Signal Processing: *Image Communication* 24 (2009) 637–650

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Video transcoding from H.264/AVC to MPEG-2 with reduced computational complexity

Sandro Moiron[a], Sérgio Faria[a,b], António Navarro[a,c], Vitor Silva[a,d], Pedro Assunção[a,b,*]

[a] *Instituto de Telecomunicações, Pólo II FCTUC, Pinhal de Marrocos, 3030-290 Coimbra, Portugal*
[b] *Polytechnic Institute of Leiria, Leiria, Portugal*
[c] *University of Aveiro, Aveiro, Portugal*
[d] *University of Coimbra, Coimbra, Portugal*

A R T I C L E I N F O

Article history:
Received 8 June 2008
Received in revised form
8 June 2009
Accepted 10 June 2009

Keywords:
Video transcoding
H.264/AVC
MPEG-2

A B S T R A C T

This paper addresses video transcoding from H.264/AVC into MPEG-2 with reduced complexity and high rate-distortion efficiency. While the overall concept is based on a cascaded decoder–encoder, the novel adaptation methods developed in this work have the advantage of providing very good performance in H.264/AVC to MPEG-2 transcoding. The proposed approach exploits the similarities between the coding tools used in both standards, with the objective of obtaining a computationally efficient transcoder without penalising the signal quality. Fast and efficient methods are devised for conversion of macroblock coding modes and translation of motion information in order to compute the MPEG-2 coding format with a reduced number of operations, by reusing the corresponding data embedded in the incoming H.264/AVC coded stream. In comparison with a cascaded decoder–encoder, the fast transcoder achieves computational complexity savings up to 60% with slightly better peak signal-to-noise ratio (PSNR) at the same bitrate.

1. Introduction

Nowadays, the H.264/AVC video standard [1] presents a much better compression performance than its recent predecessors, namely MPEG-2 [2]. However, the MPEG-2 [3] video standard is still the most used video compression format and its widespread use in both professional and user equipment is expected to last for several years ahead, particularly in digital television (DTV), personal video recorders (PVR) and digital versatile disc (DVD). Due to its higher compression efficiency, H.264/AVC is increasingly gaining acceptance in multimedia applications and services, such as high definition digital television (HDTV), mobile TV (MTV) and the internet. The use of diverse coding standards at the same time, in both the professional and consumer market, leads to interoperability problems, because the same type of source material may be available in a format which is not compatible with the target equipment. Furthermore it is not likely that technology migration, in both professional and user equipment, happens in such a short period of time that problems arising from co-existence of both standards can be ignored. Therefore, transcoding from H.264/AVC to MPEG-2 format is necessary to maintain backward compatibility and ease technology migration.

A possible application scenario, where such type of transcoding is potentially useful, is depicted in Fig. 1. Both the service provider and the network operator benefit from using H.264/AVC for content storage and delivery because it allows significant savings in storage capacity and network bandwidth, but at the user premises the MPEG-2 format is necessary because of legacy equipment.

* Corresponding author at: Instituto de Telecomunicações, Delegação de Leiria, Morro do Lena - Alto Vieiro, Leiria, Portugal. Tel.: +351 244820300; fax: +351 244820310.
E-mail addresses: assuncao@estg.ipleiria.pt, amado@co.it.pt (P. Assunção).

0923-5965/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2009.06.001

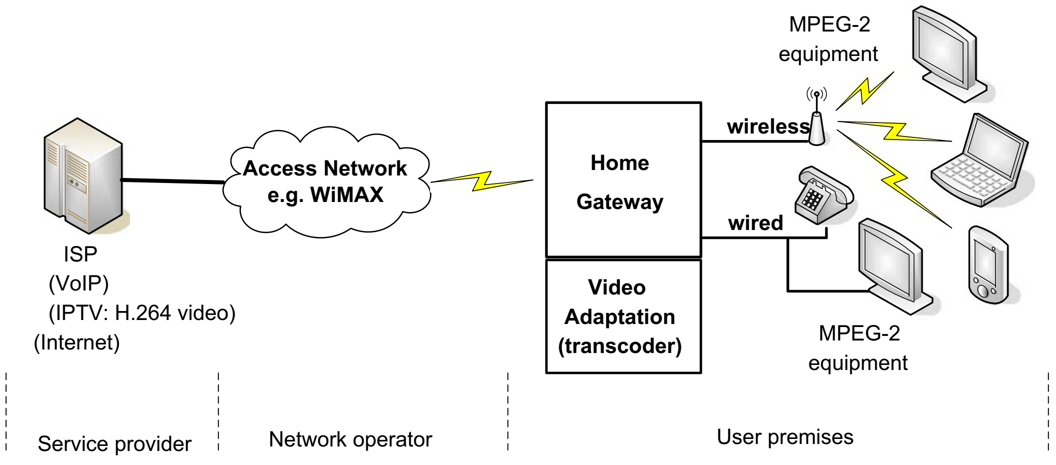


Fig. 1. An application scenario.

Therefore, a transcoding functionality must be included in the home gateway to perform the necessary format adaptation. Since this might be one of the multiple processes running on the same hardware platform, it is important to minimise the computational complexity, subject to acceptable quality constraints.

In the recent past, much research effort has been devoted to develop efficient transcoding from MPEG-2 into H.264/AVC, in order to migrate legacy video content to the new format [4–6]. However, too little effort has been devoted to the problem of backward compatibility [7] and a similar type of transcoding was mostly addressed in [8–12]. In [7], the authors highlight some technical problems and research directions, whereas a similar type of transcoding was proposed in [10–12,8]. However, these transcoders are targeted for the baseline profile, i.e., only P frames are addressed, and they do not deal with multiple reference frames in H.264/AVC. While in [11] the authors extend their previous method [10] of motion refinement by using a dynamic search window to improve transcoding efficiency, in [12] the authors use a fixed window of 2 pixels for motion vector refinement. A transform domain approach was used in [9] but the results show a quality loss of about 10 dB, which is definitely too much for any practical purpose. In our recent work, a transcoding architecture was proposed for efficient conversion between H.264/AVC and MPEG-2 [13]. Although not all block coding modes were processed by fast conversion methods, its efficiency was shown to be suitable for several applications [14].

In this paper we propose a complete transcoding architecture, expanding the previous methods to all H.264/AVC coding modes, including B frames and specific modes (e.g., unrestricted motion vectors). Table 1 shows an approximate comparison with other mechanisms proposed in the literature. Overall, the proposed fast methods are capable of reducing the computational complexity up to 60%, in comparison with a reference transcoder comprised a cascaded decoder–encoder. Furthermore, by reusing the most efficient coding decisions embedded in the H.264/AVC input stream, the proposed fast transcoder provides slightly better PSNR than the reference transcoder.

Table 1
Comparison of transcoding mechanisms.

Transcod. mechanism	Proposed	Previous work
Functional transcoding architecture	Explicit full description containing all modules	Partial textual description
Pixel domain vs transform domain processing	Pixel domain for inter and transform domain for some intramodes	Either pixel domain or transform domain for selected modes
Fast transcoding of B slices with multiple references	New references and matching MV scaling methods are devised	Not addressed
Transcoding of MB with multiple partitions	MV selection based on partition with minimum SSD	Not addressed
MV conversion from uni to bidirectional prediction	A mirroring function is used to generate the second MV	Not addressed
Unrestricted MV	A mirroring function is used to generate the alternative MV	Not addressed
MV scaling and refinement	Scaling based on temporal distance and half-pixel refinement	Scaling based on temporal distance and larger refinement windows
Intratransform domain transcoding	Efficient method devised and applied to 16 × 16 MBs	Fast methods with low quality results

A trivial implementation of a transcoder is a cascade of an H.264/AVC decoder and an MPEG-2 encoder. However, such a scheme completely ignores the

H.264/AVC encoding information embedded in the bitstream, which is the result of smart rate-distortion (RD) decisions, aiming at encoding each block with the highest possible efficiency. By using such a transcoder, the H.264/AVC decoded frames have to be fully MPEG-2 encoded, as if no previous coding information existed. In order to reduce this unnecessary complexity, the approach described in this paper aims at simplifying the MPEG-2 encoding process by reusing the information contained in the H.264/AVC bitstream. In particular, this paper proposes fast and efficient conversion methods for interframe coding modes, capable of achieving a significant reduction in computational complexity with marginal objective quality reduction, when compared with full recoding. Transform domain conversion of intracoding modes is also presented and discussed, but not implemented in the test transcoder. Due to the highly non-linear nature of most H.264/AVC intraprediction modes, the computational complexity of intraconversion in the transform domain is higher than that of pixel domain approach for most of these modes. Since intraconversion in the DCT domain was not included in the test transcoder, this topic is discussed in Section 5 and then described with more detail in Appendix A.

