slice, respectively. If the MB is not a skipped MB, a check to see if direct mode is being used is performed. If direct mode is being used, the direct mode MVs must be derived. If direct mode is not being used, the differential MVs must be encoded. Motion compensation and prediction is then performed followed by the transform and quantization operations. The next step is to check to see if the block is being coded. If not, the encoder can proceed directly to MB reconstruction. If the block is being coded, then either CABAC or CAVLC encoding is performed before MB reconstruction.

2.6.4. *MB reconstruction*

A MB may be classified as "coded" or as "not coded". As shown in Fig. 7 both types of MBs require prediction blocks for reconstruction. In case of "coded" MB, in addition, reconstructed prediction error block is needed and is generated by, inverse quantization of corresponding quantized transform coefficient block, and then followed by inverse transform of the inverse quantized block. This reconstructed prediction error block is then added to the prediction block to generate the reconstructed MB. For "not coded" MB, only prediction block is needed.

2.7. *Decoding process for residue blocks*

In Fig. 8, we now present a flow diagram of the decoding process for the residual signal. First, it is

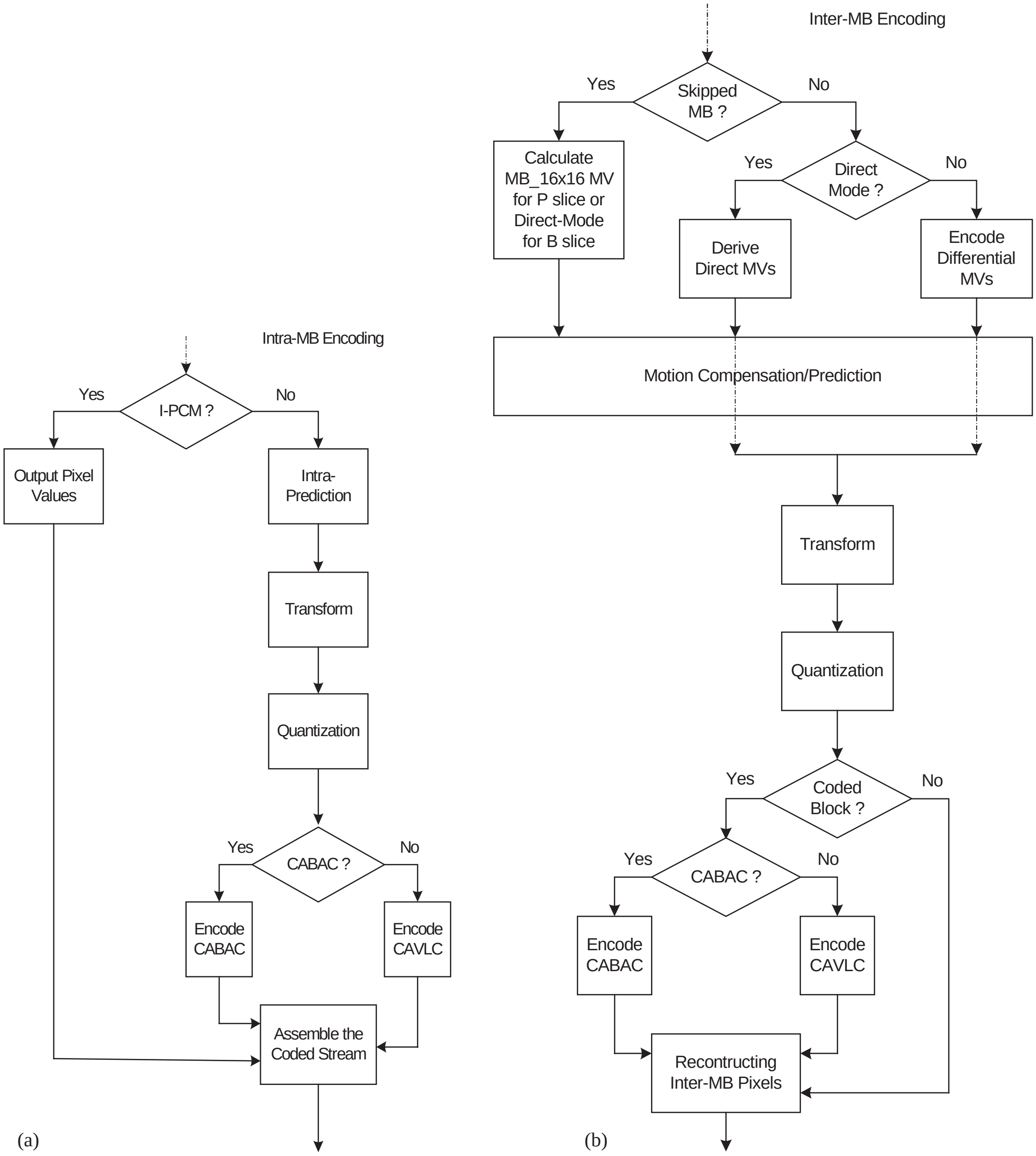

Fig. 6. Encoding flow diagrams for (a) intra MB, (b) inter MB.

detected if CAVLC or CABAC was used for the coding of residual signal and thus the corresponding entropy decoder is used for residual signal decoding. Next, we determine if the block being decoded is derived from intra 4×4 luma DC or chroma 2×2 DC (based on 16×16 intra prediction) or other (4×4 intra prediction coded or inter coded with motion compensation). Depending on the outcome, proper dequantization and inverse transform (Hadamard or HCT) is applied resulting in eventual reconstruction of residue block. The process is repeated for all blocks of

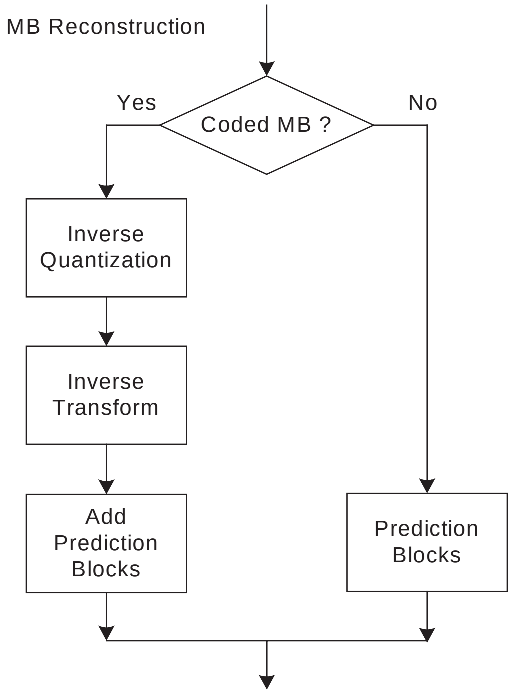

Fig. 7. Encoding flow diagram for macroblock reconstruction.

a macroblock and for all macroblocks of the picture being decoded.

3. Intra prediction, and motion compensated prediction

3.1. Intra prediction

As a first step in coding of a macroblock in intra mode, spatial prediction is performed on either 4×4 or 16×16 luminance blocks. Although, in principle, 4×4 block prediction will offer more efficient prediction compared to 16×16 block, in reality, taking into account the mode decision overhead, sometimes 16×16 block based prediction may offer overall better coding efficiency.

3.1.1. 4×4 prediction of luma

Each of the 16, 4×4 pixel blocks of the luminance component of an intra macroblock can be predicted using either the dc mode or in one of the eight coding directions listed in Fig. 9(a) and illustrated in Fig. 9(b). For the purpose of illustration, Fig. 9(c) shows a 4×4 block of pixels a,b,c,…,p, belonging to a macroblock to be coded. Pixels $A, B, C, \ldots, H$, and I, J, K, L, M are already decoded neighboring pixels used in computation of prediction of pixels of current 4×4 block.

For instance, if vertical prediction is employed, pixel A is used to predict pixel column a, e, i, m, pixel B is used to predict pixel column b, f, j, n, pixel C is used to predict pixel column, c, g, k, o, and pixel D is used to predict pixel column, d, h, l, p. Likewise in the case of horizontal prediction, pixels I, J, K, L, predict rows starting, respectively, with pixels, a, e, i, and m. In the case of dc prediction, an average of 8 pixels, A, B, C, D, I, J, K, L, is used as prediction of each of the 16 pixels of the 4×4 block. Directional predictions use a linear weighted average of pixels from among A, H, I–M, depending on the specific direction of the prediction.

3.1.2. 16×16 prediction of luma

Each 16×16 pixel block of luminance component of an intra macroblock can also be predicted using 16×16 prediction. For 16×16 block prediction, 4 modes are supported as listed in Fig. 10 comprising of the dc, vertical, horizontal and plane prediction. In vertical prediction, each of the 16 columns (of 16 pixels each) of current macroblock are predicted using only 1 past decoded pixel each, similar to the case of prediction of 4 pixels of column by a single decoded pixel in the case of 4×4 intra prediction. The horizontal prediction predicts an entire row of 16 pixels by a past decoded neighboring pixel, the process is repeated for each of the 16 rows. The dc prediction uses an average of past decoded row and column of pixels to predict all pixels of the 16×16 block. The planar prediction uses weighted combination of horizontal and vertical adjacent pixels.

The neighboring pixels used for prediction of 16×16 luminance component of current macroblock belong to neighboring decoded macroblocks.

3.1.3. Prediction of chroma

Per macroblock there are 2, 8×8 blocks of chroma one corresponding to each of the

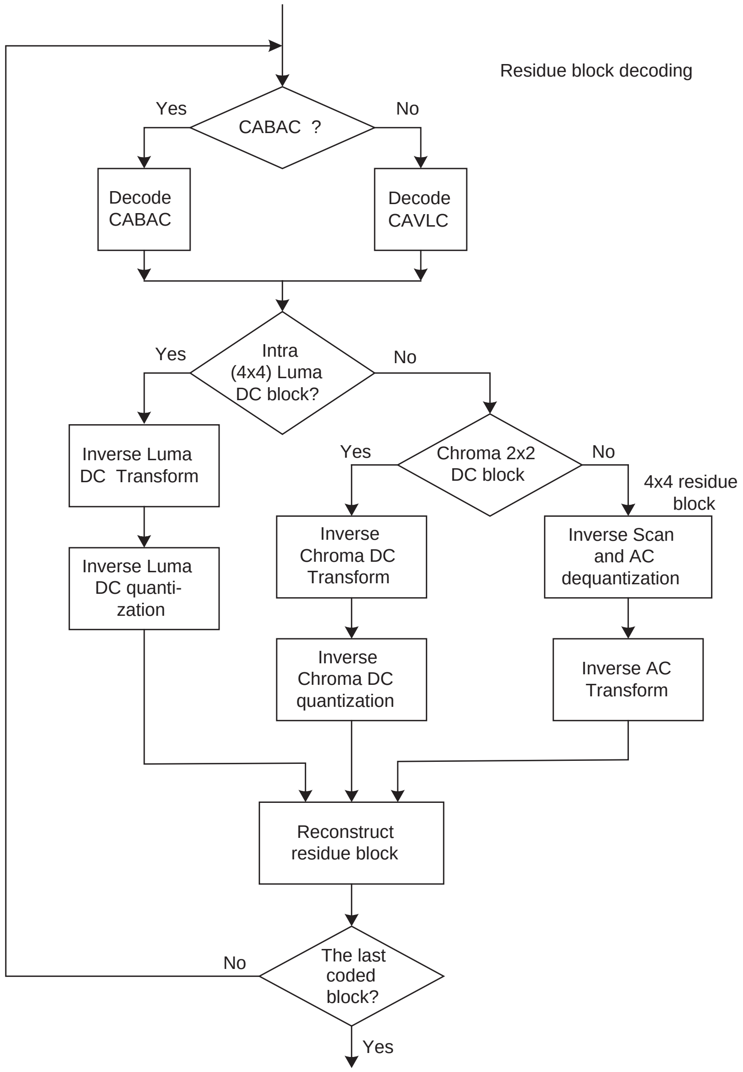
Residue block decoding
CABAC ?
Yes
No
Decode CABAC
Decode CAVLC
Intra (4x4) Luma DC block?
Yes
No
Inverse Luma DC Transform
Inverse Luma DC quanti-zation
Chroma 2x2 DC block
Yes
No
4x4 residue block
Inverse Chroma DC Transform
Inverse Chroma DC quantization
Inverse Scan and AC dequantization
Inverse AC Transform
Reconstruct residue block
The last coded block?
No
Yes

Fig. 8. Decoding flow diagram for residual signal.

components, Cb and Cr. Each 8×8 block of chroma is subdivided into 4, 4×4 blocks such that each 4×4 block depending on its location uses a pre-fixed prediction using decoded pixels of corresponding chroma component.

3.2. *Inter/motion compensated prediction*

As in the prior video coding standards, inter-macroblocks are coded using block motion compensation to determine block prediction error. However, because an MB can be partitioned into sub-blocks of various sizes and can have different coding types, a number of rules are defined so that predictions can be efficiently performed. The process of motion compensation in MB decoding is illustrated in Fig. 11.

3.2.1. *Multiple reference pictures for motion compensation*

Generally in previous standards, for prediction of blocks of a P-picture being coded, only immediately previous I- or P-picture is used as a reference. An exception was reference picture

Num	Intra 4x4 Pred Mode
0	vertical
1	horizontal
2	dc
3	diagonal_down_left
4	diagonal_down_right
5	vertical_right
6	horizontal_down
7	vertical_left
8	horizontal_up

(a)

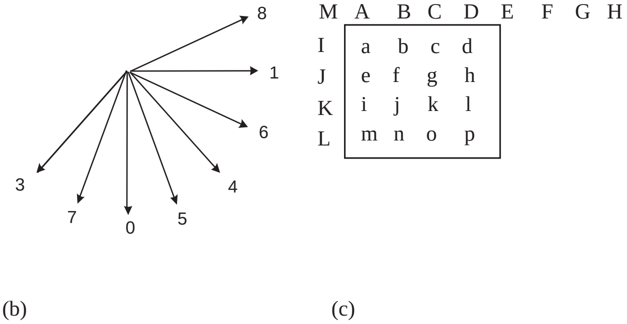

Fig. 9. (a) Intra 4 × 4 prediction modes, (b) prediction directions, (c) block prediction process.

Num	Intra 16x16 Pred Mode
0	vertical
1	horizontal
2	dc
3	plane

Fig. 10. Intra 16 × 16 prediction modes.

selection in H.263 and MPEG-4, and enhanced reference selection in H.263. The H.264/MPEG-4 AVC standard further extends the enhanced reference picture selection to enable efficient coding by allowing an encoder to select, for motion compensation purposes, among a larger number of pictures that have been decoded and stored. The same extension of referencing capability is also applied to motion-compensated bi-prediction, which is restricted in prior standards to using two specific pictures only (one of these being the previous I- or P-picture in display order and the other being the next I- or P-picture in display order).

3.2.2. Multiple block-size motion compensation with small block sizes

H.264/MPEG-4 AVC supports more selection of motion compensation block sizes than any prior standard, with a minimum luma motion compensation block size as small as 4 × 4. Segmentations of the macroblock for motion compensation are shown in Fig. 12. In this figure, the top row illustrates segmentation of macroblocks while the bottom row illustrates segmentation of 8 × 8 block partitions. The associated chroma block sizes are given in Table 2.

3.2.3. Motion vectors

Most prior standards enable half-sample motion vector accuracy. H.264/MPEG-4 AVC improves up on this by using quarter-sample motion vector accuracy for luma. In the compressed stream, only differential motion vector is coded, which is the difference between the motion vector of the block and the predictive motion vector (PMV). The PMV is usually a median value of the motion vectors of the surrounding blocks. A chroma motion vector is derived from the corresponding luma motion vector. Since the accuracy of luma motion vectors is one-quarter pixel and chroma has half resolution compared to luma, the accuracy of chroma motion vectors is one-eighth pixel.

3.2.4. Skipped and direct motion prediction modes

In prior standards, a "skipped" MB of a predictive picture could not signal motion in the scene content and thus implied a zero motion, no coded prediction error (transform coefficients) residual. Thus, a co-located macroblock from previous reference picture was simply copied to reconstruct the current skipped MB. This however meant that there was still substantial motion vector overhead when coding video containing global motion. H.264/MPEG-4 AVC addresses this by an improved design that instead infers motion in "skipped" MBs. As in earlier standards, a skipped MB in a P-slice does not send explicit

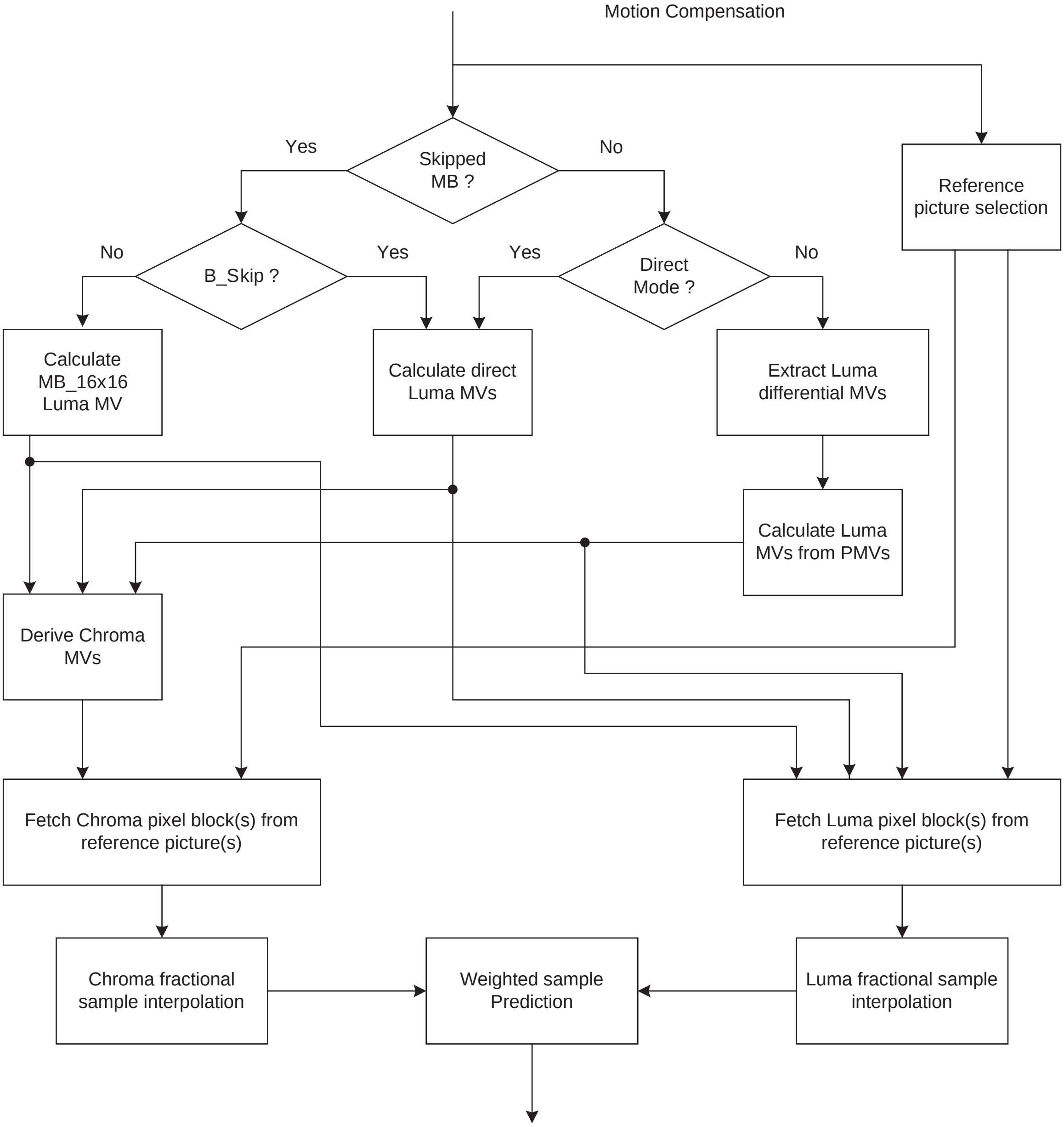
Motion Compensation
Skipped MB ?
Yes
No
B_Skip ?
No
Yes
Direct Mode ?
Yes
No
Reference picture selection
Calculate MB_16x16 Luma MV
Calculate direct Luma MVs
Extract Luma differential MVs
Calculate Luma MVs from PMVs
Derive Chroma MVs
Fetch Chroma pixel block(s) from reference picture(s)
Fetch Luma pixel block(s) from reference picture(s)
Chroma fractional sample interpolation
Weighted sample Prediction
Luma fractional sample interpolation

Fig. 11. The motion-compensation process.

motion vectors and any coded prediction error, but unlike previous standards, a skipped macroblock uses a 16×16 block prediction motion vector to copy a motion-compensated block rather than assume zero motion for such a block (and simply copy a co-located block). Further in H.264/MPEG-4 AVC, a skipped MB in a B-slice is defined as having no coded prediction error but uses "direct" mode motion vectors of 16×16 or 4, 8×8 blocks, depending on the coding of the co-located MB (as in Fig. 13) for motion compensated prediction. The motion-compensation process for skipped MBs is also illustrated in Fig. 11.

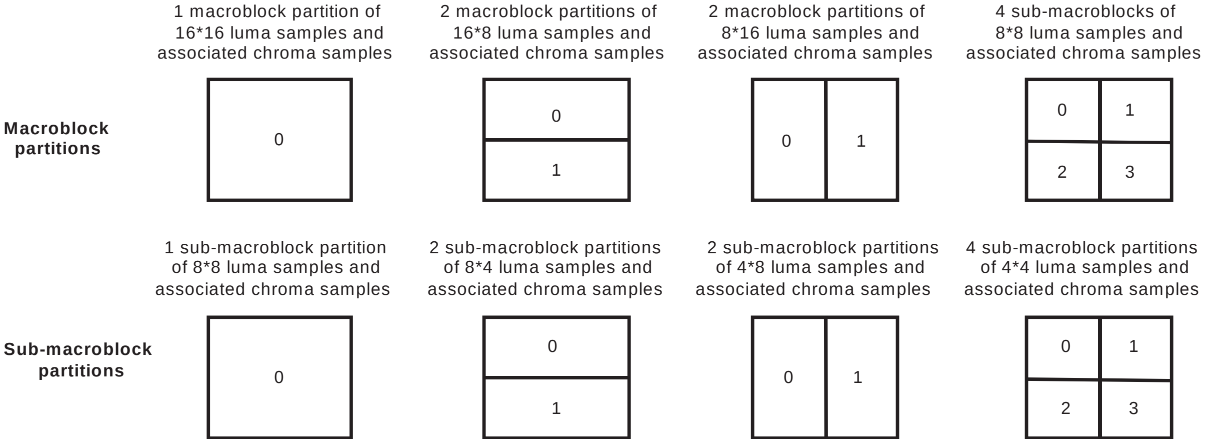
1 macroblock partition of 16*16 luma samples and associated chroma samples
2 macroblock partitions of 16*8 luma samples and associated chroma samples
2 macroblock partitions of 8*16 luma samples and associated chroma samples
4 sub-macroblocks of 8*8 luma samples and associated chroma samples
Macroblock partitions
0
0
1
0
1
0
1
2
3
1 sub-macroblock partition of 8*8 luma samples and associated chroma samples
2 sub-macroblock partitions of 8*4 luma samples and associated chroma samples
2 sub-macroblock partitions of 4*8 luma samples and associated chroma samples
4 sub-macroblock partitions of 4*4 luma samples and associated chroma samples
Sub-macroblock partitions
0
0
1
0
1
0
1
2
3

Fig. 12. Partitioning of a MB for motion compensation.

Table 2
Chroma block sizes associated with luminance partitions

Block size	Luma	16 × 16	16 × 8	8 × 16	8 × 8	8 × 4	4 × 8	4 × 4
(full pixel)	Chroma	8 × 8	8 × 4	4 × 8	4 × 4	4 × 2	2 × 4	2 × 2

The “direct” prediction motion reference design used here is an enhancement of “direct” prediction motion reference design [29,34,33] found in MPEG-4 Visual (part 2) standard. In particular, H.264/MPEG-4 AVC includes for B-slices, two types of direct motion compensation—(a new) spatial direct mode, and a simplified (version of MPEG-4’s original) temporal direct mode.

In spatial direct mode, the motion vectors are derived by examining the motion vectors of a co-located MB without the scaling process, and using motion vectors of neighboring blocks as used for generating prediction motion vectors. The detailed rules for generating motion vectors can be found in [14].

Fig. 13 illustrates the derivation of temporal direct-mode motion vectors when the current picture is temporally between the reference picture 1 and the reference picture 2, where mvCol is the motion vector of co-located partition in the reference picture 2 and mvL0 and mvL1 are the derived motion vectors for a direct mode B-partition in the current B-picture. td and tb are time units between two reference pictures and between the reference picture 1 and the current B-picture, respectively.

3.2.5. *Luminance fractional sample interpolation*

The accuracy of motion compensation is in units of one-quarter of the distance between luma pixels. In case the motion vector points to an integer-pixel position, the prediction signal consists of the corresponding pixels of the reference picture; otherwise the corresponding pixel is obtained using interpolation to generate fractional pixel positions. The prediction values at half-pixel positions are obtained by applying a 2-D FIR filter. Prediction values at quarter-sample positions are generated by a bilinear filter, i.e. averaging samples at integer- and half-sample positions.

In order to reduce the impact of aliasing on the motion-compensation, a separable 2-D Wiener filter is specified to reduce drift due to aliasing for multi-resolution hybrid video coding. Such filter is applied to attenuate aliasing in the prediction signal for single resolution hybrid video coding with displacement vector resolutions of $\frac{1}{4}$

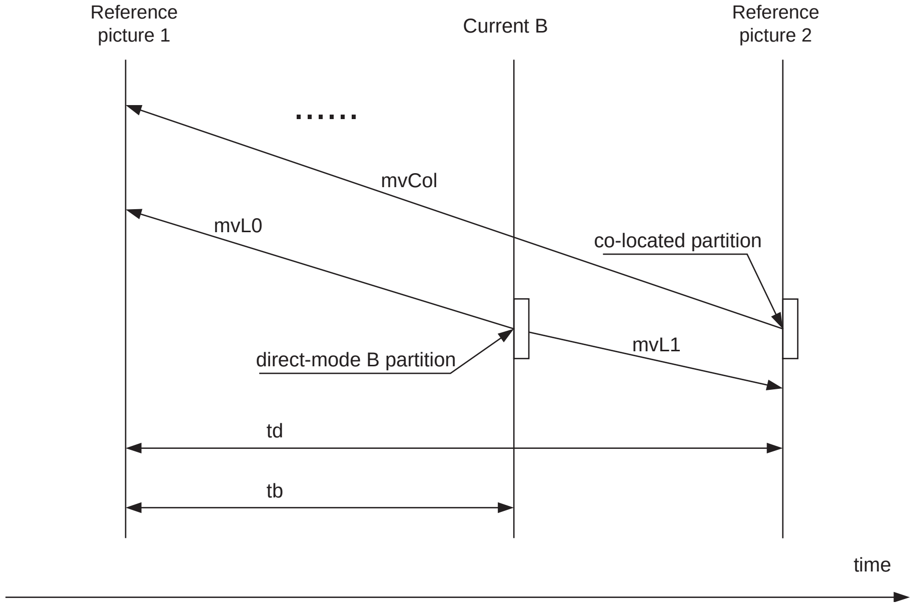

Fig. 13. Example of temporal direct-mode motion vector inference.

pel. The coefficients of such filter is given by (1, −5, 20, 20, −5, 1)/32.

Fig. 14 shows the full pixel position, half-pixel position, and quarter-pixel position and illustrates the fractional sample interpolation for half-pixel and quarter pixel. For example, the half-pixel values $H3$ and $H0$ are computed as

$$H3 = \frac{(F1 - 5F2 + 20F3 + 20F4 - 5F5 + F6 + 16)}{32},$$

$$H0 = \frac{(H1 - 5H2 + 20H3 + 20H4 - 5H5 + H6 + 16)}{32}.$$

while the quarter pixel values a and b are computed as $a = (F3 + H3 + 1/2)$, i.e. averaging with upward rounding of the two nearest samples at integer and half sample positions, and $b = (H3 + H7 + 1/2)$, i.e. averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction.

3.2.6. Chrominance fractional sample interpolation

The prediction values for the chroma component are always obtained by bilinear interpolation. Since the sampling grid of chroma has lower resolution than the sampling grid of the luma, the displacements used for chroma have one-eighth sample position accuracy. That is, for the given the chroma samples A, B, C, and D at full-sample locations, the predicted chroma sample value Chroma (x, y) is derived as follows:

$$\mathrm{Chroma}(x,y) = \frac{((8-x)(8-y)A + x(8-y)B + (8-x)y\,C + xy\,D + 32)}{64},$$

where $(x, y) \in \{(0,0),(0,1),\ldots,(3,2),(3,3)\}$, indicate various full-pel, half-pel and quarter-pel luma positions (Fig. 15).

3.2.7. Weighted sample prediction

This standard allows the motion-compensated prediction signal to be weighted and offset by

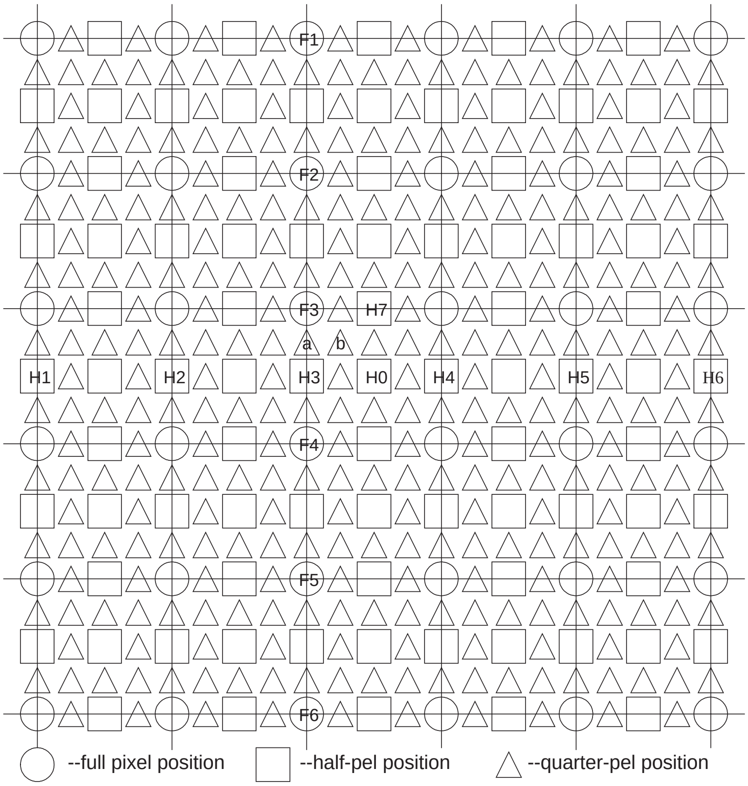

Fig. 14. Half pel and quarter pel interpolation.

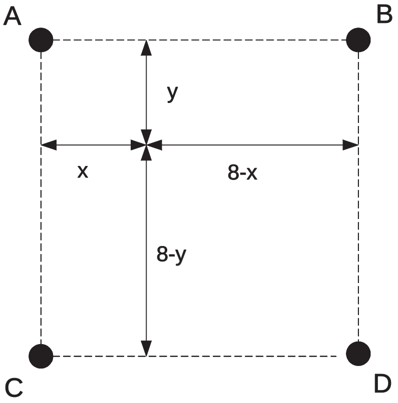

Fig. 15. Fractional sample position-dependent variables in chroma interpolation and surrounding integer position samples *A*, *B*, *C*, and *D*.

amounts specified by the encoder. This can significantly improve coding efficiency for scenes containing fades, and can be used flexibly for other purposes as well.

When two predictions are combined to create a common predicted reference pixel set, the weighted sample prediction is activated in its full functional form. The same weight sample prediction is applicable to both luma and chroma components. The operations performed are shown in Table 3.

3.3. Intra and MC modes supported

I-slices comprise of only intra macroblocks. Intra macroblocks can be coded using either 4 × 4 prediction or 16 × 16 prediction. In either case, the prediction is performed in spatial (pixel) domain. In case of 4 × 4 prediction, per 4 × 4 block, 9 possibilities for prediction exist (8 prediction directions and a dc value). In case of 16 × 16 prediction, 4 possibilities for prediction exist (3 prediction directions, and a dc value). The total number of macroblock types is 26 and is identified by mb_type value in range of 0–25.

Table 3
Operations in weighted prediction

*Prediction from reference P*0	
LWD < 1	Min (Max(0, $(P_0W_0+O_0)$), 255)
LWD ≥ 1	Min (Max(0, $(((P_0W_0+2^{\mathrm{LWD}-1})\gg \mathrm{LWD})+O_0)$), 255)
*Prediction from reference P*1	
LWD < 1	Min (Max(0, $(P_1W_1+O_1)$), 255)
LWD ≥ 1	Min (Max(0, $(((P_1W_1+2^{\mathrm{LWD}-1})\gg \mathrm{LWD})+O_1)$), 255)
Bi-directional prediction	Min (Max(0, $(((P_0W_0+P_1W_1+2^{\mathrm{LWD}})\gg(\mathrm{LWD}+1))+((O_0+O_1+1)\gg 1))))$, 255)

where, P_0 and P_1 are the predicted pixels from two references (8-bit integer, ranged between 0 and 255), W_0 and W_1 are the weighting scale factors ranged between −256 and +255, O_0 and O_1 are the offsets ranged between −256 and +255, and LWD is a normalization factor ranged between 0 and +7. Operations for the default weighted sample prediction are given by LWD = 0, $W_0=1$, $O_0=0$, $W_1=1$, $O_1=0$.

Table 4
Macroblock types in I slices (total types = 26 intra)

mb_type	Name of mb_type	16 × 16PredMode	CodedBlockPatternC	CodedBlockPatternY
0	I_4 × 4	—	—	—
$(i+1)+4(j+3(k/15))$	I_16 × 16_*i*_*j*_*k*	i = 0,1,2,3 (inner loop)	j = 0,1,2 (middle loop)	k = 0,15 (outer loop)
25	I_PCM	—	—	—

Only 1 mb_type is actually associated with 4 × 4 prediction with coding modes specified further for pairs of 4 × 4, while 24 types are associated with 16 × 16 block, and 1 type is reserved for PCM coding. The 24 types for 16 × 16 result due to combinations formed by of 4 types of prediction, 3 value for codedblockpatternchroma, and 2 values for codedblockpatternluma. The three values of codedblockpatternchroma are: 0 (no chroma coefficients), 1 (non-zero 2 × 2 coefficients and all chroma ac coefficients zero), and 2 (maybe 2 × 2 non-zero coefficients and at least one non-zero chroma ac coefficient). The two values of codedblockpatternluma are: 0 (no ac coefficient in 16 × 16 block), and 1 (at least 1 ac coefficient in 16 × 16 block). The entire set of 26 mb_type for I-slices is shown in Table 4 in a compact form.

Macroblocks in P-slices can be coded in inter or intra modes. When a macroblock is coded in inter mode, five explicit and one inferred mb_type can be used. Macroblocks of P-slices when coded in inter mode, can use frames from prediction list L0 for selecting references for prediction. The five mb_types of inter mode correspond to MC with 16 × 16, 16 × 8, 8 × 16, 8 × 8, and 8 × 8 with previous reference only, based prediction and the sixth mb_type is Skip (for which no motion vectors or transform coefficients are sent). The set of six mb_type for P-slices is shown in Table 5.

The inter modes using 8 × 8 MC prediction in addition requires specification of submacroblock types indicating MC prediction of 8 × 8, 8 × 4, 4 × 8 and 4 × 4 type. This set of 4 sub_mb_type for P-slices is shown in Table 6.

In B-slices, macroblocks can take one of 24 possible inter modes or may be coded in intra mode. Inter macroblocks of B-slices can be coded with uni-prediction and may use either list L0, or list L1, or may be coded with bi-prediction using both lists L0 and L1. In fact, whether uni- or bi-prediction is used is decided not only on macroblock basis but in fact on a partition basis, allowing even the possibility of coding one of the partition with uni-prediction using one of the two lists L0 or L1, while the other partition may be coded with bi-prediction. One of the inter modes is called direct prediction (also in MPEG-4) and

Table 5
Macroblock types 0–4 P slices (total types = 5 inter + 1 implicit + 26 intra)

mb_type	Name of mb_type	NumMbPart	Pred List for MbPart	MbPart width × height
0	P_L0_16 × 16	1	PredL0 ; —	16 × 16
1	P_L0_L0_16 × 8	2	PredL0 ; PredL0	16 × 8
2	P_L0_L0_8 × 16	2	PredL0 ; PredL0	8 × 16
3	P_8 × 8	4	— ; —	8 × 8
4	P_8 × 8ref0	4	— ; —	8 × 8
Inferred	P_Skip	1	PredL0 ; —	16 × 16

Table 6
Sub-macroblock types in P slices (total types = 4 inter)

sub_mb_type	Name of sub_mb_type	NumSubMbPart	Pred List for SubMbPart	SubMbPart width × height
0	P_L0_8 × 8	1	PredL0	8 × 8
1	P_L0_8 × 4	2	PredL0	8 × 4
2	P_L0_4 × 8	2	PredL0	4 × 8
3	P_L0_4 × 4	4	PredL0	4 × 4

Table 7
Macroblock types 0–22 in B slices (total types = 23 inter + 1 implicit + 26 intra)

mb_type	Name of mb_type	NumMbPart	Pred List for MbPart	MbPart width × height
0	B_16 × 16	1	Direct;—	16 × 16
$1+i$; $i=0,1$	B_Li_16 × 16	1	PredLi ; —	16 × 16
3	B_BI_16 × 16	1	PredBI ; —	16 × 16
$4+(2\times i)$; $i=0,1$	B_Li_Li_16 × 8	2	PredLi ; PredLi	16 × 8
$5+(2\times i)$; $i=0,1$	B_Li_Li_8 × 16	2	PredLi ; PredLi	8 × 16
$8+(2\times i)$; $i=0,1$	B_Li_Lj_16 × 8	2	PredLi ; PredLj (j = (i + 1)%2)	16 × 8
$9+(2\times i)$; $i=0,1$	B_Li_Lj_8 × 16	2	PredLi ; PredLj (j = (i + 1)%2)	8 × 16
$12+(2\times i)$; $i=0,1$	B_Li_BI_16 × 8	2	PredLi ; PredBI	16 × 8
$13+(2\times i)$; $i=0,1$	B_Li_BI_8 × 16	2	PredLi ; PredBI	8 × 16
$16+(2\times i)$; $i=0,1$	B_BI_Li_16 × 8	2	PredBI ; PredLi	16 × 8
$17+(2\times i)$; $i=0,1$	B_BI_Li_8 × 16	2	PredBI ; PredLi	8 × 16
20	B_BI_16 × 8	2	PredBI ; PredBI	16 × 8
21	B_BI_8 × 16	2	PredBI ; PredBI	8 × 16
22	B_8 × 8	4	— ; —	8 × 8
Inferred	B_Skip	—	Direct ; —	8 × 8

involves use of implicit (spatially adjacent or temporally colocated MB) scaled motion vectors; the two types of direct modes are called temporal direct and spatial direct, however this choice is allowed on slice rather than macroblock basis. The set of 24 mb_type for B-slices is shown compactly in Table 7.