The paper is organised as follows. After this introduction, where the motivation and the context of the proposed work are described, Section 2 points out the relevant differences between MPEG-2 and H.264/AVC, which are important for subsequent sections. Section 3 presents the functional architecture for the proposed transcoding scheme. Section 4 describes the fast conversion methods proposed for transcoding H.264/AVC video streams into MPEG-2, and in Section 5 the transform domain intraconversion is discussed with reference to the particular cases described in Appendix A. Finally, Section 6 presents the experimental results along with a critical discussion of the relevant issues and Section 7 concludes the paper.

2. H.264/AVC and MPEG-2: relevant differences

The H.264/AVC standard introduces a relevant number of improved coding tools in comparison to MPEG-2, as short listed in Table 2. Several types of intraprediction modes, macroblock (MB) partition in diverse subblocks, integer transform, motion accuracy up to quarter-pixel, unrestricted boundaries for motion vectors and the use of up to 16 reference frames, are examples of new tools which significantly contribute to improve the coding efficiency over previous standards. Note that MPEG-2 only uses half-pixel accuracy in motion estimation/compensation (ME/MC), the motion vectors are constrained to the frame boundaries and only one or two reference frames can be used for P and B frames, respectively. Moreover, MPEG-2 does not allow the same MB partitioning modes neither intraprediction. In the case of MPEG-2, ME is computed for either $16{\times}16$ or 16×8 blocks whereas H.264/AVC allows a wider set of block partitions, such as 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 and 4×4.

Table 2
Main differences between H.264/AVC (main profile) and MPEG-2 (main profile).

Features	H.264/AVC	MPEG-2
Motion estimation accuracy	1/4 pel	1/2 pel
Motion vectors	Unrestricted boundaries	Restricted to frame boundaries
Reference frames	Up to 16	1 for P pictures up to 2 for B pictures
Macroblock partitions	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4	16×16, 16×8 (interlace)
Spatial prediction type	Nine intraprediction modes	None
Transform type	Integer transform	Fixed point DCT
Entropy coding	CAVLC / CABAC	VLC

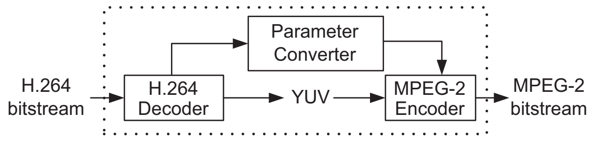


Fig. 2. Modified transcoding architecture.

3. H.264/AVC to MPEG-2 transcoding architecture

The trivial transcoding architecture is based on a cascade of an H.264/AVC decoder with an MPEG-2 encoder. This is a straightforward sequential process capable of achieving good quality results, though at maximum computational complexity cost. Such a scheme completely discards the H.264/AVC encoding information embedded in the bitstream and performs full motion estimation, which is computationally intensive and consumes most of the processor resources for encoding. Nevertheless the cascaded transcoder, referred to as the reference transcoder, is useful for comparing its performance with other architectures and algorithms.

By exploiting the coding information embedded in the incoming bitstream, it is possible to reuse several H.264/AVC coding parameters at the MPEG-2 encoder with little additional computational effort. Therefore, the computational complexity of the MPEG-2 encoder module can be greatly reduced, as compared with a standard full encoder. Fig. 2 shows the high level structure of the transcoding architecture used in this paper. It is based on a cascaded transcoder, where the decoder and the encoder are modified implementations of the H.264/AVC JM13.2 [15] and MPEG-2 video v1.2 [16], and an additional conversion module where the main adaptation functions operate.

Fig. 3 presents a more detailed structure of the proposed transcoding architecture, where a functional module is included between the H.264/AVC decoder and the MPEG-2 encoder. Such module acts as an adaptation

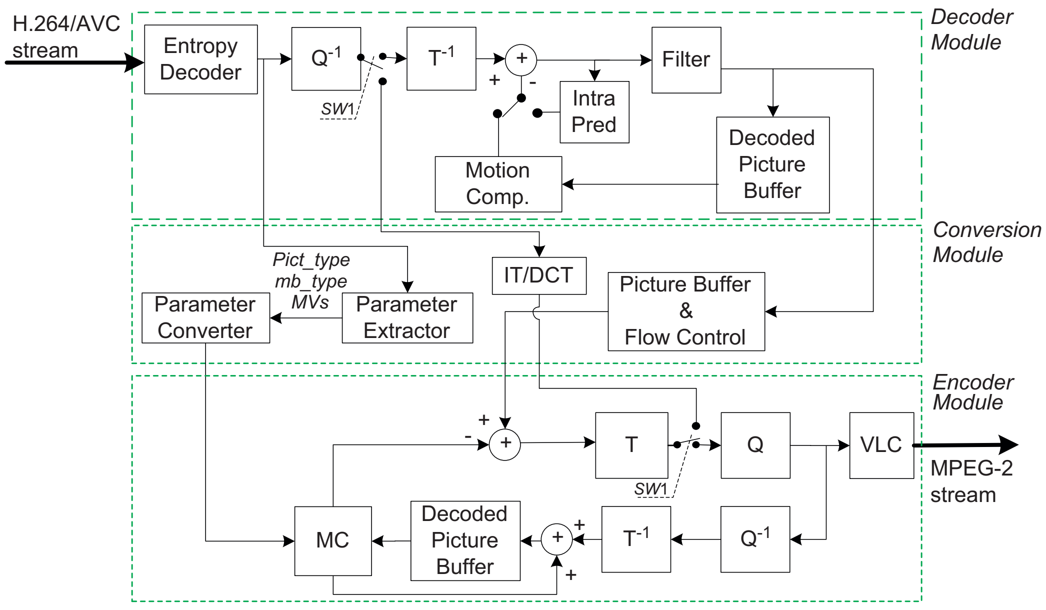


Fig. 3. Detailed transcoding architecture.

interface, by reprocessing the coding parameters received from the H.264/AVC decoder and feeding the MPEG-2 encoder, in order to short-cut some typical computation intensive operations. Since motion estimation is one of the most processor consuming tasks at the encoder, reducing the complexity of this function is one of the objectives to be pursued, in order to achieve low computational complexity transcoding [17].

When a video signal is encoded with the H.264/AVC its characteristics are analysed in detail and each MB is efficiently encoded in regard to the best rate-distortion point. Such optimal RD coding modes may be partially reused by the MPEG-2 encoding module, after being adapted in the *conversion module*. Different types of parameters, such as MB mode information, picture type, motion information, etc., are then extracted from the bitstream at the H.264/AVC decoder, and these parameters are further processed and converted to an MPEG-2 compatible format. The parameter extraction and adaptation procedure can also be used in intraframe coding, by converting the integer transform (IT) coefficients into discrete cosine transform (DCT) coefficients [18].

3.1. Decoder module

In Fig. 3, the H.264/AVC decoder module extracts coding parameters and the video signal itself from the input stream, for both the conversion and MPEG-2 encoder modules. Therefore the input video stream is decoded and the extracted information is provided to the conversion module for further processing and adaptation to the MPEG-2 format. The coding parameters consist of MB and partition types, motion vectors, reference images, prediction modes, transform coefficients, etc.

3.2. Conversion module

The conversion module is responsible for interfacing between the decoder and the encoder, by converting a set of H.264/AVC parameters into MPEG-2 format. These parameters are analysed and converted when they are useful for reducing computational complexity. Furthermore, this module is divided into three independent submodules:

- the parameter converter—exploits the coding parameters, such as motion vectors and partition types;
- the picture buffer—handles the extracted parameters and the uncompressed video data flow from the decoder to the encoder;
- the IT/DCT coefficient conversion—converts the H.264/AVC IT coefficients directly to MPEG-2 DCT in the transform domain.

The picture buffer is implemented as a ring buffer and its size is defined according to either the available memory or a related cost function. A size of one single frame should be avoided in platforms with multi-core processors, because one of the processes will be always waiting, either to store in the buffer or to read from it. The IT/DCT coefficient conversion in the transform domain is addressed in Section 5, while the interframe prediction conversion modes is presented in Section 4.

3.3. Encoder module

The MPEG-2 encoder module is a modified version of a standard encoder, which has the additional capability of receiving a set of parameters from the conversion module and also that of providing several control signals to some encoding functions, namely the motion estimation. The extracted motion parameters, after being processed by the *parameter converter*, are delivered to the *motion estimation* block which allows skipping most of the search algorithm.