As in the case of P-slices, an inter mode macroblock may also be coded in 8 × 8 mode. This in addition requires specification of

Table 8
Sub-macroblock types in B slices (total types = 13 inter)

sub_mb_type	Name of sub_mb_type	NumSubMbPart	Pred List for SubMbPart	SubMbPart width × height
0	B_Direct_8 × 8	—	Direct	4 × 4
$1+i$; $i=0,1$	B_Li_8 × 8	2	PredLi	8 × 8
3	B_BI_8 × 8	2	PredBI	8 × 8
$4+(2\times i)$; i = 0,1	B_Li_8 × 4	2	PredLi	8 × 8
5 + (2 × i); i = 0,1	B_Li_4 × 8	2	PredLi	4 × 8
8	B_BI_8 × 4	2	PredBI	8 × 4
9	B_BI_4 × 8	2	PredBI	4 × 8
$10+i$; i = 0,1	B_Li_4 × 4	4	PredLi	4 × 4
12	B_BI_4 × 4	4	PredBI	4 × 4

Table 9
Comparison of basis matrices of WHT, Slant, and DCT transforms with that used by H.264

(a) 4 × 4 WHT				*(b) 4 × 4 Slant*			
0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
0.5000	0.5000	−0.5000	−0.5000	0.6708	0.2236	−0.2236	−0.6708
0.5000	−0.5000	−0.5000	0.5000	0.5000	−0.5000	−0.5000	0.5000
0.5000	−0.5000	0.5000	−0.5000	0.2236	−0.6708	0.6708	−0.2236
(c) 4 × 4 HCT (used by H.264)				*(d) 4 × 4 DCT*			
0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
0.6324	0.3162	−0.3162	−0.6324	0.6565	0.2727	−0.2727	−0.6565
0.5000	−0.5000	−0.5000	0.5000	0.5000	−0.5000	−0.5000	0.5000
0.3162	−0.6324	0.6324	−0.3162	0.2727	−0.6565	0.6565	−0.2727

submacroblock types indicating MC prediction mode for submacroblock.(8 × 8 block). The set of 13 sub_mb_type for B-slices is shown in Table 8.

4. Transform, quantization, and loop filter

4.1. Basics of transform selected

Unlike earlier standards, H.264/AVC uses 4 × 4 transform block size instead of 8 × 8 block size for coding. Further, it does not use the DCT transform but uses a simplified integer approximation. The simplified transform in conjunction with combining of transform scale factors with quantization process allows all transform operations to be conducted using 16-bit arithmetic. Further, since transform basis matrix coefficients are all integers, its implementation at the decoder can be exact eliminating problems due to potential encoder/decoder accuracy mismatch as maybe the case with DCT. To allow ease of comparison with other well known transforms, we have included transform scaling factors in the basis matrices in Table 9 which shows not only the 4 × 4 transform employed by H.264 (known in literature as the high correlation transform (HCT)) [1,2,5,7] but also other closely related 4 × 4 transforms [5] such as the Walsh Hadammard transform (WHT), the Slant transform, and the DCT transform. As can be seen the HCT transform represents an approximation of the DCT transform similar to the Slant transform; if the scale factors are removed, the HCT transform is however simpler to implement as its coefficients are simply 1's and 2's.

Regarding the transform used, two issues of interest are: the block size, and the transform itself. In general, larger the block size used for transform, better it would be for exploiting global correlations, on the other hand, the smaller the block size used for the transform, better it would be for exploiting local adaptivity in content. In addition, smaller block sizes are beneficial for reducing implementation complexity. Thus, the 8×8 transform block size chosen by earlier standards reflects a tradeoff of conflicting requirements. Further, the transform block size should either match motion compensation block size or at the very least be no larger than the size of the smallest motion compensation block size (8×8 in H.263 and MPEG-4); if this condition is not met, spurious block edge artifacts from motion compensation would increase transform coding cost. If transform block size is reduced, normally additional inefficiencies are introduced in entropy coding as frequency of end-of-block symbols that mark end of significant coefficients along a scan path, increase, costing extra overhead bits. Since H.264/MPEG-4 AVC relies on many block sizes for prediction the smallest being 4×4, and further since the goal is to keep decoding complexity low, it uses a 4×4 transform. Of course, in order for this transform block size to be efficient in compression efficiency, more efficient ways of representing end of block implicitly had to be devised. Further, H.264 does not rely heavily on transform for decorrelation so 4×4 block size works acceptably. However, use of 4×4 block transform does require use of loop filter to reduce the appearance of grid structure at low bit-rates.

For clarity, we rewrite the basis matrix of [HCT] of 4×4 HCT transform of Table 9(c) taking out multipliers of each row as follows:

$$[HCT]=\begin{pmatrix}1/2\\1/\sqrt{10}\\1/2\\1/\sqrt{10}\end{pmatrix}\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}.$$

As a reminder, the 2D forward transform process using aforementioned basis matrix assuming $[S]$ is a 4×4 block of residue pixels to be transformed and $[C]$ is the 4×4 block of resulting coefficients, can be written as follows:

$$[C]=[\mathrm{HCT}][S][\mathrm{HCT}]^{\mathrm{T}}$$

Further, all scale factors resulting from this operation can be absorbed in the quantization process to integer accuracy.

4.2. Scan, transform and quantization

As explained earlier H.264/MPEG-4 AVC uses a 4×4 transform for coding of residue blocks. In case of intra coding, the transform operation is applied in two stages. The transform itself is often referred to as an “integer” transform. The term “integer” refers to the fact that the multiplication constants are specified as integers, and implementations are required to follow the exact values specified, unlike the 8×8 DCT used in prior coding standards. The primary transform is similar to a DCT with the terms quantized to fit into small integers, but the order is modified somewhat. The details were derived following the motivation to achieve an exactly invertible combination of transform and quantization, while limiting the arithmetic accuracy to 16-bit operations.

The second stage transforms are applied to the DC components of the first stage transform when 16×16 intra prediction is employed. The DC level transform is 4×4 for luma, and 2×2 for chroma due to the 4:2:0 sampling. The DC transforms are Hadammard transforms in both cases.

H.264/MPEG-4 AVC due to use of small block-sizes is effectively able to reduce artifacts known as “ringing”.

4.2.1. Scan

In MPEG-2 video, the 8×8 DCT coefficients can be scanned by the zig–zag scan to generate run-level events that are VLC coded. The zigzag scan works well on average and can be looked upon as a combination of three types of scans, a horizontal type of scan, a vertical type of scan and a diagonal type of scan. Often in natural images, on a block basis, a predominant preferred direction for scanning exists depending on the orientation of significant coefficients; MPEG-4 part-2 video uses adaptive scanning to exploit this property. In addition to the zig–zag scan, another scanning direction, called the alternate scan, was also specified in MPEG-2 for more efficient scanning of coefficients to produce (run, level) events that can be efficiently entropy coded as compared to scanning by zig–zag scan alone. The alternate scan is often used in MPEG-2 for block scanning of DCT coefficients of interlaced video.

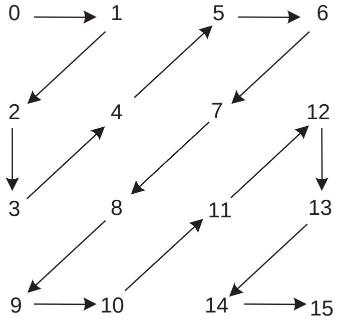

Fig. 16. Zig–zag scan used to scan frame macroblocks.

However in H.264/MPEG-4 AVC fixed scanning is employed. The 4×4 zigzag scan given in Fig. 16 appears to be fairly efficient for coding of coefficients of frame-based 4×4 transform, while the field scan (discussed in Section 4.3) is efficient for coding of coefficients of field-based 4×4 transform.

For the 2×2 transform for Chroma DC, the coefficients are scanned in the raster order.

4.2.2. Inverse transform operation for intra dc with 16×16 prediction

Inverse luma DC transform is a 4×4 Hadammard transform specified by

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}.$$

Inputs to this transform are coefficient level values for luma DC transform coefficients of Intra_16×16 macroblocks as a 4×4 array c of elements c_{ij}, where i and j form a 2-D frequency index. Outputs of this process are 16 scaled DC values for luma 4×4 blocks of Intra_16×16 macroblocks as a 4×4 array dcY of elements dcY_{ij}.

Inverse chroma DC transform is a 2×2 Hadammard transform specified by

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{00} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Inputs to this transform are coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as a 2×2 array c of elements c_{ij}, where i and j form a 2-D frequency index. Here, 'i' refers to the row index within the matrix and 'j' refers to the column index within the matrix. Outputs of this process are four scaled DC values as a 2×2 array dcC of elements dcC_{ij}.

4.2.3. Inverse quantization operation

Inverse quantization is performed by using quantization parameter (QP) specified for each MB in H.264/MPEG-4 AVC standard. $\mathrm{QP_C}$ is the QP that is used for quantizing chroma blocks (and sub blocks) while QP_Y is the QP that is used for quantizing luma blocks (and sub blocks). Differential values of QP_Y is specified for each coded MB in its header. The values of QP_i are derived from QP_Y and chroma_qp_index_offset. "chroma_qp_index_offset" is a parameter present in the "picture parameter set" layer of the bit stream.

$$\mathrm{QP}_i = \mathrm{Min}(\mathrm{Max}(0, \mathrm{QP}_y + \text{chroma\_qp\_index\_offset}), 51).$$

And QP_i is used to calculate $\mathrm{QP_C}$ as shown in the Fig. 17. Note that $\mathrm{QP}_i = \mathrm{QP_C}$ for $\mathrm{QP}_i < 30$.

The function LevelScale(m, i, j) is specified as follows:

$$\mathrm{LevelScale}(m, i, j) = \begin{cases} V_{m0} \text{ for } (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ V_{m1} \text{ for } (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ V_{m2} \text{ otherwise}, \end{cases} \quad (1)$$

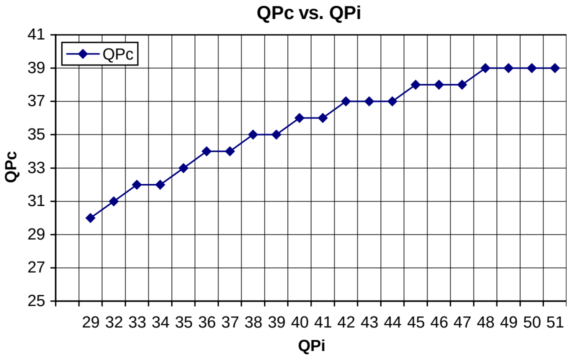

Fig. 17. Relation between $\mathrm{QP_C}$ and QP_i.

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{2}$$

As used in Eq. (1), 'm' is the row index of the matrix, v, and v is defined in Eq. (2). The function LevelScale (m, i, j) defined above is used in the Inverse scaling procedure as described below.

There are three types of inverse quantization procedure that are used for different block types.

- luma DC block,
- chroma DC block,
- all other chroma/luma AC-only and AC+DC sub blocks.

In all these equations, i and j form a 2-D frequency index for coefficients within each sub block.

Inverse luma DC quantization is performed according to the following:

- If QP_Y is greater than or equal to 12, the scaled result is derived as

$$\mathrm{dcY}_{ij} = (f_{ij}\ \mathrm{LevelScale}\,(\mathrm{QP}_Y\%6, 0, 0)) \ll (\mathrm{QP}_Y/6 - 2), i,j = 0, \ldots, 3.$$

- Otherwise, i.e. $\mathrm{QP}_Y < 12$, the scaled results are derived as

$$\mathrm{dcY}_{ij} = (f_{ij}\ \mathrm{LevelScale}\,(\mathrm{QP}_Y\%6, 0, 0) + 2^{1-\mathrm{QP}/6}) \gg (2 - \mathrm{QP_y}/6),\ i,j = 0, \ldots, 3.$$

The output of this operation is the matrix dcY.

Inverse chroma DC quantization is performed according to the following:

- If $\mathrm{QP_C}$ is greater than or equal to 6, the scaling result is derived as

$$\mathrm{dcC}_{ij} = (f_{ij}\ \mathrm{LevelScale}\,(\mathrm{QP_C}\%6, 0, 0)) \ll (\mathrm{QP_C}/6 - 1),\ i,j = 0, 1.$$

- Otherwise ($\mathrm{QP_C}$ is less than 6), the scaling results are derived by

$$\mathrm{dcC}_{ij} = (f_{ij}\ \mathrm{LevelScale}\,(\mathrm{QP_C}\%6, 0, 0)) \gg 1, \quad i,j = 0, 1.$$

The output of this operation is the matrix dcC.

Inverse AC quantization is performed according to the following equation:

$$w_{ij} = (c_{ij}\ \mathrm{LevelScale}\,(\mathrm{QP}\%6, i,j)) \ll (\mathrm{QP}/6) \quad \text{with } i,j = 0, \ldots, 3.$$

The output of this operation is the matrix w.

It can be seen from the above formula that the computation QP%6 and QP/6 require the division operations. However, those divisions by 6 can be avoided, if so desired, by noting that $\mathrm{QP}/6 = (43 \times \mathrm{QP}) \gg 8$ for all QP in [0,…,130].

4.2.4. Coefficient assembly and general inverse transform operation

Merging of DC-only and AC-only sub-blocks is performed on luma blocks in Intra_16 × 16 macroblocks and on chroma blocks within a macro

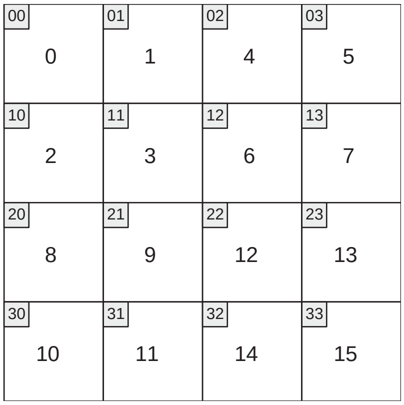

Fig. 18. Assignment of the indices of dcY to 4 × 4 luma blocks.

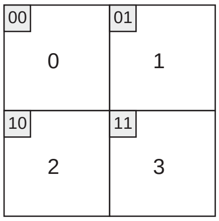

Fig. 19. Assignment of the indices of dcC to the 4 × 4 chroma block.

block. Essentially, the DC coefficients from DC-only sub blocks are inserted into specific locations in the AC-only sub blocks.

Luma DC-only block is referred to as dcY. Fig. 18 shows the assignment of the indices of the array dcY to the various luma AC-only sub blocks. The two numbers in the small squares refer to indices i and j corresponding to dcY$_{ij}$, and the numbers in large squares refer to luma4 × 4BlkIdx of the luma AC-only sub blocks.

Chroma DC-only block is referred to as dcC. Fig. 19 shows the assignment of the indices of the array dcC to the various chroma AC-only sub blocks. The two numbers in the small squares refer to indices i and j in dcC$_{ij}$, and the numbers in large squares refer to chroma4 × 4BlkIdx of the chroma AC-only sub blocks.

The following 4 × 4 transform is used to convert the block of quantized coefficients to a block of output samples:

$$\begin{bmatrix} x'_{00} & x'_{01} & x'_{02} & x'_{03} \\ x'_{10} & x'_{11} & x'_{12} & x'_{13} \\ x'_{20} & x'_{21} & x'_{22} & x'_{23} \\ x'_{30} & x'_{31} & x'_{32} & x'_{33} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \times \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

Finally, the inverse transformed coefficients are normalized with

$$x_{ij} = \left(x'_{ij} + 2^5\right) \gg 6$$

to obtain the residual sample values.

4.3. Interlace video coding tools

If video is captured in interlaced form then each frame is comprised of a pair of fields—an even field, and an odd field, such that there is a time delay of half the frame duration between these fields. As the fields are captured with a time delay, the visual information in the adjacent lines of the two fields tends to get less correlated, especially if objects in the scene are moving fast. Therefore, the regions or the frames that contain moving objects may be more efficiently compressed by compressing the two fields separately. In this standard, the decision of whether to compress the two fields separately or not can be made either at the frame

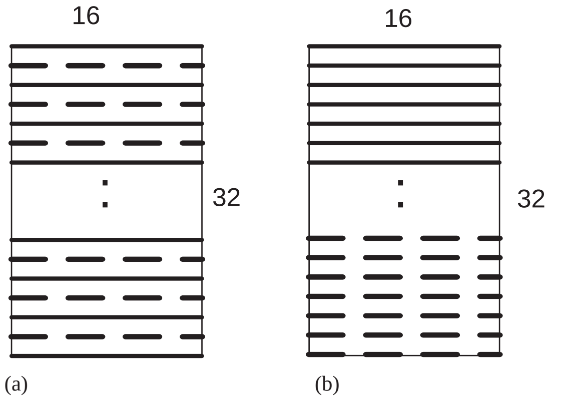

Fig. 20. Adaptive field frame (a) pair (16 × 32) of frame macroblocks, (b) pair (16 × 32) of field macroblocks. Solid lines belong to the even field and dotted lines belong to the odd fields.

level or at macroblock-pair level. When the decision is made at the frame level then entire frame is split into two fields and they are compressed separately. This is also referred as PicAFF (picture adaptive field frame) coding. When the decision is made at macroblock pair (16 × 32) level—then the two vertically aligned macroblocks can either be split into two and compressed as two separate fields or kept together and compressed as one single frame (see Fig. 20). When the decision is made at macroblock pair level, the coding type of the macroblock is referred to as MBAFF (MacroBlock Adaptive Field/Frame).

Notice that, unlike MPEG-2, in MBAFF the field/frame decision is made at macroblock pair level and not at each macroblock level. In this process, if a macroblock pair (16 × 32) is split into two fields it results in two 16 × 16 size macroblocks where each is compressed by using the tools available in normal coding (with appropriate modifications in tools like motion estimation/compensation, motion vector prediction, prediction of intra prediction modes, zig–zag scan, deblocking filter, and context modeling in CA-BAC). Motion can now be estimated between fields and frames. Therefore pixels and motion vectors need to be re-sized accordingly. Zig–zag scan (discussed earlier) is used when a macroblock is frame coded, and an alternate scan (see Fig. 21) more efficient for field coding is used when a macroblock is field coded. As an example, if MBAFF is selected and if the MB-pair is coded in frame mode, the zig–zag scan is used, and if a MB-pair is coded in field mode, then alternate scan is used.

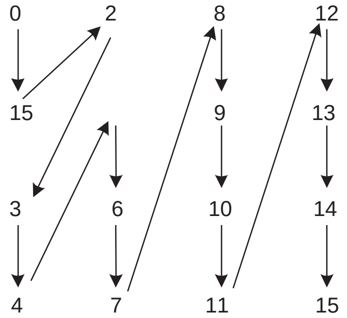

Fig. 21. Alternate scan used for scanning of field macroblocks.

As mentioned earlier, the decision of frame vs. field adaptation can also be done at frame level and is referred to as PicAFF. During the development of the standard it was reported that PicAFF provides significant gain of around 15% over frame coding for complex sequences with significant motion activity. In addition, it was also reported that MBAFF provides additional useful gain for scenes with mixed motion where some objects are stationary while others move.

4.4. Deblocking loop filter

In the H.264/MPEG-4 AVC video coding standard, there are two sources that can introduce blocking artifacts. The most significant one is the integer 4 × 4 transform in intra and inter frame prediction residue coding. Coarse quantization of the transform coefficients can cause visually disturbing discontinuities at the block boundaries. The second source of blocking artifacts is motion compensated prediction. Motion compensated blocks are generated by copying interpolated pixel data from different locations of possibly different reference frames. Since there is almost never a perfect fit for this data, discontinuities on the edges of the copied blocks of data typically arise. Additionally, in the copying process, existing edge discontinuities in reference frames are carried into the interior of the block to be compensated. Although the 4 × 4 transform size used in H.264/MPEG-4 AVC somewhat makes the artifact less visible, a deblocking filter is still a useful tool to enhance coding performance.

There are two main approaches in integrating deblocking filters into video codecs. Deblocking filters can be used either as post filters or loop filters. Post filters only operate on the display buffer outside of the coding loop, and thus are not a normative part of the standard. Because their use is optional, post-filters offer maximum freedom for decoder implementations. On other hand, loop filters operate within the coding loop. That is, the filtered frames are used as reference frames for motion compensation of subsequent coded frames. This forces a standard conformant decoder to perform filtering identical to that at the encoder in order to prevent drift.

In the post-filtering approach, the frame is typically decoded into a reference frame buffer and filtered prior to passing it to the display device; some implementation of this may require an additional frame buffer. In the loop-filtering approach, however, filtering maybe carried out for each MB during the decoding process, and the filtered output stored directly to the reference frame buffers.