The motion estimation function checks whether the current MB can be efficiently encoded with the H.264/AVC converted information and, if this is the case, the converted motion vector is used to build the best MB prediction. For those prediction modes which do not have enough similarities to be worth fast transcoding, the transcoder switches to classical motion estimation, which is the case of most intraprediction modes.

4. Inter-MB conversion

As mentioned before, for inter-MB conversion, the proposed method extracts the motion vector information and the MB types from the H.264/AVC bitstream. The reuse of such parameters in the MPEG-2 encoder allows to bypass motion estimation, which accounts for up to 70% of the encoder complexity [19]. This section describes coding mode conversion techniques for the MB types defined in P and B slices, addressing in particular those MB types that have similar characteristics and higher level of compatibility between both standards. The proposed MB conversion methods are also discussed along with possible solutions to overcome the inherent standard incompatibilities.

Fig. 4 describes the operation of the transcoder, highlighting the modifications which change the operation of a standard encoder. As the figure shows, the encoding process is performed on a MB basis and the decision to select either fast conversion or classical encoding is based on the constraints imposed by the coding modes. If such constraints exclude a MB from fast

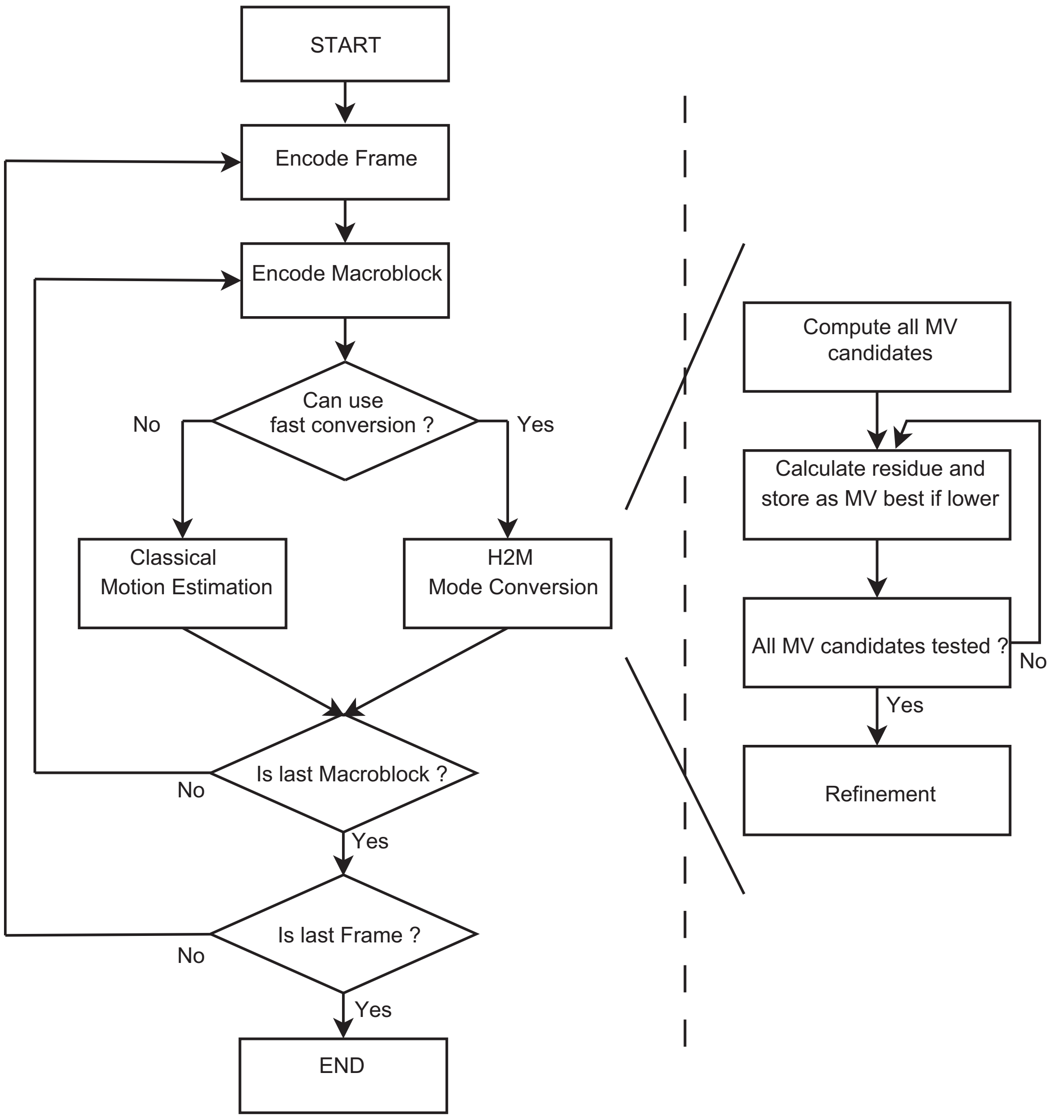

START
Encode Frame
Encode Macroblock
Can use fast conversion ?
No
Yes
Classical Motion Estimation
H2M Mode Conversion
Is last Macroblock ?
No
Yes
Is last Frame ?
No
Yes
END
Compute all MV candidates
Calculate residue and store as MV best if lower
All MV candidates tested ?
No
Yes
Refinement

Fig. 4. Functional operation mode for interframe conversion.

conversion, then full motion estimation is used without any extra input information. The remaining cases are described in the following sections.

4.1. 16×16 *SKIP*

Conversion of H.264/AVC SKIP MBs to MPEG-2 is not straightforward because the definition of a SKIP MB is different in each standard, thus full compatibility does not exist and direct conversion is not possible. However, since in P slices the H.264/AVC SKIP MB represents either constant motion or static areas, in MPEG-2 it only represents static areas, i.e., $MV = 0$, then MPEG-2 SKIP mode is actually a subset of its counterpart in H.264/AVC.

In the case of B slices, both standards define the SKIP mode for either constant motion or static areas. Thus, fast mode conversion is enabled and becomes more efficient than in P slice case. As it will be shown in Section 6, the computational complexity savings for this type of mode conversion are very impressive, because the number of processing operations necessary to accomplish a transcoded MB is very low.

In both P and B slice types, MPEG-2 constraints the use of SKIP mode in the first and last MB of each slice. Such a constraint is particularly relevant when using small frame sizes, because MPEG-2 imposes each slice to start and end in the same MB row, and this has the effect of increasing the number of excluded MBs for this SKIP mode conversion because an equivalent constraint does not exist in H.264/AVC.

4.2. 16×16 *predicted*

Conversion of 16×16 MBs from H.264/AVC to MPEG-2 can be achieved with low computational complexity because this MB size exists in both standards. Motion estimation is performed in a similar way in both cases, which allows the reuse of motion information from the H.264/AVC bitstream, thus avoiding computation of the full motion estimation.

The mode conversion of this MB type is constrained by some coding tools introduced in H.264/AVC that raise incompatibilities between both standards, such as those arising from multiple reference pictures that can be used in H.264/AVC (e.g., Fig. 5), where each slice can use up to 16 reference pictures, whereas in MPEG-2 this is restricted to a maximum of 2 adjacent coded pictures. When converting from H.264/AVC to MPEG-2, all motion vectors pointing to reference frames, which are temporally located farther than the adjacent ones, should be converted in order to make them compliant with MPEG-2 reference frames. This conversion method is described in Section 4.4, where scaling of H.264/AVC motion vectors is addressed.

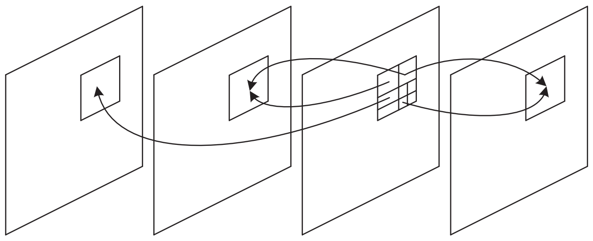


Fig. 5. Multi-reference prediction.

Along with the previous constraint, the *unrestricted motion vectors* used in the H.264/AVC standard allows the use of motion vectors pointing to areas outside the frame boundaries. Since this is not supported in MPEG-2, a specific conversion method was developed, in order to reuse such motion vectors, as described in Section 4.5.

Finally, a last constraining factor in conversion of this coding mode is the pixel accuracy used in both standards. While H.264/AVC use motion vectors with quarter pixel accuracy, MPEG-2 only allows half pixel. Therefore accuracy conversion is mandatory and it is performed by rounding the motion vector to the nearest half-pixel position.

4.3. *MB partitioning*

The H.264/AVC standard makes use of MB partitioning into blocks of sizes (16×8, 8×16 and 8×8) and then, each block can be further partitioned into several smaller subblocks of sizes (8×4, 4×8 and 4×4). Such partitioning scheme of 16×16 MBs into smaller blocks improve the prediction efficiency because, in general, a residue of lower energy is achieved for each partition. Since each 8×8 partition might have its own reference picture, a single H.264/AVC MB may use up to four different reference pictures.

Conversion of H.264/AVC partitioned MBs is implemented by merging all partitions into a unique 16×16 MPEG-2 MB, as illustrated in Fig. 6. The motion vector of each partition is used to compute a prediction error of the 16×16 MPEG-2 MB and the criterion to select the best candidate is the lowest sum of squared differences (SSD). Such a process is described as follows: the motion vectors (MV) from the set P of H.264/AVC MB partitions (p_i) are tested as MPEG-2 predictions and the best candidate (MV_{best}) is chosen according to a minimum residue criterion, the SSD, given by the solution of the following equation:

$$MV_{best} = \arg\min_{i \in P} SSD[MV(MBp_i)] \tag{1}$$

4.4. *MV scaling*

As mentioned before, in the H.264/AVC standard a MB can use up to 16 reference frames for motion estimation, while MPEG-2 uses only one or two, according to the picture type P or B, respectively. Therefore reuse of H.264/AVC motion vectors without further processing is only possible in the particular cases where the H.264/AVC reference frames are temporally located in MPEG-2 compatible positions. Since in general this is not the case, the original H.264/AVC motion vectors need to be scaled. This is shown in Fig. 7, where two possible cases of multiple references in H.264/AVC are converted into MPEG-2 pictures of type B and P, respectively. This type

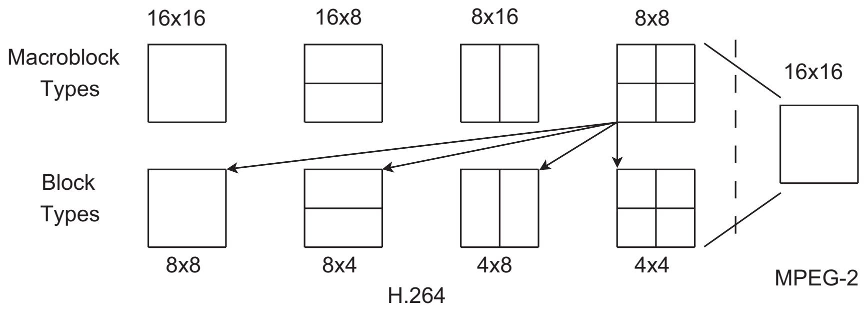


Fig. 6. Macroblock partition types.

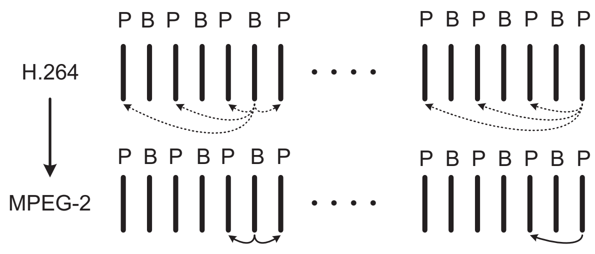


Fig. 7. Motion vector conversion for P and B slices.

of conversion assumes that motion is constant between the two frames used as references in the original and transcoded video streams.

Eqs. (2) and (3) are used to obtain the new motion vector MV_{MPEG2} from the original one ($MV_{H.264}$). These motion vectors are defined as the number of pixels that the reference block is displaced from the current one in the left/right and up/down directions. By using Eq. (2), the motion vector $MV_{H.264}$ is resized according to the temporal distance Δt between the current frame C_n and the reference frames, defined by their temporal indices $H.264ref_n$ and $MPEG2ref_n$, respectively,

$$MV_{MPEG2} = round\left(\frac{MV_{H.264}}{\Delta t}\right) \tag{2}$$

where

$$\Delta t = \frac{C_n - H.264ref_n}{C_n - MPEG2ref_n} \tag{3}$$

For example, if the following encoding sequence is used: IBPBPBPBP, with the frame numbering: 012345678 and $C_n = 8$, $H264ref_n = 2$ and $MPEG2ref_n = 6$, then $\Delta t = (8-2)/(8-6) = 3$. This means that the H.264/AVC motion vector should be resized by a factor of 3, in order to be used with an MPEG-2 reference frame.

The round operator in Eq. (2) ensures that the resulting motion vector is rounded to the nearest half-pixel position. In Eq. (3), $\Delta t = 1$ when both H.264/AVC and MPEG-2 reference frames are temporally coincident, which means that motion vector rescaling is not required.

4.5. MV conversion from uni to bidirectional prediction

In general, a bidirectional MB prediction produces a lower residue than unidirectional. Therefore conversion of a unidirectional prediction into a bidirectional one should be implemented in order to achieve better transcoding efficiency. A method to convert source H.264/AVC unidirectional motion vectors into MPEG-2 bidirectional ones is devised as follows. Assuming a constant movement of the MB area, the corresponding position in the opposite temporal direction of the current reference picture is estimated and a reverse motion vector is obtained by flipping the source motion vector in both axis, i.e., $MV^{inv}_{MPEG2} = -MV_{MPEG2}$. Then by combining this reverse motion vector with the original one, a bidirectional motion vector MV can be obtained (Eq. (4)). This process is shown in Fig. 8

$$MV = \begin{cases} MV_{MPEG2} \\ MV^{inv}_{MPEG2} \end{cases} \tag{4}$$

The candidate motion vector is selected from either unidirectional or bidirectional predictions according to the minimum residue criterion. This method can only be applied in B slices where the prediction direction can be forward, backward or both.

A similar method is used to deal with the special case of unrestricted motion vectors. In H.264/AVC these type of vectors point to blocks containing image areas outside the picture boundaries. This is particularly useful in coding sequences with camera panning, because it allows efficient motion estimation of image boundary MBs, by using a virtual reference located beyond the frame limits. Since this is not supported in MPEG-2, such motion vectors may introduce a significant burden in the mode conversion process because almost full search is necessary to find a new compliant motion vector. Fig. 9 illustrates the method used to find an MPEG-2 compliant motion vector from an H.264/AVC unrestricted one, by inverting its temporal direction.

4.6. Refinement

As already pointed out, the motion vector accuracy used in H.264/AVC is different from that used in MPEG-2.

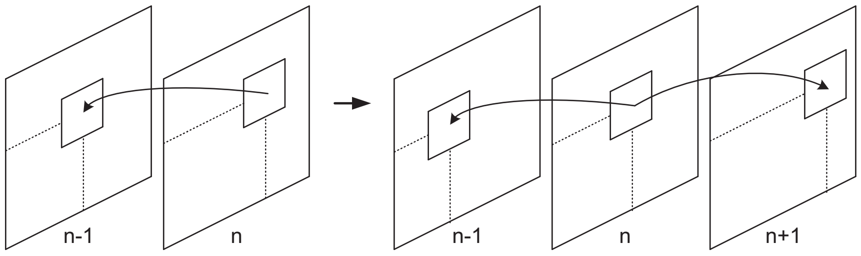


Fig. 8. Conversion of uni to bidirectional motion vectors.

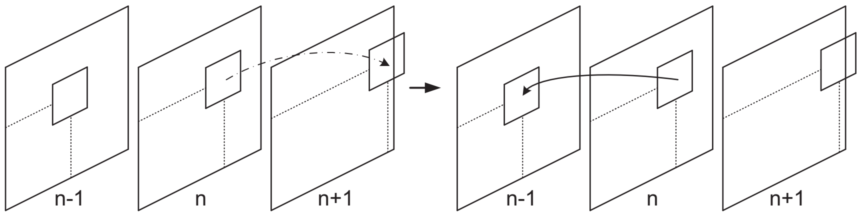


Fig. 9. Conversion of unrestricted motion vectors.