Using the loop filtering has several advantages over post filtering as follows:

- The requirement of a loop filter ensures a certain level of quality. With a loop filter in the codec design, content providers can safely assume that the video is processed by proper deblocking filters, guaranteeing the quality level expected by the producer.
- There is no need for potentially an extra frame buffer at the decoder.
- Empirical tests have shown that loop filtering typically improves both objective and subjective quality of video streams. Quality improvements are mainly due to the fact that filtered reference frames offer higher quality prediction for motion compensation.

In the H.264/MPEG-4 AVC design, a conditional filtering can be applied to all 4 × 4 block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled explicitly by bitstream parameters. This filtering process can be performed on a MB basis, with all MBs in a picture processed in the order of increasing MB addresses. When MBAFF is disabled, this order is the same as the raster scan order of all the macroblocks across the whole picture. In MBAFF mode, the partition of the picture into MB-pairs and the process order are depicted in Fig. 22. The highlighted area in the picture shows one MB pair. The numbers and the zigzagged line with ending arrow indicate the scanning order of the MB pairs.

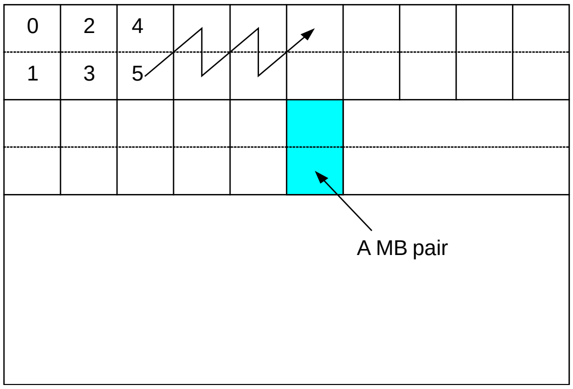

Fig. 22. Partitioning of decoded picture into MB pairs, and scanning order of MB pairs.

Prior to the operation of the deblocking filter process for each MB, the filtered samples of the MB or MB pair (when MB-AFF is active) above (if any) and the MB or MB pair to the left (if any) of the current MB must be available. The deblocking filter process is invoked for the luma and chroma components separately. For each MB, vertical edges are filtered first, from left to right, and then horizontal edges are filtered from top to bottom.

4.4.1. *Deblocking rules*

The luma deblocking filter process is performed on four 16-sample edges and the deblocking filter process for each chroma components is performed on two 8-sample edges, for the horizontal direction as shown on the left side of Fig. 23 and for the vertical direction as shown on the right side of the

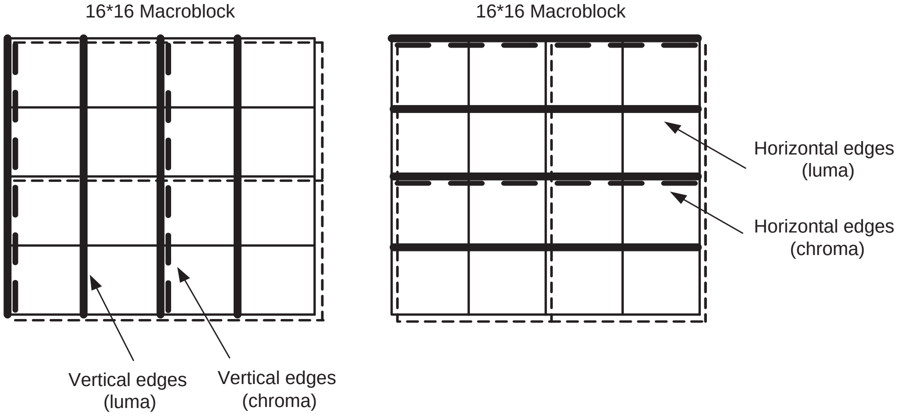

Fig. 23. Boundaries in a MB to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dashed lines).

same figure. During the filtering process, the following rules apply:

- Sample values above and to the left of the current MB that may have already been modified by the deblocking filter process operation on previous MBs shall be used as input to the deblocking filter process on the current MB and may be further modified during the filtering of the current MB.
- Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges for the same MB.
- Sample values modified during filtering of previous edges are used as input for the filtering of the next edge in both vertical and horizontal filtering directions.

Depending on the picture and MB format, the filtering can be done in either frame mode or field mode. In frame mode filtering, deblocking is performed on the frame samples. In field mode filtering, deblocking is performed on the field samples of the same field parity. For internal edges, where the pixels on both sides of the edge are all in the same MB, the filter mode naturally matches the format of the MB: For frame coded MB, frame mode filtering is used. For field coded MB, field mode filtering is used.

When MB-AFF is not used, all the MBs in the whole picture will be coded in the same format, frame or field. The deblocking filtering for the whole picture will be in the same mode, including both the internal edges and the edges on the border of two adjacent MBs.

When MB-AFF is active, the neighboring MBs can be coded in different formats. Different filtering modes may be applied for the same MB when filtering with its neighboring MBs.

For the vertical edge between two neighboring MBs (left MB edge), the following rules apply:

- If a frame MB adjacent to another frame MB, deblocking is performed on the frame samples, i.e. across the boundary between adjacent frame MBs as they appear in the frame structure.
- If a field MB pair adjacent to another field MB pair, deblocking is performed across the boundaries between field MBs of the same field polarity. Deblocking is never performed across the boundary between field MBs of opposite field polarities.
- If a frame MB pair with a field MB pair to the left of it, the lines of the field MB pair are logically reordered into frame order and deblocking is performed across the boundaries between the frame MBs in the frame MB pair and the result of the reordering from the field MB pair.
- If a field MB pair with a frame MB pair to the left of it, the lines of the frame MB pair are logically reordered into field order and deblocking is performed across the boundaries

between the field MBs in the field MB pair and the result of the reordering from the frame MB pair.

For the horizontal edge between two neighboring MBs (top MB edge), the following rules apply:

- If a frame MB adjacent to another frame MB, whether in the same MB pair or a different MB pair, deblocking is performed on the frame samples, i.e. across the boundary between adjacent frame MBs as they appear in the frame structure.
- If a field MB pair adjacent to another field MB pair, deblocking is performed across the boundaries between field MBs of the same field polarity. Deblocking is never performed across the boundary between field MBs of opposite field polarities.
- If a field MB pair with a frame MB pair above it, deblocking is performed in field mode across the boundary between the top field MB of the field MB pair and the top field samples in the lower frame MB of the frame MB pair above it, and then deblocking is performed in field mode across the boundary between the bottom field MB of the field MB pair and the bottom field samples in the lower frame MB of the frame MB pair above it.
- If a frame MB pair with a field MB pair above it, deblocking is performed in field mode across the boundary between the upper MB in the frame MB pair and the top field MB in the field MB pair above it, and then deblocking is performed in field mode across the boundary between the upper MB in the frame MB pair and the bottom field MB in the field MB pair. Deblocking is also performed in frame mode across the boundary between the upper and lower MBs in the frame MB pair.

Note that for each MB, 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a MB. When field mode filtering is applied to the top edges of a frame MB, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame MB and the above MB pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame MB. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular MB.

4.4.2. Filtering process

The filtering process is applied to a set of eight samples across a 4×4 block horizontal or vertical edge denoted as p_i and q_i with $i = 0, \ldots, 3$ as shown in Fig. 24 with the edge lying between $p0$ and $q0$.

Content-dependent boundary filtering strength: For each edge between neighbouring 4×4 luma blocks, a boundary strength (Bs) is derived based on the MB type, reference picture ID, motion vector and other MB coding parameters (see Fig. 25). For every edge between two 4×4 luminance sample blocks, a Bs parameter is assigned an integer value from 0 to 4. Fig. 25 shows how the value of Bs depends on the modes and coding conditions of the two adjacent blocks. Bs indicates the strength of the filtering performed on the edge including a selection between the three filtering modes. If Bs is zero, the filtering process is skipped for the current 4×4 block edge. When Bs equals to 4, a "strong filter" will be applied to the samples across the edge. In the standard mode of filtering which is applied for edges with Bs from 1 to 3, the value of Bs affects the maximum modification of the sample values that can be caused by filtering. The gradation of Bs reflects that the strongest blocking artifacts are mainly due to intra and prediction error coding and are to a somewhat smaller extent caused by block motion compensation.

In interlaced video, the deblocking filter in a field MB is applied to the pixels belonging to the

p3	p2	p1	p0	q0	q1	q2	q3

Fig. 24. Samples across 4×4 block horizontal or vertical boundaries used in filtering.

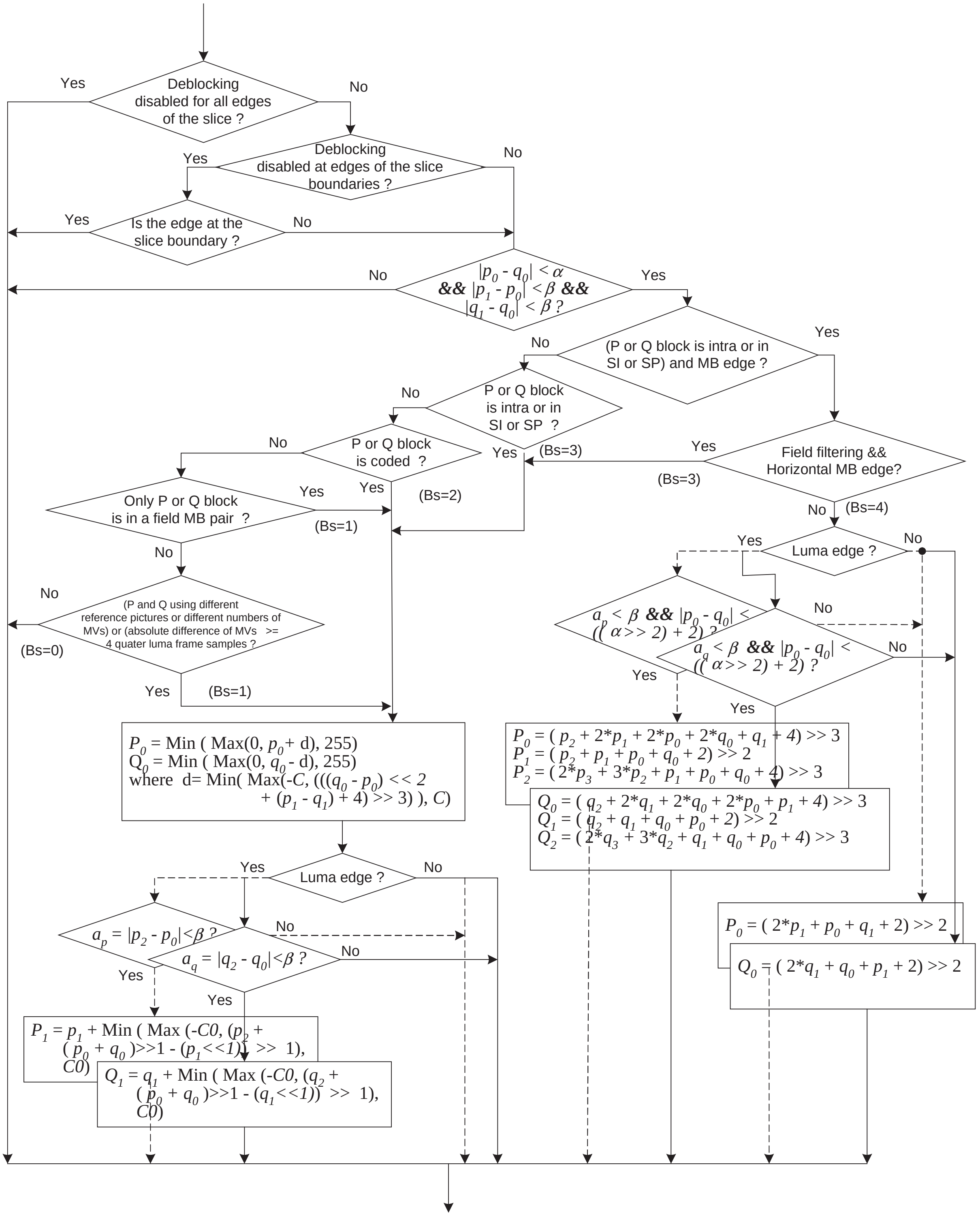
Deblocking disabled for all edges of the slice ?
Yes
No
Deblocking disabled at edges of the slice boundaries ?
Yes
No
Is the edge at the slice boundary ?
Yes
No
$|p_0 - q_0| < \alpha$ && $|p_1 - p_0| < \beta$ && $|q_1 - q_0| < \beta$?
No
Yes
(P or Q block is intra or in SI or SP) and MB edge ?
No
Yes
P or Q block is intra or in SI or SP ?
No
Yes
(Bs=3)
Field filtering && Horizontal MB edge?
Yes
(Bs=3)
No
(Bs=4)
P or Q block is coded ?
No
Yes
(Bs=2)
Only P or Q block is in a field MB pair ?
Yes
(Bs=1)
No
(P and Q using different reference pictures or different numbers of MVs) or (absolute difference of MVs >= 4 quater luma frame samples ?
No
(Bs=0)
Yes
(Bs=1)
P_0 = Min (Max(0, p_0+ d), 255)
Q_0 = Min (Max(0, q_0- d), 255)
where d= Min(Max(-C, (((q_0 - p_0) << 2 + (p_1 - q_1) + 4) >> 3)), C)
Luma edge ?
Yes
No
$a_p = |p_2 - p_0| < \beta$?
Yes
No
$a_q = |q_2 - q_0| < \beta$?
Yes
No
$P_1 = p_1$ + Min (Max (-$C0$, (p_2 + (p_0 + q_0)>>1 - (p_1<<1)) >> 1), $C0$)
$Q_1 = q_1$ + Min (Max (-$C0$, (q_2 + (p_0 + q_0)>>1 - (q_1<<1)) >> 1), $C0$)
Luma edge ?
Yes
No
$a_p < \beta$ && $|p_0 - q_0| <$ ((α>> 2) + 2) ?
Yes
No
$a_q < \beta$ && $|p_0 - q_0| <$ ((α>> 2) + 2) ?
Yes
No
P_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3
P_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2
P_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3
Q_0 = (q_2 + 2*q_1 + 2*q_0 + 2*p_0 + p_1 + 4) >> 3
Q_1 = (q_2 + q_1 + q_0 + p_0 + 2) >> 2
Q_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3
P_0 = (2*p_1 + p_0 + q_1 + 2) >> 2
Q_0 = (2*q_1 + q_0 + p_1 + 2) >> 2

Fig. 25. Deblocking Filter Decision Logic.

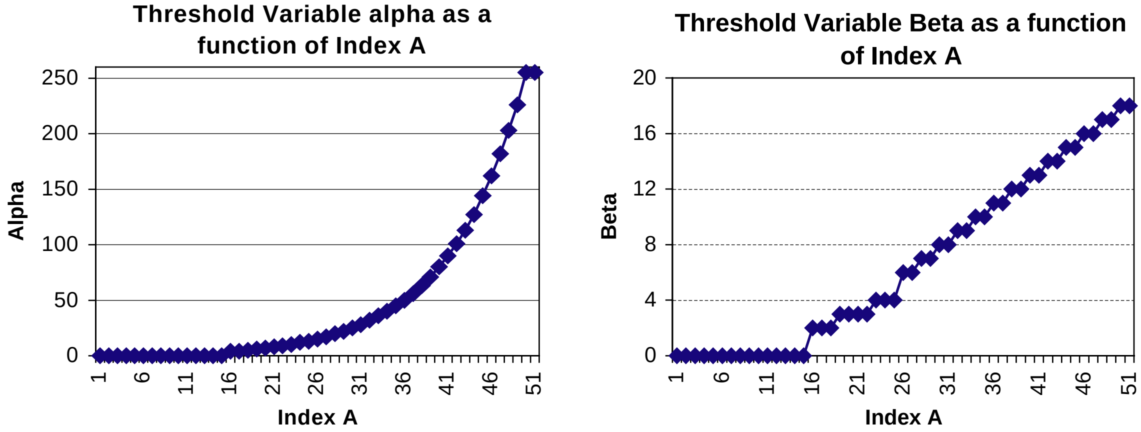

Fig. 26. Threshold variable α and β as a function of indexA.

same field. Strong filtering (Bs = 4) in not used vertically to filter horizontal macroblock edges in field macroblocks. Horizontal edges of the MBs are instead filtered using Bs = 3 if current or neighboring MB is compressed in Intra mode. This is because field lines are spatially twice as far apart as frame lines. Vertical edges of the MBs are filtered using strong filtering if current or neighboring MB is compressed in Intra mode. Additionally, if current and neighboring MB are in different, field or frame mode, then some filtering (Bs = 4, 3, 2, or 1) is always done and Bs = 0 is not used.

The Bs values for filtering of chrominance block edges are not calculated independently, but instead copied from the values calculated for their corresponding luminance edges.

Note that Fig. 25 only shows a logical flow of the deblocking filtering algorithm for simplicity. An actual implementation can be different. For example, the condition of $|q_0 - p_0| < \alpha \&\& |p_1 - q_0| < \beta \&\& |q_1 - p_0| < \beta$ at the top of the figure could indicate the need to access the pixel data before calculating the Bs values. The actual implementation can calculate Bs first and only access pixel values (to calculate this condition) if Bs > 0.

Threshold for each block edge: Even when the boundary strength is non-zero, the deblocking process may not be needed for a particular edge. This is especially true when there is a real sharp transition across the edge. Applying the deblocking process to such edge will result in blurry image. The blocking artifacts are most noticeable in very smooth region where the pixel values do not change much across the block edge. Therefore, in addition to the boundary strength, a filtering threshold based on the pixel values will be derived. This threshold will also be used to determine if deblocking process should be carried for the current edge.

The threshold variables α and β are specified in Fig. 26 depending on the values of indexA and indexB, where the variables indexA and indexB are derived based on the quantization parameter values for the MBs containing the samples p_0 and q_0. The values of α and β are defined approximately (i.e. tabulated values according to the following formula:

$$\alpha(y) = \frac{4}{5}\left(2^{y/6} - 1\right), \quad \beta(y) = \frac{y}{2} - 7.$$

- An adjusted threshold C is determined as follows.
  - ○ For luma block edges,

    $$C = C0 + ((\mathrm{a}_p < \beta)?1:0) + ((\mathrm{a}_q < \beta)?1:0).$$
  - ○ For chroma block edges,

    $$C = C0 + 1.$$

The threshold $C0$ is specified in Fig. 27 depending on the values of indexA and Bs.

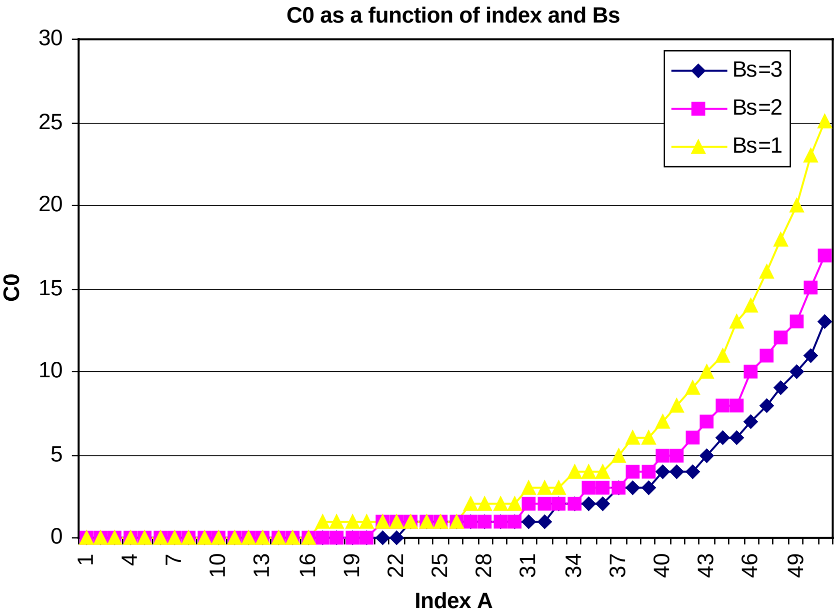

Fig. 27. Value of filter clipping variable $C0$ as a function of IndexA and Bs.

Details of the filtering process are illustrated in Fig. 25.

5. Entropy coding

As mentioned earlier, H.264/MPEG-4 AVC uses a number of techniques for entropy coding: Golomb codes, context adaptive variable word length coding (CAVLC), and context adaptive binary arithmetic coding (CABAC).

5.1. Golomb code and code mapping

The standard uses exponential Golomb (Exp-Golomb) codes to represent syntax elements of various types. Exp-Golomb codes follow the structure illustrated in Fig. 28(a) consisting of a prefix part (1, 01, 001, 0001,…) and a suffix part which is a string of bits (x_0, x_1x_0, $x_2x_1x_0$,…) where x_i are 0 or 1. The number of possible combinations of x_i strings reflect number of codes for a given prefix code. Each syntax element to be coded in the bitstream is assigned a type reflecting how data is to be mapped to codes such as unsigned exponential, ue(v), signed exponential, se(v), mapped exponential, me(v), or truncated exponential, te(v). Fig. 28(b) shows assignment of codewords to codeNum that are used for ue(v) codes. Fig. 28(c) shows assignment to signed values of a syntax element, codeNum, that can then be used with Fig. 28(b) to assign codewords.

There are yet other types of syntax elements such as for coded block pattern (CBP) where its values corresponds to a pattern number such as 47, 31,…,etc; this requires an explicit mapping of these values to codeNum followed by use of Table 5(b) to look up the actual codeword bitstring assignment. Such syntax elements are referred to as mapped exponential me(v) type.

5.2. CAVLC

For compressing quantized transform coefficients, an efficient VLC method called the context-based VLC (CAVLC) is employed. In this scheme, inter-symbol redundancies are exploited by switching VLC tables for various syntax elements depending on already transmitted coding symbols. The CAVLC method, however, may not be fully

Bitstring form	Range
1	0
$0\ 1\ x_1$	1-2
$0\ 0\ 1\ x_1\ x_0$	3-6
$0\ 0\ 0\ 1\ x_2\ x_1\ x_0$	7-14
$0\ 0\ 0\ 0\ 1\ x_3\ x_2\ x_1\ x_0$	15-30
$0\ 0\ 0\ 0\ 0\ 1\ x_4\ x_3\ x_2\ x_1\ x_0$	31-62
...	...

(a)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
..	..

(b)

codeNum	syntax element value
0	0
1	1
2	-1
3	2
4	-2
5	3
6	–3
7	4
8	–4
...	...

(c)

Fig. 28. (a) Prefix and suffix bitstrings, (b) exp-Golomb bitstrings, (c) mapping for signed exp-Golomb bitstrings.

able to adapt to actual conditional symbol statistics. Also, symbol probabilities that are greater than 0.5 are not handled efficiently. This may at times prevent usage of symbols with a smaller alphabet size for coding of residual data. On the positive side, with this method, since VLC tables for various syntax elements are switched depending on previous coded syntax elements, the increased adaptivity allows improved coding in comparison to schemes using a single VLC table.

Based on the syntax elements present in the bit stream, the run/level pairs within a block are obtained. In the CAVLC coded bit streams, the syntax elements within a coefficient-block (the coefficient levels, trailing 1 s and run of zeros) are present in the bit stream from the end of the block to the first.

In the CAVLC entropy-coding algorithm, the number of non-zero quantized coefficients (TotalCoeff) and the actual size and position of the coefficients are coded separately. After zig–zag (or alternate field) scanning of transform coefficients, their statistical distribution typically shows large values for the low-frequency part decreasing to small values later in the scan for the high-frequency part. An example for a typical zig–zag scan of quantized 4×4 transform coefficients could be given as follows:

$$8, 0, 0, 2, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0.$$

Based on this statistical behavior, the following data elements are used to convey information of quantized transform coefficients for a luma 4×4 block.

- Number of non-zero coefficients (TotalCoeff): In the above example, TotalCoeff $=4$.
- Trailing ones (Trailing_Ones): This indicates the number of (consecutive) non-zero coefficients with absolute value equal to 1 at the end of the scan. In the above example, Trailing_Ones $=2$.

The above two values (TotalCoeff, Trailing_Ones) are coded as a combined event, called **coeff_token**. One out of five VLC tables (nC entries) in H.264/MPEG-4 AVC is used based on the number of coefficients in neighboring blocks.

- Coding the value of coefficients: The values of the coefficients are coded. The Trailing_Ones need only sign specification (**trailing_ones_sign_flag**) since they are equal to either –1 or +1. Note that the statistics of coefficient values has less spread for the last non-zero coefficients than for the first ones. For this reason, coefficient values are coded in reverse scan order. In the examples above, 2 is the first coefficient value to be coded. The coefficients are coded as, so called, LevelCodes. The sign of a coefficient is represented as an even number (for a positive coefficient) or an odd number (for a negative coefficient) of the LevelCode. The LevelCode is encoded by using a VLC (**level_prefix**) table [14].

Positions of each non-zero coefficient are coded by specifying the positions of 0s before the last non-zero coefficient. It is split into two parts:

- **total_zeros**: This codeword specifies the number of zeros between the last non-zero coefficient of the scan and its start. In the above example the value of **total_zeros** (between –1 and 8) is 4. Since it is already known that TotalCoeff = 4, the number must be in the range between 0 and 12. A VLC table is available for in the **total_zeros** range between 1 and 15 for all TotalCoeff.
- **run_before**: This is the number of consecutive zero-valued transform coefficient levels in the reverse scan (starting from the last non-zero valued transform coefficient level, e.g. –1 in the above example). For each block, **run_before** specifies zero-runs before the last non-zero coefficient, i.e. the number of 0s before the last coefficient is coded. In the above example the first **run_before** number is 2 (between −1 and 1) and the next (also the last) **run_before** number is also 2 (between 2 and 8). A VLC table is also specified for coding **run_before** [14].

5.3. CABAC

For lossless entropy coding, arithmetic coding is known to achieve higher efficiency than VLC coding, albeit at the cost of higher complexity. Arithmetic coding achieves the coding efficiency benefits due to its effective use of probability models of occurrence of symbols, combined with the capability of coding a string of symbols with a single codeword. If the coding bits cost of the symbol string is mapped to individual symbols, they appear, in effect, to be coded with a high fractional accuracy in bits whereas VLC coding can only reach integer bits accuracy per symbol. This is particularly true for symbols having high probability (>0.5) of occurrence.

Further, adaptive arithmetic coding by virtue of adaptive modeling can relatively easily adapt to changing statistical characteristics of the data to be coded which is useful in video coding as it offers flexibility in dealing with larger variety of video content, and bit-rates. By comparison, adaptive VLC coding such as that using multiple VLC tables may still be limited in its ability to adapt to such conditions. Thus adaptive arithmetic coding may reduce the risk of mismatch to actual symbol statistics, performing more often close to entropy limits than possible with VLCs. This advantage however comes at the expense of added processing complexity that may require high precision in word length, probability estimation and update computation, and serial nature of many operations. A number of simplifications exist to reduce complexity, however, and the resulting coding efficiency performance may then depend on nature of simplifications.

5.3.1. CABAC basics

In this standard, an advanced method called context adaptive binary arithmetic coding, CABAC, is included for entropy coding of syntax element values. Encoding with CABAC [18] consists of three stages—binarization, context modeling and adaptive binary arithmetic coding. Fig. 29 shows a high level block diagram of CABAC encoder showing these various stages and their interdependence.

The syntax elements of H.264/MPEG-4 AVC may be either binary valued or non-binary multi-valued. The binarization stage is needed for syntax elements that are non-binary valued and this step converts each syntax element in to a unique binary string composed of 1's and 0's referred to as bins.

The next stage is called context modeling in which a model is selected such that the choice may depend on previous encoded syntax elements or bins. After assignment to each bin a context model, the bin and the model is input to the binary coding engine.

In the third and the final stage, coding of bin sequence along with updating of probabilities takes place.

5.3.2. Binarization

CABAC uses four basic types of tree structured codes tables for binarization. Since these tables are rule based, they do not need to be stored. The four basic types are the unary code, the truncated unary code, the *k*th order exp-golomb code, and, the fixed-length code. Further, there is binarization

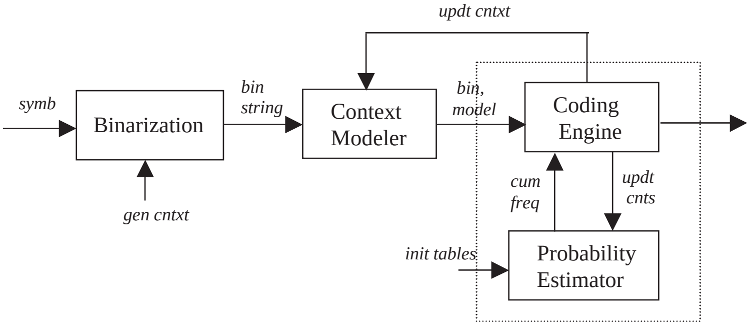

Fig. 29. CABAC encoder block diagram.

based on five less structured manually chosen code tree tables, as well as three binarizations based on concatenation of the four basic types.

5.3.3. *Context modeling*

CABAC uses four basic types of context models based on conditional probability. The first type uses a context template that includes up to two past neighbors to the syntax element currently being encoded. For instance modeling may use a neighbor immediately before and a immediately above the current element, and further, the modeling function may be based on bin-wise comparison with neighbors. The second type of context model is used only for syntax elements of mb_type and sub_mb_type and uses prior coded *i*-th bins for coding of *i*th bin. The third and fourth types of context models are used for residual data only and are used for context categories of different block types. The third type does not rely on past coded data but on the position in the scanning path, and the fourth type depends on accumulated levels.

In addition to these context models, the bins themselves are assigned fixed probability models depending on their indices, for use when no context model of a certain category is available. Overall, the context modeling process only uses coded values of past syntax elements of the same slice.

5.3.4. *Adaptive binary coding engine*

The arithmetic coding engine used is similar to that used by other arithmetic coders and generally involves, based on the probability of a symbol to be coded to perform recursive subdivision to fractional accuracy of an existing interval that initially spans the range from 0 to 1. Since the multiplication operation used in interval subdivision is a reason for increased complexity of arithmetic coders, a multiplication free state transition table based approach is used by CABAC. Complexity is reduced by allowing a simpler coding mode called the ‘bypass mode’.

6. Core encoding issues

6.1. *Motion estimation*

We now briefly discuss the commonalities and differences in motion estimation for encoding with this standard as compared to earlier standards.

6.1.1. *Motion estimation search strategy*

Exhaustive search motion estimation is a compute-intensive process although it can yield better results in terms of reduced overall bit-rate for coding of residual signal resulting from motion compensation. Over the past many years, to reduce the complexity of block motion estimation search, a number of methods have been designed such as, using a subset of residual-block pixels in computing matching criteria, examining a subset of search points in a number of steps [20,30], starting search from a prediction motion vector derived from spatially or temporally adjacent blocks [20,31], using simpler matching criteria such as sum of absolute differences (SAD) instead of mean square error, and, partial search based on threshold values of matching criteria to exit the search etc. In fact, typically, these methods are even combined to reduce complexity even further. While in MPEG-4 part 2, two block sizes, 16×16 and 8×8 are allowed for motion estimation, H.264/MPEG-4 AVC allows motion compensation with many more (16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4) block sizes. However in order to reduce complexity an encoder can very well choose to use only a subset of block sizes, performing a tradeoff of coding efficiency and computational complexity.

For maximum coding gain but at reduced complexity, JVT's JM software implementation of H.264/MPEG-4 AVC starts with a spatial prediction motion vector as a starting estimate [13,20,31] and then performs full search along a spiral path using SAD as a matching criteria for integer pixel search. In the first step, the search window is adjusted to make sure that the zero displacement vector is included in the search. Since the search is done relative to motion prediction based on neighboring blocks, the search range chosen is actually independent of the distance between the picture being coded and the reference picture. Further, use of spiral search allows for the possibility of exiting the search early for some blocks and still achieving reasonable motion compensation. Further, since many block sizes are supported in H.264/MPEG-4 AVC, at each position in the search, SAD's are computed for all 16, 4×4 luminance blocks of a macroblock. The appropriate SAD values are summed to generate corresponding bigger blocks such as 4×8's, 8×4's, 8×8's, 16×8's, 8×16's, and 16×16. The search simply saves all partial SAD values for every integer motion vector candidate position in search. Despite shortcuts, the JM motion estimation search is still excessively compute intensive.

Further, since, typical coding with this standard may use multiple frames, the search may be repeated for all reference frames. The number of frames to use as reference is up to the encoder within the context of values allowed for a level of a profile (more on levels in Section 9.2). Another related issue is that of how many B-frames [25] to use to allow proper tradeoffs of motion estimation complexity with coding efficiency benefits. In H.264/MPEG-4 AVC encoding, aspects related to a priori block-size reduction in motion estimation to reduce complexity, are discussed in [12], and shortcuts are addressed in [41]. In general, it is advisable that before choosing motion estimation shortcuts, considerable investigation be performed to determine robust, noise insensitive, solution that performs well over a wide range of source material and bit-rates.

6.1.2. 1/4 pel using SATD instead of SAD

In [13] for 1/4 pel search, the last stage in motion estimation for computing candidate motion vectors, instead of SAD, a more accurate criterion called sum of absolute transform differences (SATD) can be used. Basically, the residue block representing motion compensated errors is 4×4 Hadammard transformed and the absolute value of 4×4 coefficient array is summed at every 1/4 pixel candidate displacement. The location yielding the smallest SATD value of the 1/4 pel candidates is taken to be the best match and provides the motion vector at 1/4 pel accuracy used for motion compensation.

6.1.3. Rate distortion optimized motion vector decision

The goal of rate distortion optimization for motion estimation [38,37] is to help select the best motion vector and reference picture from the choice of candidate motion vectors and reference pictures during the block matching motion search. For every motion vector and reference picture combination, the coding cost (rate) and the resulting distortion is first computed. Next, a lagrange formulation is applied that calculates the optimum choice by minimizing the following:

$$D_1(m) + \lambda_D \times R(m),$$

where D_1 is the distortion measure used, in this case, sum of absolute prediction difference of the luminance component, R is total number of bits for representing motion information, m is the motion information that includes motion vector and the picture reference parameter, and λ_D is the Lagrange multiplier for motion estimation that controls the tradeoff of rate with distortion.

6.2. Rate distortion optimized mode decision

Like in previous standards, in H.264/MPEG-4 AVC, a macroblock can be coded in one of the many possible modes that are enabled depending on the picture/slice type. For instance, in the case of MPEG-4 part 2 video, P-picture coding offers a choice of up to 6 modes from a set of modes as follows: {*skip, inter, inter with quantizer, inter with* 8×8 *motion vectors, intra, intra with quantizer*}. Thus in MPEG-4 for every macroblock in P-picture a decision needs to be made regarding the coding mode to be used (this decision is also required for macroblocks in I- and B-pictures as well). While, typically in encoding with previous standards, mode decisions were based on criteria such as sum of absolute difference (SAD) or sum of square of difference (SSE), this method of selecting a mode is not optimum. While admittedly mode decision is an encoding issue and thus outside the scope of a standard, it has been shown [13,35] that additional important gains in coding efficiency become possible if macroblock mode decision is performed carefully. Other methods of mode decision that have been tried include explicitly counting bits in coding a macroblock in various modes to select the mode which minimizes the bit count.

Since H.264/MPEG-4 AVC includes many more coding modes than that in earlier standards, it is even more important that the mode decision be correct otherwise with sub-optimum choices, some of the coding efficiency benefits of H.264/MPEG-4 AVC coding may be lost. However, as we will see, thus far the additional gains can be extracted only at the expense of considerable increase in encoding complexity.

The goal of rate distortion optimized mode decision [38,37] is to help select the best block partitioning as well as inter/intra decision for a macroblock. First, the coding cost (rate) and the resulting distortion is computed for all possible macroblock types (as well as submacroblock types). Next, a lagrange formulation is applied that calculates the optimum mode choice by minimizing the following:

$$D_2 + \lambda_M \times R(M|Q),$$

where D_2 is the distortion criterion used, in this case, sum of square of reconstruction error of the macroblock, R is total number of bits that includes mode signaling bits, motion vector coding bits, and transform coefficient coding. bits for the macroblock, λ_M is the lagrange multiplier for mode decision that controls the tradeoff of rate with distortion, Q is the quantizer used for quantization of coefficients of the macroblock undergoing mode decision evaluation, and, M is the set of modes available, as an example, for a P-slice in H.264/MPEG-4 AVC, the set of potential macroblock types are

$$\{skip,\ inter\_16\times 16,\ inter\_16\times 8,\ inter\_8\times 16,\ inter\_8\times 8,\ intra\_4\times 4,\ intra\_16\times 16\}$$

and is applied after first making a similar decision on submacroblock types. of a macroblock.

The mode decision lagrange multiplier has been experimentally found to be related to quantizer parameter as follows:

$$\lambda_M = 0.85 \times 2^{\mathrm{QP}/3}.$$

Further, the lagrange multiplier for motion is chosen to relate to the lagrange multiplier for mode as follows:

$$\lambda_D = \sqrt{\lambda_M}.$$

6.3. Quantizer adaptation and rate control

Rate control is another operation performed only during encoding and thus is not an issue for standardization. However very much like motion estimation and mode decision, it can have a significant impact on the coded video quality so it is an important topic nevertheless. While an important requirement in rate control is to ensure that on the average, coding bit-rate does not exceed target bit-rate, this has to be done while maintaining acceptable video quality. Thus adaptive quantization is also closely related to rate control [26] as adaptation of quantizer used in transform coding is a common approach to control rate of generation of bits in video coding. More successful techniques for rate control have to be generally aware of characteristics of the content, features of video coders, as well as spatial/temporal quality expectations from an application. Being aware of codec features typically involves knowing about, individual picture types (I-, P-, B- and others) and their bit-rate needs, picture coding structures [25] that can be derived from picture types, tradeoffs in motion coding vs. transform coding, impact of quantizer adjustment vs. frame dropping etc. Among the many solutions for rate control available, the rate control of MPEG-2 Test Model 5 (TM5) [36] still offers a good starting point and can be the basis of design for a new, custom rate controller. The rate controller consists of three main steps—target bit allocation, virtual buffer based bit-rate control, and adaptive quantization. TM5 rate controller while it is a reasonable starting point, it is also known to have well-documented shortcomings. This is usually an issue when coding bit-rates are rather limited such as that for many applications of H.264/MPEG-4 AVC standard. For the purpose of showing good coding quality under bit-rate control with H.264, effort has been put in JM development to design new rate controllers [12] suitable for the H.264 standard. However, a rate controller that achieves good quality is generally an application dependent issue so it is difficult to design a single solution that works well for all types of bit-rates, frame rates, and video material. Besides, there are several new issues with H.264 as compared to earlier standards that one needs to be careful about in designing a rate controller. While a detailed discussion of such issues is outside of the scope of this paper, a number of relevant the issues can be listed as follows.

- Since coding occurs at relatively lower bit-rates, larger bit-rate fluctuations can occur causing difficulties in rate control.
- The typical distribution of bits between motion and texture may cause difficulties in rate control if only texture bits are controlled. Also the quantizer precision may not be sufficient for good rate control for some applications.
- The bit-rates for B-pictures of H.264 being small (than earlier standards) can add to difficulties in rate control.
- With changes in quantizer and thus in loop filtering, spatial/temporal quality variations may appear as artifacts.
- Quantizer changes need to be carefully restricted based on scene complexity, picture types, and, coding bit-rate.
- Improper mode decision/reference indexing tradeoffs can at times cause excessive bits generated for certain frames.
- Macroblock quantizer or RDopt lambda parameter changes can introduce spatial quality variations in fine texture.