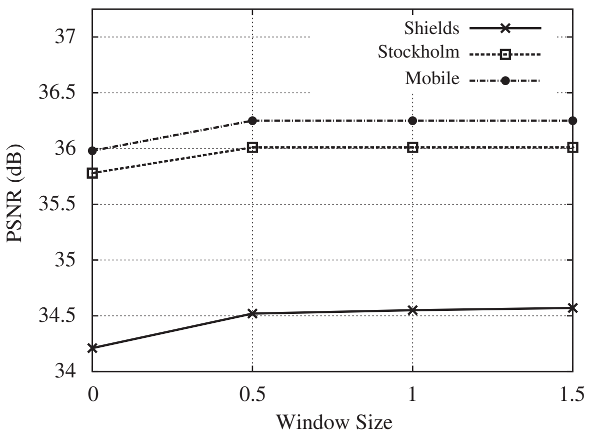


Fig. 10. PSNR gain with different refinement window sizes.

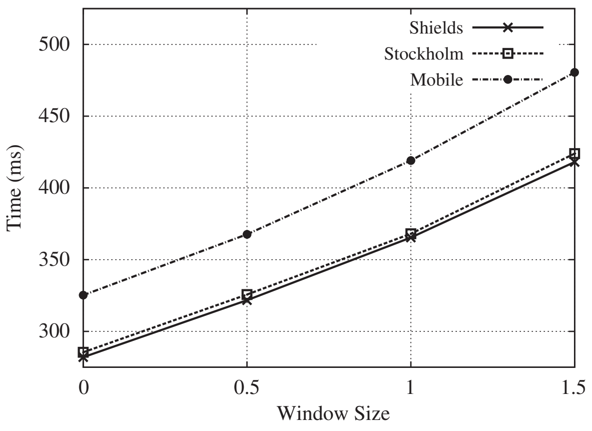


Fig. 11. Processing time for different refinement window sizes.

Therefore, accuracy conversion is also necessary to make MPEG-2 compliant motion vectors. Since a simple rounding operation to convert the pixel accuracy introduces suboptimal prediction, a refinement procedure through a search around the candidate motion vector is performed in order to fine tune the motion vector candidate. This refinement improves the prediction quality, as it minimises the inaccuracy of long distance motion vector scaling.

In order to obtain a good tradeoff between quality gain and computational complexity increase, several search window sizes were tested for three different test sequences, by measuring the objective quality increase, as well as computational increase. The results of Figs. 10 and 11 show that using a search window larger than half pixel does not yield significant quality gains, whereas the computational complexity increases linearly with the window size.

5. Intraconversion in the transform domain

In order to use transform domain intraconversion in H.264/AVC to MPEG-2 transcoding, direct computation of DCT coefficients from IT is necessary. This is quite different from the type of transform domain conversion needed in MPEG-2 to H.264/AVC transcoding, which was addressed in [20], mainly because of intraprediction. While intraprediction can be avoided in DCT to IT conversion, i.e., MPEG-2 to H.264/AVC transcoding, it is unavoidable in IT to DCT conversion because an H.264/AVC stream to be transcoded is not supposed to be constrained in its intraprediction modes. The H.264/AVC standard defines a total of four intraprediction modes for 16×16 blocks and nine prediction modes for 4×4 blocks. Since in IT to DCT conversion, the final block size is always 8×8, if one considers the total number of permutations of the nine possible prediction modes by groups of four blocks of size 4×4 (i.e., 8×8 blocks), then it results in $9^4 = 6561$

different possible cases. This means 6561 different types of transform domain conversions to be independently solved, which leads to an impracticable number of cases and makes pixel domain conversion more affordable for most of them.

In the case of the 16×16 intraprediction modes referred to as DC, vertical, horizontal and plane, the prediction values are computed from pixels in adjacent locations of the MB to be encoded as shown in Fig. A1 (Appendix A). Since the plane prediction mode use different weights for each individual pixel, it is computationally more efficient to convert this prediction mode in the pixel domain rather than in the transform domain. For the other three modes a transform domain conversion method was developed under the scope of this work, aiming at reduction of computational complexity. The proposed IT-to-DCT conversion method is based on the so-called **S** matrix, where such matrix is multiplied by an 8×8 block ($\boldsymbol{X}$) comprised four blocks of size 4×4 ($\boldsymbol{X}1, \boldsymbol{X}2, \boldsymbol{X}3, \boldsymbol{X}4$) of IT coefficients, in order to produce the corresponding 8×8 block ($\boldsymbol{Y}$) of DCT coefficients [21]. This is given by the matrix operations $\boldsymbol{Y} = \boldsymbol{S} \times \boldsymbol{X} \times \boldsymbol{S}^T$ and the full method is described in Appendix A. For the reasons mentioned above, intraconversion in the transform domain was not implemented in the transcoding architecture, and the topic is compiled in Appendix A. Note that in general, intraslices have a much lower temporal rate than P and B ones and almost all of the video data to be transcoded in a coded stream belongs to P and B slices. Therefore, intraslices cannot contribute significantly to reduce the overall computational complexity because they only constitute a small fraction of any video signal.

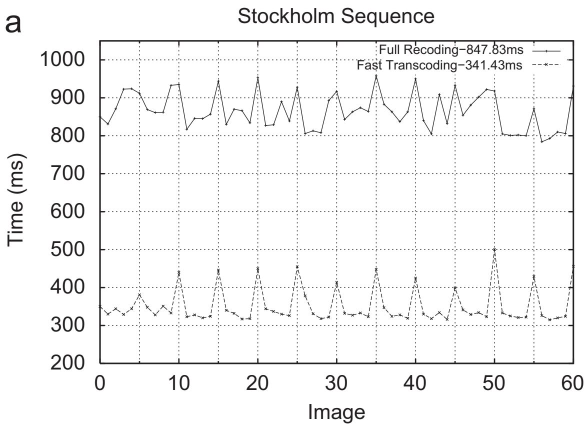
6. Experimental results

In this work, the fast transcoder implementation was based on H.264/AVC JM13.2 and MPEG-2 v1.2 (MSSG) *MPEG Software Simulation Group*. In order to evaluate the performance with the proposed interconversion methods, three sequences were used (Mobile, Stockholm and Shields) at 720×576 at 25 Hz, each with 250 frames. These source sequences were encoded with H.264/AVC using "Main Profile" at 5 Mbit/s (CBR), with RD optimisation enabled, a GOP (*Group Of Pictures*) size 12 following an "IBPBP" structure and allowing five reference pictures. The signal structure of the transcoded MPEG-2 stream was chosen to be the same as the incoming H.264/AVC one. Thus, the same parameters were used and a direct correspondence between both of them was done, including GOP structure, bit rate, etc. The experiments were carried out to compare direct MPEG-2 encoding "*MPEG-2_MSSG*", cascade decoding–encoding "*Full Recoding*" and the proposed transcoding scheme "*Fast Transcoding*". The PSNR and processing time were used as performance evaluation parameters for comparison between the different transcoding schemes.

Table 3 shows the average luminance PSNR obtained from direct coding (i.e., no transcoding) and transcoding the three sequences using both the reference cascade decoder–encoder transcoding scheme (Full_rec) and the proposed fast transcoder (Fast_trc). The PSNR is obtained by comparing the original sequence (i.e., before encoding with H.264/AVC) with the transcoded one, (i.e., after decoding with MPEG-2 decoder). These results show that full recoding and fast transcoding achieve the same practical objective quality and it is noteworthy that the proposed fast transcoder provides slightly higher PSNR than full-recoding. This results from the reuse of some better coding modes in the fast transcoder which benefits from the optimal mode decisions made by the initial H.264/AVC encoder. Another reason for such a difference is related with motion vector selection criterion, since the reference transcoder uses the sum of absolute differences (SAD) and fast transcoder uses the sum of squared differences. Although SAD computation has smaller

Table 3
Average luminance PSNR obtained from the proposed transcoder.

	Mobile PSNR (dB)	Stockholm PSNR (dB)	Shields PSNR (dB)
H.264/AVC	41.159	38.986	38.849
MPEG-2	36.571	36.975	34.605
Full_Rec	35.990	35.925	34.215
Fast_Trc	36.280	36.035	34.545



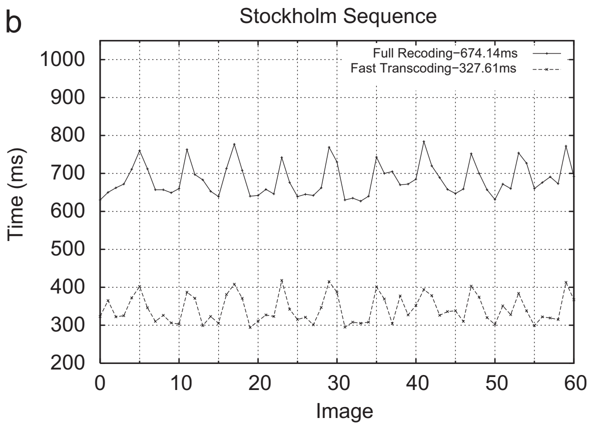


Fig. 12. Encoding time results. (a) Encoding time for P slices and (b) encoding time for B slices.