7. Experimental results

While overall the H.264/MPEG-4 AVC standard provides significant improvement over earlier standards, often the coding results for the standard quoted are those achieved by the JM software implementation. As mentioned earlier, H.264 like previous standards only specifies a bitstream syntax and decoding semantics, the encoder optimizations are really outside of the standard. However the software does include a variety of encoding optimization for picture quality, some of them such as motion estimation, and RD Optimization (RDOpt) add significantly to complexity. For design of a practical encoder, often, shortcuts must be found and so it is essential to understand the picture quality performance individual tools and as encoding optimizations/simplifications provide so that proper performance/complexity tradeoffs can be made. Towards this end, we report results of a detailed set of experiments we have conducted using MPEG standard test sequences and other test sequences, organized as three test sets in Table 10. Further, for all our experiments we have used the public domain JVT implementation referred to as JM6.1e [13] so that results are easier to compare with other published results.

7.1. *Motion estimation*

7.1.1. *Motion vector range*

In Table 11, we present results of experiments with different motion update ranges (with respect

Table 10
Sequences in each of the three test sets used in experiments and quantizers used in coding

CIF (352 × 288, 30 fps) test set			Interlaced (704 × 480, 30 fps) test set			Movie (704 × 352, 24 fps) test set		
Sequence	Num Frm	Qi,*p*,*b*	Sequence	Num Frm	Qi,*p*,*b*	Sequence	Num Frm	Qi,*p*,*b*
Mobile	300	30,31,32	Mobile	150	30,31,32	Apollo13.1	96	28,29,30
Tempete	260	30,31,32	Carousel	150	30,31,32	Apollo13.2	149	38,29,30
TableTen(nis)	300	29,29,31	TableTen(nis)	150	29,29,31	Golden(Eye).1	177	29,29,31
FlowerGa(rden)	250	30,31,33	FlowerGa(rden)	150	30,31,33	Golden(Eye).2	83	28,28,30
Bus	150	30,30,32	Bus	150	30,30,32	AnyGiv(enSunday).1	65	31,31,33
Football	150	30,31,32	Football	150	30,31,32	AnyGiv(enSunday).2	54	29,30,32
Stefan	300	30,31,33						

Table 11
Motion vector (MV) range experiments on CIF test set while coding using 1 B-pic, 1 Ref pic, spatial direct, RDOpt, and CAVLC, spatial direct, and RDOpt

Sequence	MV range ± 15		MV update range ± 23		MV update range ± 31	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	740.16	31.01	741.44 (+0.17%)	31.01	741.03 (+0.11%)	31.01
Tempete	580.51	32.11	579.81 (−0.12%)	32.11	578.80 (−0.31%)	32.12
TableTen.	470.08	34.30	468.10 (−0.42%)	34.30	467.34 (−0.58%)	34.30
FlowerGa.	864.40	31.62	863.85 (−0.06%)	31.61	863.58 (−0.09%)	31.62
Bus	717.96	32.82	716.38 (−0.22%)	32.82	715.42 (−0.35%)	32.82
Football	773.89	34.57	732.40 (−5.36%)	34.57	720.68 (−6.87%)	34.58
Stefan	888.46	32.66	779.08 (−12.31%)	32.70	728.50 (−18.00%)	32.70

to prediction motion vector) while using *spiral full search* motion estimation during coding of CIF test set with JM.

The results show that the differences are significant only on fast motion sequences such as *Football* and *Stefan* where prediction motion vectors used in motion estimation may not work well thus requiring a larger update range. For subsequent experiments with CIF test set, MV update range of 16 will be used for *Mobile*, 23 will be used for *Tempete*, *TableTennis*, *FlowerGarden* and *Bus*, and, a range of 31 will be used for *Football* and *Stefan*.

7.1.2. *Without and with Hadammard*

Table 12 shows results of experiments without, and with Hadammard transform used in selection of the best $\frac{1}{4}$ pel motion vector in motion estimation while coding of CIF test set with JM.

The results show only small statistical differences, with the largest difference only on *Stefan* sequence, however, visually, "with Hadammard" appears preferable and thus it will be used in all our subsequent experiments.

7.2. *Number of B-pictures*

Table 13 shows results of experiments with no B-picture, 1 B-picture, and 2 B-pictures in coding of CIF test set with JM.

The results show significant reduction in bit-rate of typically in 9–27% range when 1 B-picture is used as compared to no B-pictures. While with chosen quantizers, 1 B-picture produces lower SNR than the no B-picture case, the visual quality difference is imperceptible due to temporal masking benefits of B-pictures. Further, 2 B-pictures provide an additional gain

Table 12
Without, and with Hadammard experiments on CIF test set while coding using 1 B-pic, 1 Ref pic, spatial direct, RDOpt, and CAVLC

Sequence	Without Hadammard		With Hadammard	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	735.73	30.93	740.16 (+0.60%)	31.01
Tempete	575.04	32.03	579.81 (+0.83%)	32.11
TableTen.	466.66	34.24	468.10 (+.0.31%)	34.30
FlowerGa.	863.80	31.56	863.85 (+0.00%)	31.61
Bus	712.90	32.75	716.38 (+0.49%)	32.82
Football	719.93	34.53	720.68 (+0.10%)	34.58
Stefan	736.21	32.64	728.50 (−1.05%)	32.70

Table 13
No B-pic, 1 B-pic, 2, B-pic comparison experiments on CIF test set while coding using 1 Ref pic, spatial direct, RDOpt, and CAVLC

Sequence	No B-pic		1 B-pic		2 B-pic	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	1106.98	31.12	740.16 (−33.14%)	31.01	631.73 (−42.93%)	30.80
Tempete	791.57	32.24	579.81 (−26.75%)	32.11	517.98 (−34.56%)	31.90
TableTen.	583.15	34.41	468.10 (−19.73%)	34.30	433.27 (−25.70)	34.16
FlowerGa.	1132.73	32.02	863.85 (−23.74%)	31.61	765.65 (−32.41%)	31.23
Bus	886.97	33.08	716.38 (−19.23%)	32.82	665.19 (−25.00%)	32.58
Football	761.27	34.77	720.68 (−5.33%)	34.58	712.99 (−6.34%)	34.34
Stefan	803.07	33.05	728.50 (−9.29%)	32.70	667.94 (−16.835)	32.40

typically in 6–9% range with further reduction in SNR but with hardly any reduction in visual quality.

7.3. Number of reference pictures

Table 14 shows results of experiments with 1 reference picture, 2 reference pictures, 3 reference pictures, and 5 reference pictures while using 1 B-picture in coding of CIF test set with JM. A clarification is in order of how to interpret the terminology used such as "*p* Ref, *q* B-pic" used in this as well as many subsequent tables, particularly since, for example, the term such as "1 Ref, 1 B-pic" can be confusing. Basically, as one would expect, all B-pictures used, employ a prediction in backward direction and a prediction in forward direction. As in coding with earlier standards, one decoded future picture is used to provide backward prediction. Unlike previous standards, forward prediction uses one of '*p*' choices of pictures such that the best reference is chosen on a MB basis. Further, prediction in P-pictures is performed similar to prediction in forward prediction mode of B-pictures, i.e., choosing on a MB basis best reference picture from a choice of '*p*' references. As mentioned earlier, we use JM software for our coding experiments, and we believe this is how the JM6.1e software was designed to work as it allows independent selection of '*p*' and '*q*'.

The results show that when two reference pictures are used instead of one, typical bit-rate reduction is around 2.5–3% but for sequences with background that remains similar, it can be as high as 7%; 1 B-picture coding is used. Further, if 5 reference pictures are used instead of 1, typical gains could be in 5% range, and for sequences with

Table 14
1 Ref pic, 2 Ref pic, 3 Ref pic, and 5 Ref pic comparison experiments on CIF test set while coding using 1 B-pic, spatial direct, RDOpt, and CAVLC

Sequence	1 Ref, 1 B-pic		2 Ref, 1 B-pic		3 Ref, 1 B-pic		5 Ref, 1 B-pic	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	740.16	31.01	693.53 (−6.29%)	31.18	676.57 (−8.59%)	31.25	664.04 (−10.28%)	31.35
Tempete	579.81	32.11	537.97 (−7.22%)	32.27	531.94 (−8.26%)	32.34	524.88 (−9.47%)	32.40
TableTen.	468.10	34.30	456.28 (−2.53%)	34.34	452.45 (−3.34%)	34.36	445.56 (−4.81%)	34.38
FlowerGa.	863.85	31.61	834.15 (−3.44%)	31.68	824.45 (−4.56%)	31.71	814.65 (−5.69%)	31.73
Bus	716.38	32.82	699.92 (−2.29%)	32.88	698.74 (−2.46%)	32.89	697.57 (−2.63%)	32.90
Football	720.68	34.58	714.61 (−0.84%)	34.57	714.07 (−0.92%)	34.57	710.99 (−1.34%)	34.57
Stefan	728.50	32.70	709.11 (−2.66%)	32.77	697.62 (−4.24%)	32.78	697.91 (−4.20%)	32.80

Table 15
1 Ref pic, 2 Ref pic, 3 Ref pic, and 5 Ref pic comparison experiments on CIF test set while coding using 2 B-pic, spatial direct, RDOpt, and CAVLC, with JM6.1e

Sequence	1 Ref, 2 B-pic		2 Ref, 2 B-pic		3 Ref, 2 B-pic		5 Ref, 2 B-pic	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	631.73	30.80	610.44 (−3.37%)	30.96	602.91 (−4.56%)	31.03	597.84 (−5.36%)	31.11
Tempete	517.98	31.90	491.95 (−5.02%)	32.05	488.70 (−5.65%)	32.11	486.71 (−6.03%)	32.15
TableTen.	433.27	34.16	425.41 (−1.81%)	34.17	422.82 (−2.40%)	34.19	417.17 (−3.715)	34.22
FlowerGa.	765.65	31.23	750.87 (−1.93%)	31.29	745.66 (−2.61%)	31.32	737.68 (−3.66%)	31.34
Bus	665.19	32.58	657.58 (−1.14%)	32.61	658.70 (−0.97%)	32.63	656.35 (−1.33%)	32.63
Football	712.99	34.34	710.04 (−0.41%)	34.33	709.66 (−0.47%)	34.33	708.76 (−0.59%)	34.34
Stefan	667.91	32.40	677.75 (+1.43%)	32.44	668.94 (−1.47%)	32.46	672.49 (+0.70%)	32.46

background that remains similar, gains could be as high as 10%.

Table 15 shows results of experiments with 1 reference picture, 2 reference pictures, 3 reference pictures, and 5 reference pictures while using 2 B-pictures in coding of CIF test set.

The results show, lower improvements when many reference pictures are used with 2 B-picture coding as compared to 1 B-picture coding. When 2 reference pictures are used instead of 1, typical bit-rate reduction is in the range of 1–2% but for sequences with background that remains similar, it can be as high as 5%; 2 B-pictures are used in coding. Further, if 5 reference pictures are used instead of 1, typical gains could be around 3.5%, and for sequences where background remains similar, the gains could be around 5.5–6%. A note of caution: for more complex sequences with fast motion when using 2 B-pictures, there may be no gains unless mode decisions are perfect.

7.4. All MB vs. no sub8 × 8 MC modes

In Table 16, we present results of experiments with all MB partitions for motion compensation, with no smaller than 8 × 8 blocks (no sub8 × 8) in coding of CIF test set with JM.

The results show that with 5 references and 1 B-picture, if no sub8 × 8 MC modes are used instead of all MC modes, typically 1–3% increase in bit-rate results, but could be as high as 5%. Further, with 3 references and 2 B-picture, if typically, 1–5% increase in bit-rate results, but could be as high as 6%.

Table 16
All MB vs. no sub8 × 8 MC modes experiments on CIF test set while coding using either 1 B-pic and 5 Ref, or, 2 B-pic and 3 Ref, spatial direct, RDOpt, and CAVLC

Sequence and Qi, *p*, *b*	All MB MC modes, 5 Ref, 1 B-pic		No sub8 × 8 block MC modes, 5 Ref, 1 B-pic		All MB MC modes, 3 Ref, 2 B-pic		No sub8 × 8 block MC modes, 3 Ref, 2 B-pic	
	Bit-rate kbps	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	664.04	31.35	670.83 (+1.02%)	31.21	602.91	31.03	615.61 (+2.11%)	30.91
Tempete	524.88	32.40	525.02 (+0.03%)	32.26	488.70	32.11	492.90 (+0.86%)	31.99
TableTen.	445.56	34.38	450.18 (+1.04%)	34.30	422.82	34.19	427.83 (+1.18%)	34.12
FlowerGa.	814.65	31.73	854.52 (+4.89%)	31.56	745.66	31.32	787.70 (+5.64%)	31.16
Bus	697.57	32.90	705.79 (+1.18%)	32.79	658.70	32.63	669.63 (+1.57%)	32.52
Football	710.99	34.57	715.89 (+0.69%)	34.56	709.66	34.33	714.98 (+0.75%)	34.33
Stefan	697.91	32.80	718.92 (+3.01%)	32.68	668.94	32.46	707.66 (+5.79%)	32.34

7.5. *RDOpt vs. non-RDOpt mode selection*

Table 17 presents results of experiments with and without RDOpt mode selection on CIF test set coded with JM.

The results show that "no RDOpt" as compared to "RDOpt" for the case of 5 References and 1 B-picture results in 6.5%–13% increase in bit-rate. Further, with 3 references and 2 B-picture, typical increase in bit-rate may be in the range of 10%–22%.

7.6. *Spatial vs. temporal direct mode*

Next, in Table 18, we present results of experiments comparing Spatial Direct mode and Temporal Direct mode for B-pictures using CIF test set coded with JM.

The results show that comparison of temporal direct mode with spatial direct mode for the case of 5 References and 1 B-picture, performs in the range of nearly the same (±0.5%) to slight increase (3%) in bit-rate on this test set. Further, with 3 references and 2 B-picture, variations in its performance may lie over a wider range, e.g., 3.5% reduction in bit-rate to 5% increase in bit-rate. Possibly, temporal direct may perform better on lower detailed scenes at lower coding bit-rates where higher quantizer values are used.

7.7. *Loop filter on vs. loop filter off*

In Table 19 we show results of experiments with and without loop filter in coding of CIF test set with JM.

The results show that the case of "no loop filter" as compared to having "default loop filter" performs very close, mainly produces bit-rate higher by up to 1%, but at the most 2% higher in bit-rate. So, at least, the bit-rate savings from loop filter is not significant. Since the main purpose of loop filter is in blockiness reduction, as expected, its benefit is in improvement of visual quality, especially when bit-rates are lower.

7.8. *CAVLC vs. CABAC*

Results of experiments comparing CAVLC with CABAC in coding of CIF test set with JM are shown in Table 20.

The results show that with "CAVLC" as compared to "CABAC" for the case of 5

Table 17
With RDOpt and no RDopt experiments on CIF test set with coding using either 1 B-pic and 5 Ref, or, 2 B-pic and 3 Ref, spatial direct, and CAVLC

Sequence and Qi, *p*, *b*	With RDOpt, 5 Ref, 1 B-pic		No RDOpt, 5 Ref, 1 B-pic		With RDOpt, 3 Ref, 2 B-pic		No RDOpt, 3 Ref, 2 B-pic	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	664.04	31.35	752.59 (+13.33%)	31.40	602.91	31.03	737.12 (+22.26%)	31.22
Tempete	524.88	32.40	593.28 (+13.03%)	32.45	488.70	32.11	594.27 (+21.60%)	32.28
TableTen.	445.56	34.38	476.57 (+6.96)	34.30	422.82	34.19	466.39 (+10.17%)	34.17
FlowerGa.	814.65	31.73	861.82 (+5.79%)	31.77	745.66	31.32	821.52 (+10.17%)	31.47
Bus	697.57	32.90	743.12 (+6.53%)	32.85	658.70	32.63	726.92 (+10.36%)	32.67
Football	710.99	34.57	782.27 (+10.03%)	34.67	709.66	34.33	797.45 (+12.37%)	34.55
Stefan	697.91	32.80	708.08 (+1.46%)	32.81	668.94	32.46	738.38 (+10.38%)	32.54

Table 18
Spatial direct and temporal direct mode comparison experiments on CIF test set with coding using either 1 B-pic and 5 Ref, or, 2 B-pic and 3 Ref, RDOpt, and CAVLC

Sequence and Qi, *p*, *b*	Spatial direct, 5 ref, 1 B-pic		Temporal direct, 5 ref, 1 B-pic		Spatial direct, 3 ref, 2 B-pic		Temporal direct, 3 ref, 2 B-pic	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	664.04	31.35	685.81 (+3.28%)	31.31	602.91	31.03	609.14 (+1.03%)	31.00
Tempete	524.88	32.40	527.45 (+0.49%)	32.38	488.70	32.11	471.87 (−3.44%)	32.11
TableTen.	445.56	34.38	443.89 (−0.37%)	34.38	422.82	34.19	423.60 (+0.18%)	34.20
FlowerGa.	814.65	31.73	821.71 (+0.87%)	31.75	745.66	31.32	743.03 (−0.35%)	31.33
Bus	697.57	32.90	693.11 (−0.64%)	32.89	658.70	32.63	653.70 (−0.76%)	32.63
Football	710.99	34.57	725.83 (+2.08%)	34.63	709.66	34.33	738.53 (+4.07%)	34.43
Stefan	697.91	32.80	720.60 (+3.25%)	32.8 5	668.94	32.46	701.58 (+4.88%)	32.56

References and 1 B-picture, average reduction in bit-rate is around 6.8%. Typical values lie in range of 5–6.5%, reaching 8% in one case and over 10% in another case. For the case of 3 references and 2 B-pictures the results are similar, with average reduction in bit-rate over CAVLC found to be 7.2%. The performance of CABAC may be higher for much simpler scenes but such scenes are

Table 19
Loop filter vs. no loop filter comparison experiments on CIF test set with coding using either 1 B-pic and 5 Ref, or, 2 B-pic and 3 Ref, RDOpt, and CAVLC

Sequence	Loop filter, 5 ref, 1 B-pic		No loop filter, 5 ref, 1 B-pic		Loop filter, 3 ref, 2 B-pic		No loop filter, 3 ref, 2 B-pic	
	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)
Mobile	664.04	31.35	665.27 (+0.19%)	31.30	602.91	31.03	606.19 (+0.54%)	30.99
Tempete	524.88	32.40	528.30 (+0.65%)	32.39	488.70	32.11	490.11 (+0.29%)	32.07
TableTen.	445.56	34.38	452.02 (+1.45%)	34.41	422.82	34.19	427.54 (+1.11%)	34.23
FlowerGa.	814.65	31.73	814.19 (−0.06%)	31.72	745.66	31.32	744.62 (−0.14%)	31.31
Bus	697.57	32.90	703.36 (+0.83%)	32.66	658.70	32.63	662.27 (+0.54%)	32.60
Football	710.99	34.57	722.57 (+1.63%)	34.53	709.66	34.33	719.60 (+1.40%)	34.28
Stefan	697.91	32.80	697.37 (−0.08%)	32.74	668.94	32.46	681.61 (+1.89%)	32.38

Table 20
CAVLC vs CABAC comparison experiments on CIF test set with coding using either 1 B-pic and 5 Ref, or, 2 B-pic and 3 Ref, and RDOpt

Sequence	CAVLC, 5 ref, 1 B-pic		CABAC, 5 ref, 1 B-pic		CAVLC, 3 ref, 2 B-pic		CABAC, 3 ref, 2 B-pic	
	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)	Bit-rate (kbps)	Y SNR (dB)
Mobile	664.04	31.35	626.94 (−5.6%)	31.43	602.91	31.03	569.42 (−5.6%)	31.13
Tempete	524.88	32.40	499.81 (−4.8%)	32.47	488.70	32.11	462.89 (−5.3%)	32.17
TableTen.	445.56	34.38	419.31 (−5.9%)	34.40	422.82	34.19	396.34 (−6.3%)	34.22
FlowerGa.	814.65	31.73	727.92 (−10.6%)	31.81	745.66	31.32	666.22 (−10.7%)	31.40
Bus	697.57	32.90	656.05 (−5.9%)	32.95	658.70	32.63	617.28 (−6.3%)	32.69
Football	710.99	34.57	663.69 (−6.7%)	34.72	709.66	34.33	666.62 (−6.1%)	34.45
Stefan	697.91	32.80	641.14 (−8.1%)	32.87	668.94	32.46	600.01 (−10.3%)	32.52

usually not hard to code. Another case where CABAC's performance may be higher is when bit-rates are rather low, or in other words, large quantizer step sizes are used. In such a coding scenario, resulting quality may not be that good regardless.

Table 21
Frame mode, Field mode, Pic AFF mode and MB AFF mode comparison experiments on interlaced 4:2:0 test set while coding using 3 Ref pic, 2 B-pic, spatial direct, RDOpt, and CAVLC

Sequence	Frame		Field		PicAFF		MBAFF	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Mobile	2133.84	31.10	2115.42 (−0.86%)	30.89	2005.15 (−6.03%)	31.08	1887.88 (−11.53%)	31.20
Carousel	4620.54	32.19	3307.84 (−28.41%)	32.66	3297.01 (−28.64%)	32.64	3143.61 (−31.96%)	32.66
TableTen.	2157.62	32.70	1991.05 (−7.72%)	32.53	1898.15 (−12.02%)	32.74	1856.20 (−13.97%)	32.78
FlowerGa.	2989.56	31.00	2945.73 (−1.47%)	30.92	2594.88 (−13.20%)	31.08	2569.45 (−14.05%)	31.14
Bus	2719.76	32.27	2160.37 (−1.47%)	32.63	2058.34 (−24.32%)	32.64	1983.15 (−27.08%)	32.75
Football	2688.56	33.61	1853.68 (−31.05%)	34.21	1841.43 (−31.51%)	34.19	1791.07 (−33.38%)	34.23

7.9. Interlaced video coding

We now present results of experiments with tools especially designed for efficient coding of interlaced video. Table 21 shows a comparison of 4 potential modes: frame, field, Pic adaptive frame/field, and macroblock adaptive frame/field for coding of interlaced video.

The results exhibit a strong dependence on motion and details in the interlaced video sequence being coded. For very detailed scenes with slow motion (e.g. Mobile), picture-based frame or picture-based field modes perform similarly and PicAFF provides half of the improvement possible with MBAFF. Thus MBAFF is quite useful for such scenes. For scenes with medium trackable motion and medium to high details (e.g. FlowerGarden, Bus) while field mode performs similar to frame mode, PicAFF is able to achieve nearly all the improvement possible, and MBAFF provides small additional improvement. For scenes with fast motion and low to medium details (e.g. Carousel, Football) field mode provides most of the improvement and the performance of PicAFF mode is identical to the field mode, with MBAFF providing small additional improvement. In a mixed sequence with two very different subscenes (e.g. TableTennis), field coding achieves over half of the total gains, with PicAFF able to achieve most of the gains.

Overall, PicAFF is able to achieve a significant portion of the gain available. MBAFF provides additional SNR gain or bit-rate saving gain for sequences with scenes containing mixed (moving and static) regions. However, gain for MBAFF tends to be more local in nature and thus more noticeable in visual quality locally rather than in an average SNR/bits savings measurement for a sequence.

7.10. Movie scenes coding

In Table 22, we present results of our experiments on coding of movie scenes test set under pre-selected combinations of reference frames, B-pictures, and CAVLC or CABAC.

For the movie scene test set, the results show a significant reduction in bit-rate in 9–19% range (14% average) when 1 B-picture coding is used compared to no B-picture coding. This reduction in bit-rates is accompanied with slight reduction in SNR, but visually the quality is the same as in the case of no B-pictures. Using 2 B-pictures, an additional gain of about 3–10% was obtained for a total improvement range of 12–30% over no B-pictures. Next, for the 3 reference and 2 B-picture case we replaced CAVLC with CABAC and find that additional improvement of 5–10% range (7% average) could be obtained with use of CABAC.

8. Additional tools, features, and system support

8.1. Arbitrary slice order, and slice groups

As mentioned earlier, a picture may be divided into one or more slices, where a slice is a sequence

Table 22
Experiments in coding of movie test set with coding using either 3 Ref pic and no B-pic, or 3 Ref pic and 1 B-pic, or 3 Ref pic and 2B-pic, or 3 Ref pic ,2 B-pic and CABAC; spatial direct, and RDOpt

Sequence	3 Ref pic, No B-pic, CAVLC		3 Ref pic, 1 B-pic, CAVLC		3 Ref pic, 2 B-pic, CAVLC		3 Ref pic, 2 B-pic, CABAC	
	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)	Bit-rate (kbps)	*Y* SNR (dB)
Apollo.1	445.81	37.46	406.55 (−8.81%)	37.37	386.69 (−13.26%)	37.20	352.72 (−20.87%)	37.24
Apollo.2	726.21	34.66	583.82 (−19.61%)	34.54	511.35 (−29.59%)	34.40	480.48 (−33.83%)	34.45
Golden .1	892.29	33.94	725.25 (−18.72%)	33.92	631.26 (−29.25%)	33.66	582.33 (−34.74%)	33.73
Golden.2	777.23	39.13	660.14 (−15.20%)	38.72	605.91 (−22.04%)	38.50	554.97 (−28.59%)	38.56
AnyGiv.1	1804.41	33.99	1632.26 (−9.54%)	33.55	1589.49 (−11.92%)	33.28	1456.61 (−19.27%)	33.34
AnyGiv.2	778.67	36.51	680.54 (−12.60%)	36.31	642.45 (−17.49%)	36.03	588.71 (−24.39%)	36.08

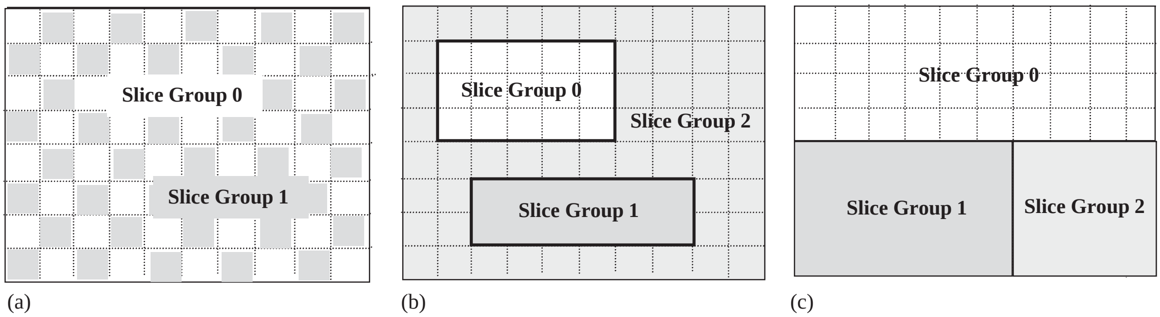

Fig. 30. (a)–(c) Examples of macroblocks organized into slice groups.

of macroblocks. For the purpose of coding, a slice is defined to be a self-contained entity.

In some profiles of the standard, slices can be sent in an arbitrary order. When the slices are sent in an arbitrary order, at the decoder, macroblock address of the first macroblock of a slice may be smaller than macroblock address of some other preceding slice in the same coded picture.

Further, slices can be bundled into groups, appropriately called, slice groups. Thus, a slice group may have one or more slices. The exact number of slices is specified by a parameter (num_slice_groups_minus1) in picture_parameter_set. If there are more than one slice groups an additional parameter slice_group_map_type specifies a number of mapping arrangements of slices and can take a value in the range of 0 to 6. Fig. 30 shows examples of a few arrangements of slice groups supported by the standard.

When the value of slice_group_map_type is 0, it means interleaved slice groups, when 1, it means dispersed allocation, when 2, it means 1 or more slice groups may represent foreground with left-over, when 3,4,5, it means that slice groups are changing as per a preset pattern which can be deduced from size and shape, and, when 6, it means explicit assignment slice group to each macroblock. For each value of slice_group_map_type, different syntax elements may be carried in the bitstream to precisely be able to deduce at the decoder, mapping of macroblocks to slice groups used while encoding.

While a detailed discussion of various syntax elements is outside of the scope of this paper, in Table 23, we include a summary of various slice-group mapping types, and their meaning and syntax elements needed.

Further, the standard also supports the concept of redundant slices, which implies that in certain

Table 23
Slice group mapping identification

slice_group_map_type	Slice group mapping	Info carried in bitstream
0	Interleaved	run_length_minus1
1	Dispersed	—
2	1 or more foreground, and a left-over	top_left[i], bottom_right[i]
3	Changing slice groups: box-out slice	slice_group_change_direction _flag and slice_group_change_rate_minus1
4	Changing slice groups: raster scan	slice_group_change_direction _flag and slice_group_change_rate_minus1
5	Changing slice groups: wipe	slice_group_change_direction _flag and slice_group_change_rate_minus1
6	Explicit assignment of slice group to each map unit	picture_size_in_map_units_minus1, slice_group_id[i]

error prone applications, certain coded slices may be sent more than once (repeated) as needed.

8.2. 3:2 Pull down

In 525-line interlaced display video systems such as that used in north America television video signals are sampled and transmitted at approximately 59.94 fields/ps. The digital television system for 525-line interlaced video format, uses compressed video streams encoded using MPEG-2 video at approximately 29.97 frames per second (fps). Hereinafter, for simplicity, we may use the term 30 fps to refer to rates such as 29.97 fps.

Film material produced at 24 fps is routinely converted to 60 field/ps in many applications. This is known as the 3:2 pull-down process. When using the MPEG-2 video standard with the film mode, the frame rate encoded in the sequence header is 30 fps for interlaced display, even though the video is actually coded as a 24 fps film sequence. The encoder also conveys to the decoder, proper display timing based on the frame rate of 30 fps. The flags **top_field_first** and **repeat_first_field** in the picture coding extension header are used for indicating how a picture should be displayed. These two flags are mandated MPEG-2 syntax elements that are carried in the bitstream for use by the decoder. However, this lack of flexibility is not be desirable, particularly, when the type of display device may vary, for example, either an interlaced television or a progressive monitor. Furthermore, the encoder does not know the type of display employed at the decoder end.

In MPEG-2, the flags **top_field_first** and **repeat_first_field** along with the frame rate can also be used to derive Decoding Time Stamps (DTS) and Presentation Time Stamps (PTS) for some pictures. The flags (i.e., top_field_first and repeat_first_field) are used to achieve proper timing for decoding and displaying the coded 24 fps film material to generate output video at 30 fps. As mentioned earlier, this may not be desirable when the display device is not an interlaced television (e.g., it is a progressive monitor), and further, since the encoder does not know the type of display employed at the decoder. The problem may be further complicated because, in broadcast systems, there may be many decoders decoding the same signal and possibly different types of monitors being used to display the same signal.

H.264/MPEG-4 AVC provides an alternative approach [4,6] for solving this problem. In H.264/MPEG-4 AVC, the picture timing SEI message can be used as a picture-level optional hint message for display related information that indicates how the picture should be displayed on interlaced monitors. The hint message may be used to facilitate the coding and display of, for example, converted film material.

The encoding and display processes may be decoupled using the hint messages to support interlaced display. For example, a film sequence may be originally coded at the frame rate of 24 fps. Hint messages may be inserted in each picture-level header to indicate the 3:2 pull-down process for interlaced display. It is different from typical use of MPEG-2 video because the frame rate here is the rate for actual coded frames. Decoders may be free to ignore or to use the optional hint messages. Decoders with progressive display devices may, for example, ignore the hints. Decoders with interlaced display devices may, for example, use those hints in a manner that is similar to the MPEG-2 film mode.

This alternative approach allows encoding systems that are capable of coding progressive or interlaced video to not to have to make implicit assumptions about the nature of the display device, e.g. encoding the film material at the actual desired film frame rate (e.g., 24 fps) in a progressive sequence. The encoder may embed hints in the coded bit stream about how to display the encoded content, while keeping the timing information of the encoded video based on a source temporal sampling. Accordingly, in some cases, the time-stamping process for DTS and PTS in MPEG-2 systems may be different than that for MPEG-2 video.

8.3. Trick modes for PVR

Personal video recorder (PVR) offers consumers a hard disk or networked-based VCR that digitally records live video programs while offering the versatility of select playback and associated special features. The viewer can take advantage of trick play features such as pause/still, fast forward, slow forward, rewind, slow reverse, skip, etc.

When the compressed stream is being recorded, an index table pointing to start code within the stream can be created and written into a file. The table is used for indexing locations of (I-, P-, and B-) pictures within the stream to allow a decoder to further manipulate the stream such as to remove certain pictures or other trick mode elements without parsing the entire stream. In prior standards, picture-coding type for each picture is always using a fix-length code for simplicity and immediately follows picture start code. This makes the table creation process simple. However, if the compressed stream is scrambled, the picture start codes and coding types are scrambled since these are hard to separate from other picture header data. Therefore, in this case, the index table cannot be created without descrambling the stream first.

In H.264/MPEG-4 AVC, Access unit delimiter NAL unit can be used to index location of primary decoded picture within the stream to support trick mode. To provide better indexing efficiency and more security, the access unit delimiter NAL unit can be transferred without scrambling while all other AVC contents are scrambled. During recording stage, the video streams do not need to be descrambled. Since access unit delimiter NAL unit is not scrambled, indexing engine can find the unit and build the index table without descrambling and processing the video content, i.e. recorded video content is still in its originally scrambled form.

When a stored stream is played back, in order to accomplish trick mode for audio or/and video, modifications to the configuration of the decoder and/or manipulations to the stream itself may be used. Common applications of this would be to allow for fast-forward or rewind of a stream while it is being played from a disk. If a stream is manipulated after recording but before playback by the decoder in the proper way, the decoder can be configured to decode the result as if it had not been altered at all. An example would be to send only I-frames to the decoder, dropping P and B frames before playback.

8.4. NAL packetization

This standard is designed for diverse applications ranging from broadcasting over Cable, Satellite or Terrestrial networks to streaming over IP networks to video telephone or conferencing over wireless or ISDN channels. They use different protocols and schemes for the distribution of compressed video. To allow maximum flexibility to adapt the syntax to a particular application, network environment and delivery mechanism, the video syntax is broken into two layers—video coding layer (VCL), and network abstraction layer (NAL). VCL consists of the bits associated with the slice layer or below - the primary domain of the compression tools. NAL formats the compressed video data (VCL) and provides additional non-VCL information such as, sequence and picture parameters, access unit delimiter, filler data, supplemental enhancement information (SEI),

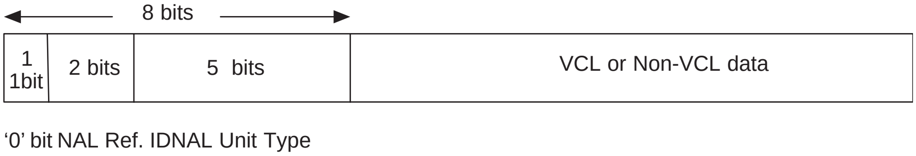

Fig. 31. NAL unit format.

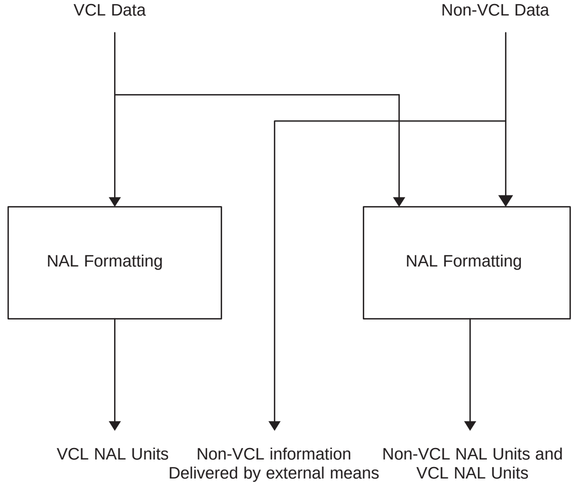

Fig. 32. NAL formatting of VCL and non-VCL data.

display parameters, picture timing etc., in a way most appropriate for a particular network/system such as packet oriented, or bitstream oriented. For example, to carry the data using MPEG-2 TS, start code prefixes are added and specific order in which information should arrive is defined. In the standard, this type of format is called Byte-Stream format.

All data related to video stream is encapsulated in packets called NAL Units (NALU). Format of a NALU is shown in Fig. 31.

First byte of each NALU is a header byte and the rest is the data (including emulation prevention byte). First bit of the header is a 0 bit. Next 2 bits indicate whether the content of NALU consist of sequence or picture parameter set or a slice of a reference picture. Next 5 bits indicate the NALU type corresponding to the type of data being carried in that NALU. There are 32 types of NALUs allowed. These are classified in two categories: VCL NAL units and non-VCL NAL Units. NALU types 1–5 are VCL NALUs and contain data corresponding to the VCL. NALUs with NALU type indicator value higher than 5 are non-VCL NALUs and carry information like SEI, sequence and picture parameter set, Access Unit Delimiter etc. For example, NALU type 7 carries the sequence parameter set and type 8 carries the picture parameter set. As shown in Fig. 32, depending upon a particular delivery system and scheme non-VCL NALUs may or may not be present in the stream containing VCL NALUs. When non-VCL NALUs are not present in the stream containing VCL NALUs, the corresponding information can be conveyed by any external means in place in the delivery system.

8.5. HRD issues

In order to make sure that encoders do not generate a bit stream that is not compliant with the standard, a hypothetical reference decoder (HRD) model is provided. It provides model of a theoretical decoder that is assumed by an encoder while generating the bit streams. This model contains input coded picture buffer (CPB), an instantaneous decoding process and an output decoded picture buffer (DPB). It also includes the timing models—rate and time when the bytes arrive at the input of those buffers and when the bytes are removed from those buffers. An encoder must create the bit streams such that the CPB and DPB buffers of HRD do not overflow or underflow.

There are two types of conformance that can be claimed by a decoder—output order conformance and output timing conformance. To check conformance of a decoder, test bitstreams conforming to the claimed Profile and Level are delivered by a hypothetical stream scheduler to both HRD and the decoder under test. For output order conformant decoder, the values of all decoded pixels and order of the output pictures must be the same as those of HRD. For output timing conformant

Table 24
Profiles of H.264/MPEG-4 AVC standard and summary of tools included in each profile

Coding tools	Profile		
	Baseline	Main	Extended
I, P slices	X	X	X
CAVLC	X	X	X
Error resilience tools (FMO, ASO, Redundant slices)	X		X
Streaming/higher error resilience tools (SI, SP slices, Data partioning)			X
B slices		X	X
Interlaced coding tools (Frame, Field, PicAFF, MBAFF)		X	X
CABAC		X	

decoder, in addition, the output timing of the pictures must also be the same as that of HRD.

9. Profiles, levels and complexity

9.1. Details of Profiles and Levels

As discussed in Section 2.2, the standard currently includes three profiles. The profile structure can be best summarized with help of Table 24, which explicitly shows the tools categories as well as individual tools of the standard. Actually, to define an interoperability point, we need to specify not only a Profile but also a Level. A Level provides semantic constraints on parameters/options within a tool and is important as it indicates memory or processing limit. A decoder compliant with a particular Profile and Level must implement all the tools and constraints specified. An encoder may choose to use a smaller subset of tools specified in a Profile to generate and still be able to generate a bitstream compliant with a chosen Profile/Level.

Syntax of the standard also allows one to send three constraint flags in the bit stream: constraint_set0_flag, constraint_set1_flag and constraint_set2_flag. When constraint_set0_flag is set to 1 then it implies that the bit stream is compliant with Baseline Profile, when constraint_set1_flag is set to 1 then it implies that the bit stream is compliant with Extended Profile and when constraint_set2_flag is set to 1 then it implies that the bit stream is compliant with Main Profile. When two or more of these flags are set to 1 then it implies that the bit stream is compliant with two or more profiles. For example, if constraint_set0_flag and constraint_set2_flag both are set to 1 then the bit stream is compliant with both Baseline and Main profiles, i.e. the encoder can use only those coding tools that are common to both Baseline and Main profiles. In this case, the bitstream can be decoded by both Baseline and Main profile decoders. This way one can achieve cross-profile interoperability and generate bit streams that can be decoded by all AVC/H.264 decoders compliant to any of the profiles.