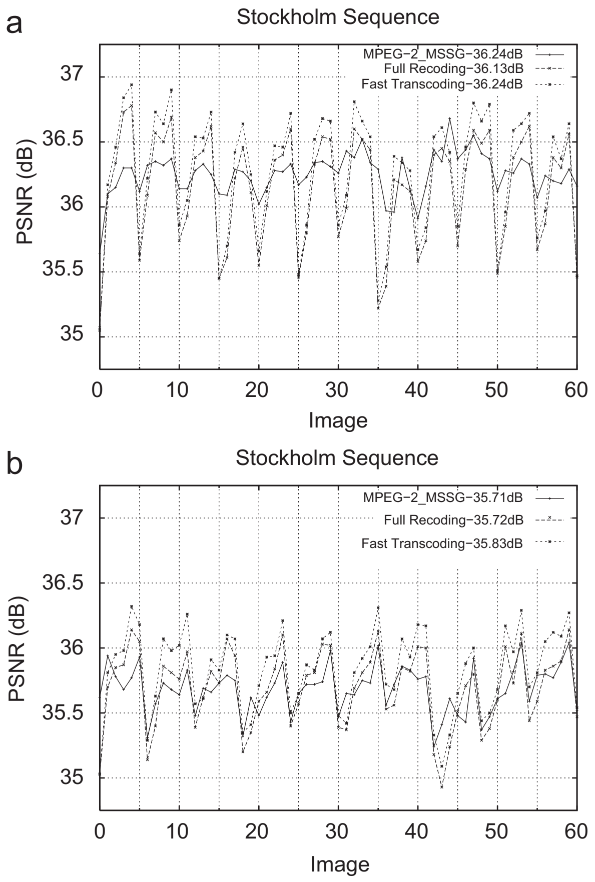


Fig. 13. PSNR results. (a) PSNR for P slices and (b) PSNR for B slices.

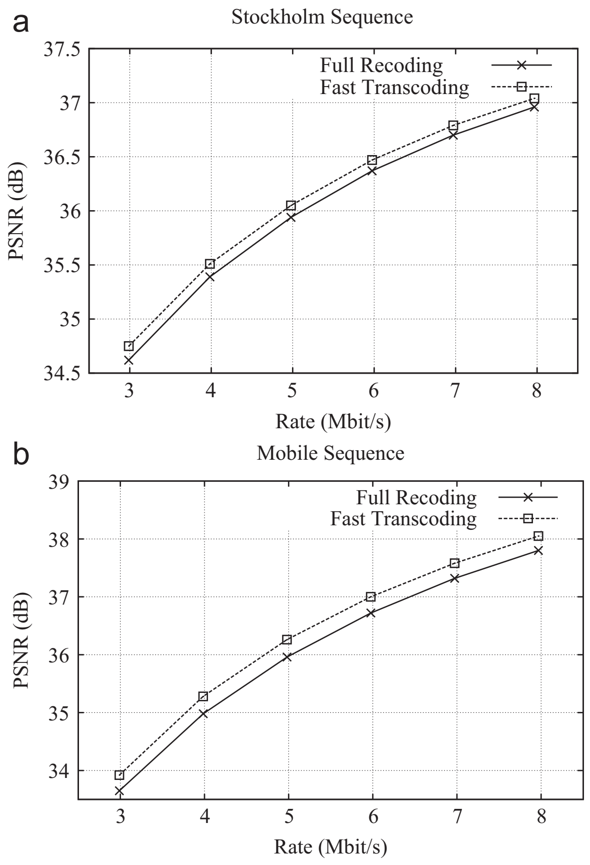


Fig. 14. Transcoding quality. (a) Rate-PSNR performance—Stockholm and (b) Rate-PSNR performance—Mobile.

Table 4
Complexity reduction compared with the reference transcoder.

Bitrate Mbit/s	Stockholm (%)	Mobile (%)
3	63.38	58.19
4	61.69	58.86
5	61.78	58.67
6	61.95	58.98
7	62.14	58.19
8	61.72	58.67

computational complexity than SSD, the latter is typically a better residue selection criterion [22].

As mentioned above, the processing time of both transcoding schemes was measured and compared for evaluating the relative computational complexity performance of the proposed fast transcoder. Figs. 12(a) and (b) show that savings in processing time up to 60% are achieved at lossless transcoding quality, when compared with full recoding. The quality difference between both transcoders is again very small, as shown in Figs. 13(a) and (b). The periodicity observed in these figures is related with the GOP size.

The overall quality of the transcoder was evaluated with further experiments carried out at different bitrates. The average PSNR results are shown in Figs. 14(a) and (b), while the percentage of computational complexity reduction in comparison with the reference transcoder is shown in Table 4 for the Stockholm and Mobile video sequences, respectively. These results show a consistent transcoding performance, with respect to both PSNR and complexity, over a wide range of bitrates.

It was also found that by using motion vector refinement within a 1×1 search window, at the half-pixel positions surrounding the integer motion vector candidate, improves the picture quality up to 0.3 dB at the expense of a maximum penalty of 5% in processing time. Therefore such a refinement is used in the transcoder for all motion vectors converted from the H.264/AVC input video stream. If larger search windows are used, then the computational complexity increases significantly, without resulting in relevant quality gains.

In order to provide further evidence about the efficiency of the proposed transcoder, different sequences with 125 frames of CIF and QCIF resolutions and lower bitrates were also transcoded. The coding conditions were the same as described for the previous experiments, except the bitrates of the source sequences which were set to 2 Mbps and 512 kbps for CIF and QCIF, respectively. Table 5 shows the average PSNR for the three sequences, i.e., encoded with H.264/AVC, MPEG-2 and transcoded

Table 5
PSNR for CIF and QCIF.

	Bus PSNR (dB)	Mobile PSNR (dB)	Stefan PSNR (dB)
CIF			
H.264/AVC	39.06	36.10	48.46
MPEG-2	37.79	34.79	40.91
Full_Rec	36.09	32.89	40.23
Fast_Trc	36.77	33.20	40.57
ΔPSNR	+0.68	+0.31	+0.34
QCIF			
H.264/AVC	38.70	36.89	39.73
MPEG-2	37.64	34.59	38.47
Full_Rec	35.67	32.97	36.30
Fast_Trc	35.80	33.38	36.53
ΔPSNR	+0.13	+0.41	+0.23

Table 6
Sequence processing time in seconds.

	Bus	Mobile	Stefan
CIF			
Full_Rec	25.688	18.759	19.429
Fast_Trc	8.517	7.556	9.474
Reduction (%)	66.84	59.72	51.24
QCIF			
Full_Rec	4.840	4.165	4.181
Fast_Trc	2.049	1.931	1.767
Reduction (%)	57.67	53.64	57.74

using full recoding and the proposed fast transcoder. Table 6 shows the processing time for the same three sequences using full recoding and the proposed fast transcoder. As these tables show, the results obtained for CIF and QCIF sequences are consistent with the ones obtained for higher resolution and bitrates. The PSNR obtained at the same bit rate for full recoding is slightly lower than for fast transcoding and much more relevant is the reduction between 51.2% and 66.8% in the processing time of the proposed transcoder, i.e., much lower computational complexity.

In general, the transcoding process achieves different computational complexity savings depending on the MB coding modes, thus the overall performance may also vary according to the sequence characteristics. However, among the three sequences used in these simulations, both the computational complexity gain and the video quality remain similar, and these results are consistent with other test sequences which were used to validate the implementation of the fast transcoder. In regard to P frames, the results obtained with the proposed transcoder are similar to those obtained in [10] for both quality and complexity, which also validates the transcoding methods proposed in this paper.

7. Conclusion

In this paper a fast and efficient transcoder was proposed, including novel methods for converting H.264/AVC video into MPEG-2 format, namely conversion of MB types resulting from unconstrained motion vectors and conversion from P-type MBs into B-type ones. Transcoding of interframe coding modes were thoroughly analysed, as they constitute the majority of the coded data in a video stream. The proposed mode conversion methods achieve a computational complexity reduction of up to 60%, with slightly better PSNR when compared with the reference transcoder. It was shown that, for some intraprediction modes, transcoding of intraslices in the transform domain is computationally more efficient than its pixel domain counterpart. However, since intraslices only account for a small fraction of a coded video stream and there is a huge number of possible combinations of intraprediction modes, for most of them pixel domain conversion can be more efficient than transform domain. Overall, the proposed transcoding methods are envisaged for software based implementations in devices with either limited processing power or hardware platforms running several simultaneous processes, such as home gateways and media adaptation proxies.