We now discuss details of how levels are defined for these profiles and provide examples of typical use. For decoder specification it is important to specify the processing power and the memory size needed for implementation. Picture size plays an important role in influencing those parameters. The standard has defined 15 different Levels tied mainly to the picture size and maximum compressed bit-rates parameters. Further, Levels also provide constraints on the number of reference pictures that can be used in the standard.

Table 25 shows Levels specified in the standard and some of the parameter values and the constraints specified. A decoder compliant with a particular Profile and Level must be able to decode a bitstream compliant to that Profile and Level as well as all the bitstreams compliant with that Profile and all the lower numbered Levels. Also, as the interlaced scan formats are used for half horizontal resolution (known as HHR,

Table 25
Levels to be supported in profiles of H.264/MPEG-4 AVC standard

Level number	Example of typical picture size	Typical frame rate for typical picture size	Maximum compressed bit-rate (for VCL)	Maximum number of reference frames for typical picture size
1	QCIF	15	64 kbps	4
1.1	QVGA (320 × 240)	10	192 kbps	3
	QCIF	30		9
1.2	CIF	15	384 kbps	6
1.3	CIF	30	768 kbps	6
2	CIF	30	2 Mbps	6
2.1	HHR		4 Mbps	
	(352 × 480)	30		7
	(352 × 576)	25		6
2.2	SD	15	4 Mbps	
	(720 × 480)			6
	(720 × 576)			5
3	SD		10 Mbps	
	(720 × 480)	30		6
	(720 × 576)	25		5
	VGA (640 × 480)	30		6
3.1	1280 × 720P	30	14 Mbps	5
	SVGA (800 × 600)	56		9
	1280 × 720P	60	20 Mbps	5
3.2	4VGA (1280 × 960)	45		4
4	All HD Formats		20 Mbps	
	(1280 × 720P)	60		9
	(1920 × 1080I)	30		4
	2k × 1k	30		4
4.1	HD Formats		50 Mbps	
	(1280 × 720)	60		9
	(1920 × 1080)	30		4
4.2	1920 × 1080	60	50 Mbps	4
5	2k × 1k	72	135 Mbps	14
	16VGA (2560 × 1920)	30		5
5.1	2k × 1k	120	240 Mbps	16
	4k × 2k	30		5

i.e. 352 × 480 or 352 × 576), SDTV and HDTV pictures; interlaced coding tools are applicable only at Levels 2.1–4.1.

Note that in the standard, Levels specify the maximum frame sizes in terms of only the total number of pixels/frame. The constraints on the number of pixels along horizontal and vertical dimensions are not specified except that the horizontal and the vertical sizes cannot be more than square root (8 × maximum frame size) Frame sizes shown are only some of the examples of typical frame sizes used in various applications at different Levels. A user has freedom to choose other frame sizes as long as total number of pixels/frame and horizontal and vertical sizes are with in the specified constraints.

Also, instead of specifying a maximum frame rate at each Level, maximum sample (pixel) rate, in terms of MB/s is specified. The maximum frame rates for the example typical picture sizes shown in the 3rd column of the table. If the picture sizes are smaller than the typical picture sizes in that column then the frame rates can be higher than those by up to the maximum of 172 fps.

Furthermore, the maximum number of reference frames is specified in terms of total memory. The maximum number of reference frames for example typical picture sizes in 2nd column, is shown by values in the 5th column. If, at a particular Level, the picture size is less than the one in that column then more number of reference frames, up to 16 frames, can be used for motion estimation.

9.2. Complexity implications

We now briefly discuss basic implications in terms of complexity [16,40] of individual tools of the standard.

- *Intra prediction:* Intra prediction in H.264/MPEG-4 AVC is relatively more complex than MPEG-2 video and MPEG-4 video (part 2) due to the nature of 4 × 4 block prediction, which needs to compute many different prediction candidates for various 4 × 4 blocks in a macroblock. Further, each direction of prediction uses specific combination of decoded neighbor pixels.
- *Motion estimation:* As in previous standards, motion estimation is a non-normative, performed only during encoding. Motion search complexity can be quite high due to possible of use of multiple reference frames, multiple block sizes, and $\frac{1}{4}$ pixel accuracy. Further, as in the case of previous standards, the complexity depends on whether motion estimation uses full search or reduced search, and if search starts with a zero motion estimate or with a prediction motion vector based on neighboring blocks.
- *Motion compensated inter prediction:* Motion compensated prediction will need to be performed on all block sizes such as 16 × 16, 16 × 8, 8 × 16, 8 × 8, 8 × 4, 4 × 8, and 4 × 4 which can contribute to a significant increase in memory bandwidth caused by the worst case of small block sizes. Further, $\frac{1}{4}$ pixel motion compensation requires 6 tap filters in horizontal and vertical direction for $\frac{1}{2}$ pixel positions, and a 2-tap horizontal, vertical or diagonal filter for $\frac{1}{4}$ pixel refinement. The 6 tap filters require data arrays input to the filter to be up to 5 more pixels extra in each direction. For example a 4 × 4 MC sub-block with quarter-pel motion requires reading a 9 × 9 array of pixels from reference picture in DRAM. Since with MC subblocks as small as 4 × 4, there can be as many as 16 subblocks for which MC is needed in one MB. With bi-prediction, two references may be simultaneously needed. Thus the memory bandwidth requirement of various modes available for inter prediction is somewhat excessive.
- *Transform/quantization:* The calculation of scale factor involves the use of tables and modulo-6 operation, among other things.
- *Loop filter:* The loop filter is specified to operate on the MBs in raster scan order. This is significant because the filter needs access to pixels in the neighboring decoded MBs above and to the left of the current MB, and in some decoder implementations there may be a problem to have the upper and left neighbor MBs available to the loop filter. Additional complexity of the deblocking filter is mainly due to the high adaptivity of the filter that requires conditional processing on the block edge. As a consequence, conditional branches are required in the most inner loops of the MB reconstruction process. This requires intensive branching in filtering operations and can be a challenge for parallel processing in certain implementations. Further, since the filter is in the coding loop, encoder implementations must exactly match mathematically with the filter specified in the standard. Another reason for the high complexity is the small block size employed for residual coding in the H.264 coding algorithm. With the 4 × 4 blocks and a typical filter length of 2 samples in each direction, almost every sample in a picture must be loaded from memory, either to be modified or to determine if neighboring samples will be modified. This was not the case for the H.263 loop filter or with MPEG-4 (part 2) post filter, which operates on an 8 × 8 block.
- *Mode decision:* Mode decision although an encoding issue tends to be substantially more complex due to the larger number of modes

possible for each MB in the AVC standard. Efficient mode decision may use RDOpt which requires multiple coding iterations on a macroblock basis followed by the determination of best RD tradeoffs.

- *Entropy coding:* Decoding of three types of entropy coding (Exp-Golomb, CAVLC, and CABAC) requires different architectures. CAVLC although more complex then the VLC methods of previous standards is still relatively lower in complexity as compared to CABAC, in which major reason for complexity is the serial nature of many operations and context formation.
- *Interlace coding:* In frame pictures with MB-AFF enabled, all MBs are organized into pairs called Super-MBs, with each pair consisting of a vertically adjacent pair of MBs. Each Super-MB is either field coded or frame coded. This vertical pairing and Super-MB level variability in format can be a challenge in decoding coded streams.

10. Summary

In this paper, we first presented an overview of video coding as per the H.264/MPEG-4 AVC standard, introducing tools in this standard and briefly discussed profiles of the standard. Several of the important coding tools were then discussed in detail. We then presented results of our study comprising of systematic evaluation by experiments, the contribution to coding performance of many of these tools these tools, as well as certain combinations of these tools. We then discussed system tools, details of profiles and levels, and the important issue of coding complexity. In terms of coding performance the significant findings of our study are as follows:

- As in the case of earlier standards, B-pictures remain a significant tool. Coding with 1 B-picture provides a gain of 10–24%, and with 2 B-pictures, 14–34%, over the case of no B-pictures. When coding of movie scenes, the gain with B-pictures was slightly lower but still–quite significant, i.e. with 1 B-picture, it is in the range of 9–19%, and with 2-B pictures, it is in the range of 12–29%. While it is true that the stated B-picture gains are accompanied with small reduction in SNR due to slightly higher quantizer for such pictures, due to temporal masking and noise averaging properties of B-pictures, these SNR differences are not visually perceptible.
- Although strictly an encoding issue, due to a plethora of coding modes, good mode decision is quite important and ranks second, right next to B-pictures. Actually, a good mode decision is even more important with more (e.g. 2) B-pictures as compared to the case of fewer (e.g. 1) B-pictures. In comparison to non-RD optimized mode decision, RD optimized mode decision provides, for the case of 1 B-picture coding, a gain of 5–13% (7.5% on the average), and for the case of 2 B-picture coding, a gain of 10–22% range (14% on the average). The RD optimized mode decision of JM, while it performs well, it explicitly requires multiple coding iterations for each macroblock and thus presents difficulties in realizing a practical, fast encoder.
- The gain from CABAC can also be important and varies depending on the properties of sequence and coding bit-rates. For the test sequences and conditions we chose as being representative of normal coding, the average gain, for 1 B-picture coding was found to be 6.8%, and, with 2 B-pictures, the average gain was slightly higher, 7.2%. The overall average gain was thus found to be 7%. For coding of movie scenes, the gain was found to be also similar. Quite likely, in coding less detailed sequences or when coding with higher quantizers in general, gains may be higher.
- Gains from multiple reference pictures are typically small but sometimes can be higher as the performance of this tool is very scene dependent. Further, multiple reference pictures are effective with fewer B-pictures. For instance, in the case of 1 B-picture coding, the use of 3 reference pictures as compared to 1 reference picture, provides a gain of 2–8% (5% average), while the use of 5 references

pictures can provide 2–10% gain. However, for the case of 2 B-picture coding, use of 5 reference pictures over 1 reference picture, provides a gain of only 1–6% (3% average). Overall the gains from additional B-pictures are higher than that from reference pictures, thus 2 B-picture coding with 1 reference picture performs much better than 1 B-picture coding with 5 reference pictures!

- The impact of loop filter in terms of reduction of coding bit-rate is relatively small (0.5%–2%), however it has a bigger impact on perceived visual quality. Loop filtering results in smoothing of block boundaries but since transform coding block sizes are only 4×4, due to the nature of the support used by the filter, especially at lower bit-rates, some fine details may also be lost. The loop filter, smoothes out blockiness and coding noise and generally produces more visually acceptable quality. Proper regulation of the loop filter to produce even improved blockiness/visual quality trade-offs by encoder parameter selection, may be possible.
- For motion compensation, use of no smaller than 8×8 blocks as compared to all available block sizes for the case of 1 B-picture, results in loss of 1.75% on the average, and for the case of 2 B-pictures, loss of 2.5% on the average. In fact, the results are very scene dependent and range from loss of around 1% on slower motion scenes to as much as 5% on more complex fast moving scenes. Note that these results were obtained for the case of RDOptimized mode decision. Besides bit-rate difference, use of small blocks may also provide some improvement in visually quality.
- For interlaced video coding, picture adaptive frame/field coding provides a gain of 6–32% over frame coding, while macroblock adaptive frame/field coding provides a gain of 11–34% gain. Macroblock adaptive frame/field coding is however more complex. Since picture adaptive frame field coding can offer most (if not all) of the gains of macroblock adaptive frame field coding, it offers a good trade-off. Performance of other interlaced tools such as frame coding or field coding is very dependent on motion and details in the sequence but is lower than picture adaptive frame/field coding.
- Other tools as well as related encoding options seem to make very little difference to overall coding efficiency.

Acknowledgements

We would like to thank Dr. Karsten Suehring for the public domain JM6.1e software used in our experiments. We would like to thank Dr. Gary Sullivan for his thorough and careful review and many helpful suggestions for improvement. We would also like to thank Dr. Shawn Zhong and Dr. Wade Wan for their very helpful review of earlier drafts of the paper.

References

[1] W. K. Cham, Family of order-4 four level orthogonal transforms, Electron. Lett. 19 (21) (October 1983) 869–871.

[2] W.K. Cham, R.J. Clarke, Simple high efficiency transform for image coding, in: Proceedings of the International Picture Coding Symposium, Davis, CA, 1983, pp. 66–67.

[3] X. Chen, R. Eifrig, A. Luthra, K. Panusopone, Coding of an arbitrary shaped interlaced video in MPEG-4, in: Proceedings of the IEEE International Conference on Accoustics, Speech and Signal Processing (ICASSP'99), 1999 pp. 3121–3124.

[4] S. (X.) Chen, A.G. MacInnis, SEI message for the film mode hint in JVT codec, JVT-E078, Joint Video Team of ISO/IEC MPEG and ITU-T VCEG, Geneva, October 2002.

[5] R.J. Clarke, Transform Coding of Images, Academic Press, New York, 1985.

[6] D. Green, G. Sullivan, Field repetition and timing indications, JVT-E122, Joint Video Team of ISO/IEC MPEG and ITU-T VCEG, Geneva, October 2002.

[7] A. Hallapuro, M. Karczewicz, H. Malvar, Low complexity transform and quantization—Part I: basic implementation, JVT-B38, Joint Video Team of ISO/IEC MPEG and ITU-T VCEG, January 2002.

[8] B.G. Haskell, A. Puri, A.N. Netravali, Digital Video: An Introduction to MPEG-2, Chapman & Hall/Kluwer Academic, New York, 1997, ISBN: 0-412-08411-2.

[9] ISO/IEC JTC1/SC29, Coding of moving pictures and associated audio for digital storage media up to about

1.5 Mbit/s, ISO/IEC 11172-2, International Standard, November 1992.

[10] ISO/IEC JTC1/SC29, Generic coding of moving pictures and associated audio, ISO/IEC 13818-2, Draft International Standard, November 1994.

[11] ISO/IEC JTC1/SC29, Coding of audio-visual objects, ISO/IEC 14496-2, International Standard:1999/Amd1:2000, January 2000.

[12] Joint Model Editing Committee (G. Sullivan, T. Wiegand, K.-P Lim), Joint model reference encoding methods and decoding concealment, JVT-I049, Joint Video Team of ISO/IEC MPEG and ITU-T VCEG, September 2003.

[13] JVT Software Implementation Group (K. Suehring, et al.), JM6.1e Reference Software, March 2003.

[14] JVT Editors (T. Wiegand, G. Sullivan, A. Luthra), Draft ITU-T Recommendation and final draft international standard of joint video specification (ITU-T Rec.H.264|ISO/IEC 14496-10 AVC), JVT-G050r1, Geneva, May 2003.

[15] A. Kaup, H. Mooshofer, Performance and complexity analysis of rate constrained motion estimation in MPEG-4, in: Proceedings of the SPIE Multimedia Systems and Applications, Boston, September 1999, pp. 202–211.

[16] V. Lappalainen, A. Hallapuro, T. D. Hamalainen, Complexity of optimized H.26L video decoder implementation, IEEE Trans. Circuits Systems Video Technol. 13 (7), (July 2003) 717–723.

[17] A. Luthra, P. Topiwala, Overview of H.264/AVC video coding standard, in: Proceedings of the SPIE—Applications of Digital Image Processing XXVI, San Diego, Vol. 5203, August 2003.

[18] D. Marpe, H. Schwarz, T. Wiegand, Context-based adaptive binary arithmetic coding in the H.264/AVC compression standard, IEEE Trans. Circuits Systems Video Technol. 13 (7), (July 2003) 620–636.

[19] K. Panusopone, X. Chen, R. Eifrig, A. Luthra, Coding tools in MPEG-4 interlaced video, IEEE Trans. Circuits Systems Video Technol. 10 (5), (August 2000) 755–766.

[20] A. Puri, Efficient motion compensated coding for low bitrate video applications, Ph.D. Thesis, February 1988.

[21] A. Puri, Conditional motion-compensated interpolation and coding, in: Second International Workshop on 64 Kb/s coding of moving video, Hannover, West Germany, September 1989, pp. 1.5.

[22] A. Puri, Multi-frame conditional motion-compensated interpolation and coding, in: Picture Coding Symposium, Cambridge, MA, March 1990, pp. 8.3.1–8.3.2.

[23] A. Puri, Video coding using the MPEG-1 compression standard, Proceedings of the International Symposium of Society for Information Display, Boston, May 1992. pp. 123–126.

[24] A. Puri, Video coding using the MPEG-2 compression standard, in: Proceedings of the SPIE EI—Visual Communication and Image Processing, SPIE, Boston, vol. 1199, November 1993, pp. 1701–1713.

[25] A. Puri, R. Aravind, On comparing motion-interpolation structures for video coding, in: Proceedings of the SPIE Visual Communication and Image Processing, Lausanne, Switzerland, October 1990, pp. 1560–1571.

[26] A. Puri, R. Aravind, Motion-compensated video coding with adaptive perceptual quantization, IEEE Trans. Circuits Systems Video Technol. 1 (4), (December 1991) 351–361.

[27] A. Puri, R. Aravind, B. Haskell, Adaptive frame/field motion compensated video coding, Signal Processing: Image Communication, 5 (1–2) (February 1993) 39–58.

[28] A. Puri, R. Aravind, B. G. Haskell, R. Leonardi, Video coding with motion-compensated interpolation for CD-ROM applications, Signal Processing: Image Communication, 2 (2), (August 1990) 127–144.

[29] A. Puri, T. Chen, Multimedia Systems, Standards, and Networks, Dekker, New York, 2000, ISBN: 0-8247-9303-X.

[30] A. Puri, H.-M. Hang, D. L. Schilling, An efficient block-matching algorithm for motion compensated coding, in: Proceedings of the IEEE International Conference on Acoustics, Speech, Signal Processing (ICASSP'87), Dallas, April 1987, pp. 25.4.1–25.4.4.

[31] A. Puri, H.-M. Hang, D. L. Schilling, Motion-compensated transform coding based on block motion tracking algorithm, Proceedings of the IEEE International Conference Communication (ICC'87), Seattle, June 1987, pp. 5.3.1–5.3.5.

[32] A. Puri, H.-M. Hang, D. L. Schilling, Interframe coding with variable block-size motion compensation, in: Proceedings of the IEEE Global Communication Conference (GLOBECOM'87), (Tokyo) November 1987, pp. 65–69.

[33] A. Puri, R. L. Schmidt, B. G. Haskell, Improvements in DCT based video coding, in: Proceedings of the SPIE EI-Visual Communication and Image Processing, San Jose, February 1997.

[34] A. Puri, R. L. Schmidt, B. G. Haskell, Performance evaluation of the MPEG-4 visual coding standard, in: Proceedings of the SPIE EI-Visual Communication and Image Processing, San Jose, January 1998.

[35] H. Schwarz, T. Wiegand, An improved MPEG-4 coder using lagrangian coder control, VCEG-M49, Video Coding Experts Group, March 2001.

[36] Video Test Model Editing Committee, MPEG-2 video test model 5 (TM5), ISO/IEC JTC1/SC29/WG11 N0400, April 1993.

[37] T. Wiegand, B. Girod, Lagrange multiplier selection in hybrid video coder control, in: Proceedings of the IEEE International Conference on Image Processing (ICIP'01), Greece, 2001.

[38] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, G. Sullivan, Rate-constrained coder control and comparison of video coding standards, IEEE Trans. Circuits Systems Video Technol. 13 (7) (July 2003) 688–703.

[39] T. Wiegand, G. J. Sullivan, G. Bjoentegaard, A. Luthra, Overview of the H.264/AVC video coding standard, IEEE Trans. Circuits Systems Video Technol. 13 (7) (July 2003) 560–576.

[40] A. Wise, R. Whiten, Y. Nemouchi, P. B. Thomas, Model for estimating prediction bandwidth for H.26L, JVT-E93, Joint Video Team of ISO/IEC MPEG and ITU-T VCEG, October 2002.

[41] P. Yin, H.-Y. Tourapis, A. Tourapis, J. Boyce, Fast mode decision and motion estimation for JVT/H.264, in: IEEE International Conference on Image Processing (ICIP'03), 2003.