Acknowledgements

The authors would like to thank the anonymous reviewers for their useful comments and suggestions which were helpful to improve the quality of the paper.

Appendix A. Conversion of 16 × 16 intrapredicted blocks

Fig. A1 shows the predictions modes used in 16 × 16 blocks, namely vertical, horizontal, DC and plane. Fast and efficient transform domain conversion of the DC, vertical and horizontal modes is described in the following.

As pointed out in Section 5, $\boldsymbol{S}$ is the conversion matrix and $\boldsymbol{S}^T$ is its transpose. The $\boldsymbol{S}$ matrix is defined as shown below:

$$\boldsymbol{Y} = \boldsymbol{S} \times \boldsymbol{X} \times \boldsymbol{S}^T$$

$$\boldsymbol{S} = \begin{pmatrix} a & 0 & 0 & 0 & a & 0 & 0 & 0 \\ b & c & d & e & -b & c & -d & e \\ 0 & f & 0 & g & 0 & -f & 0 & -g \\ h & i & j & k & -h & i & -j & k \\ 0 & 0 & a & 0 & 0 & 0 & a & 0 \\ l & m & n & o & -l & m & -n & o \\ 0 & -g & 0 & f & 0 & g & 0 & -f \\ p & q & r & s & -p & q & -r & s \end{pmatrix} \tag{A.1}$$

and the elements $a, b, \ldots, s$ are as follows [20]:

$$\begin{array}{llll} a = 1.4142, & b = 1.2815, & c = 0.4618, & d = -0.1065 \\ e = 0.0585, & f = 1.1152, & g = 0.0793, & h = -0.45 \\ i = 0.8399, & j = 0.7259, & k = -0.0461, & l = 0.3007 \\ m = -0.4319, & n = 1.0864, & o = 0.5190, & p = -0.2549 \\ q = 0.2412, & r = -0.5308, & s = 0.9875 & \end{array} \tag{A.2}$$

The fast conversion algorithm is based on the symmetry properties of the ***S*** matrix.

A.1. Vertical prediction mode

In the vertical mode, samples of each column are predicted from the pixel values in the last row, at the same column, of the MB above. In the pixel domain this is equivalent to compute a prediction block through a matrix operation. Since DCT is computed for 8×8 blocks, it is necessary to divide the MB into four blocks of 8×8 and then for each one of these, the corresponding intraprediction in the pixel domain may be obtained by applying the following matrix operation to the 8×8 block located in the same vertical direction:

$$\begin{bmatrix} V_1 & V_2 & \cdots & V_8 \\ V_1 & V_2 & \cdots & V_8 \\ \vdots & \vdots & \ddots & \vdots \\ V_1 & V_2 & \cdots & V_8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \cdots & \times \\ \times & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ V_1 & V_2 & \cdots & V_8 \end{bmatrix} \tag{A.3}$$

In order to perform the equivalent operation in the transform domain, it is necessary to DCT transform each member of (A.3), which results in

$$\mathbf{T}\left(\begin{bmatrix} V_1 & V_2 & \cdots & V_8 \\ V_1 & V_2 & \cdots & V_8 \\ \vdots & \vdots & \ddots & \vdots \\ V_1 & V_2 & \cdots & V_8 \end{bmatrix}\right) = \underbrace{\mathbf{T}\left(\begin{bmatrix} 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}\right)}_{\mathbf{x}} \times \underbrace{\mathbf{T}\left(\begin{bmatrix} \times & \times & \cdots & \times \\ \times & \times & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ V_1 & V_2 & \cdots & V_8 \end{bmatrix}\right)}_{\mathbf{V}} = \mathbf{x}_{(1\times 8)} \times \mathbf{V}_{(8\times 8)}, \tag{A.4}$$

where **x** is a vector resulting from the DCT of the auxiliary matrix, and **V** represents the upper 8×8 coefficient block. Predictions in the vertical mode may be computed in the frequency domain by using expression (A.4). Moreover, the operation defined in (A.4) can be performed more efficiently than a matrix product, due to the vector form of **x**. In order to obtain the full MB of intrapredicted coefficients, it is necessary to perform the same operation (A.4) to the up right block and, then copy the coefficients from the upper blocks to the positions below.

A.2. Horizontal prediction mode

In the horizontal intraprediction mode, samples of each row are predicted from the pixel values in the last column of the MB on the left. By applying a similar procedure to that described in (A.1), one can obtain this intraprediction mode in the transform domain, by using the following expression:

$$\mathrm{T}\left(\begin{bmatrix} H_1 & H_1 & \cdots & H_1 \\ H_2 & H_2 & \cdots & H_2 \\ \vdots & \vdots & \ddots & \vdots \\ H_8 & H_8 & \cdots & H_8 \end{bmatrix}\right) = \underbrace{\mathrm{T}\left(\begin{bmatrix} \times & \times & \cdots & H_1 \\ \times & \times & \cdots & H_2 \\ \vdots & \vdots & \ddots & \vdots \\ \times & \times & \cdots & H_8 \end{bmatrix}\right)}_{\mathbf{H}} \times \underbrace{\mathrm{T}\left(\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}\right)}_{\mathbf{x}} = \mathbf{H}_{(8\times 8)} \times \mathbf{x}^T_{(8\times 1)}. \tag{A.5}$$

Like in the vertical prediction mode, the matrix multiplication, defined in Eq. (A.5), results in a product of a

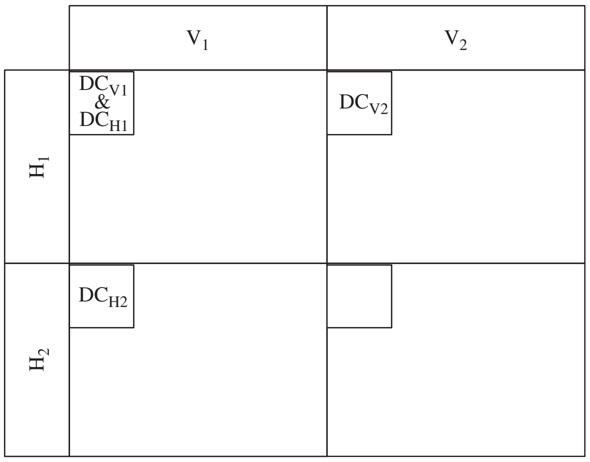


Fig. A2. DC coefficients notation, based on predicted blocks.

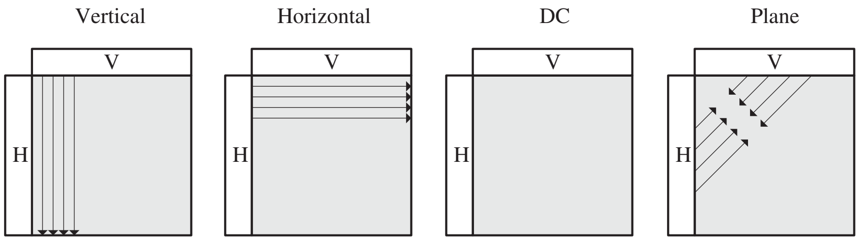


Fig. A1. Intraprediction modes 16×16.

matrix by a vector. In order to obtain the full MB of intrapredicted coefficients for this mode, it is necessary to perform the operation defined in (A.5) to the blocks located at the up right and down right positions of the MB, which is on the left of the MB to be encoded. Then the coefficients from the blocks on the left must be copied to the blocks on right positions.

A.3. DC prediction mode

In the DC prediction mode, the mean of the samples located in the last row of the upper MB and in the last column of the left MB need to be determined. It is possible to compute the mean of the neighbouring pixel values in the transform domain by using the vertical and horizontal intraprediction methods defined in Sections A.1 and A.2.

In the transform domain, this operation is carried out using the DC coefficients extracted from Eqs. (A.4) and (A.5). Using the notation depicted in Fig. A2, one can obtain the predicted block by using the following method: firstly, the sum of the DC coefficients shown in Fig. A2 is computed in order to obtain the mean value of the neighbouring pixels in the transform domain; secondly a rounding operation is applied as defined in [1], which results in the following expression:

$$\left[\mathrm{T}\left(\left(\left(\sum_{i=0}^{15}(H_i+V_i)+16\right)\geqslant 5\right)\times \mathbf{1}_{(8\times8)}\right)\right]_{DC} = ((DC_{H1}+DC_{H2}+DC_{V1}+DC_{V2})+16)\geqslant 2, \qquad (A.6)$$

where $\mathbf{1}_{(8\times8)}$ is a matrix of ones. The above equation formulates the DC prediction mode in the transform domain. The resultant value must be copied to each DC coefficient of the 8×8 blocks represented in Fig. A2. Note that in (A.6) the only nonzero matrix element is the DC coefficient.

Table A1
Number of operations required for intraprediction and transform conversion.

Method	Mul./shift	Add	Total
Vert. pred.	64	64	128
Horz. pred.	64	64	128
DC pred.	32	34	66
IT-to-DCT	352	352	704
FDCT	176	520	696
4Inverse IT	64	256	320

A.4. Computational complexity

The computational complexity of the proposed method for intratranscoding in the transform domain is analysed

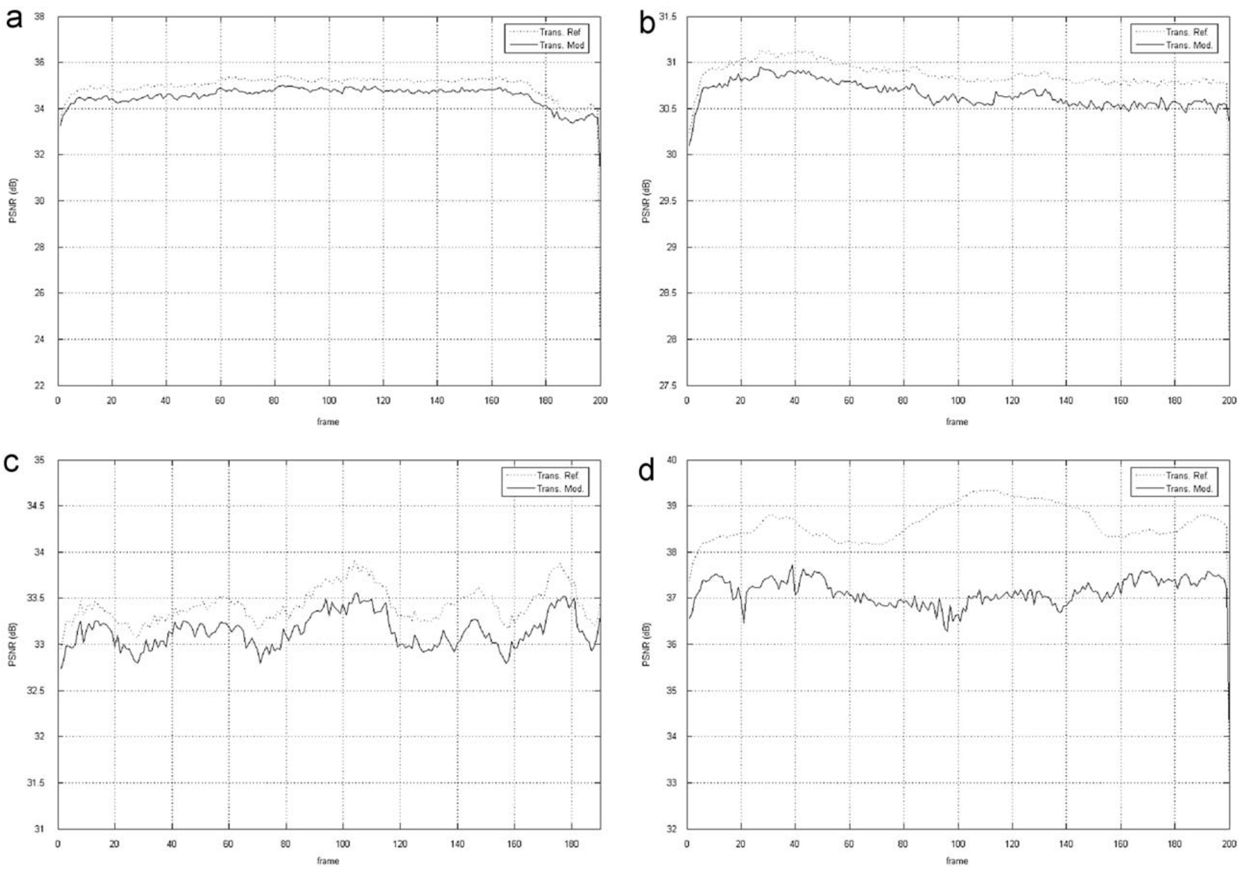


Fig. A3. PSNR for the sequences: (a) Carphone; (b) Foreman; (c) Mobile and (d) Akyio using $QP = 15$.

and compared with the pixel domain approach. In Table A1, it is shown that the pixel domain approach requires 696 operations to perform the forward DCT and 320 to perform four inverse integer transforms, resulting in a total of 1016 operations [23,24]. However, transcoding in the transform domain, only requires 704 operations for IT-to-DCT conversion and 128 operations for either the vertical or horizontal intraprediction (worst case), which results in 832 operations. Therefore the computational efficiency is improved about 19% in the worst case. In the case of the DC intraprediction mode, the gain in computational efficiency is increased at least to 25%.

A.5. Intratranscoding efficiency

The transcoding efficiency of the intraconversion method described above was evaluated by using an H.264/AVC stream encoded only with intraframes constrained to 16×16 intraprediction modes. In this experiment two transcoders were evaluated: the pixel domain transcoder (Trans. Ref.) and the transform domain transcoder (Trans. Mod.). In both transcoders a fixed quantisation step size was used ($QP = 15$). The peak signal-to-noise ratio (PSNR) of the transcoded sequences was computed using the same original sequence as reference for comparison. Fig. A3 shows the PSNR for both pixel and transform domain transcoding, using four sequences: Carphone, Foreman, Mobile and Akyio. As shown in Fig. A3, the PSNR degradation is small while a significant reduction in computation complexity is achieved by using the method described in Section 5. Therefore fast transcoding of intracoded MBs is worth to do for such prediction modes because the contribution for the overall reduction of computational complexity is significant at the expense of small degradation.

References

[1] JVT Joint Video Team of ISO/IEC MPEG and ITU-T VCEG JVT. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC), March 2003.

[2] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi, Video coding with H.264/AVC: tools, performance, and complexity, IEEE Circuits and Systems Magazine 1st Quarter (2004) 7–28.

[3] ITU-T, Recommendation H.262, Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video, February 2000.

[4] J. Xin, A. Vetro, S.-i. Sekiguchi, K. Sugimoto, MPEG-2 to H.264/AVC transcoding for efficient storage of broadcast video bitstreams, in: Proceedings of the International Conference on Consumer Electronics, 2006, pp. 417–418.

[5] L. Yang, X. Song, C. Hou, J. Dai, A scheme for MPEG-2 to H.264 transcoding, in: Proceedings of the Canadian Conference on Electrical and Computer Engineering, May 2006, pp. 310–313.

[6] T. Qian, J. Sun, D. Li, X. Yang, J. Wang, Transform domain transcoding from MPEG-2 to H.264 with interpolation drift-error compensation, IEEE Transactions on Circuits and Systems for Video Technology 16 (4) (April 2006) 523–534.

[7] H. Kalva, Issues in H.264/MPEG-2 video transcoding, in: Proceedings of the IEEE Consumer Communications and Networking Conference, January 2004, pp. 657–659.

[8] J. Chu, W. Lu, Y. Liu, Y. Song, X. Song, S. Yu, H.264 to MPEG-2 transcoding based on personal video recorder platform, in: Proceedings of the Ninth International Symposium on Consumer Electronics, June 2005, pp. 438–440.

[9] V. Patil, T. Kalyani, A. Bhartia, R. Kumar, J. Mukherjee, DCT domain transcoding of H.264/AVC video into MPEG-2 video, in: ICCVGIP06, 2006, pp. 696–707.

[10] P. Kunzelmann, H. Kalva, Reduced complexity H.264 to MPEG-2 transcoder, in: Proceedings of the IEEE International Conference on Consumer Electronics, Las Vegas, USA, January 2007.

[11] H. Kalva, P. Kunzelmann, Dynamic motion estimation for transcoding P frames in H.264 to MPEG-2 transcoders, IEEE Transactions on Consumer Electronics 54 (2) (May 2008) 657–662.

[12] S. Sharma, K.R. Rao, Transcoding of H.264 bitstream to MPEG-2 bitstream, in: Proceedings of the Asia-Pacific Conference on Communications, October 2007, pp. 391–396.

[13] S. Moiron, S. Faria, P. Assuncao, V. Silva, A. Navarro, Fast interframe transcoding from H.264 to MPEG-2, in: Proceedings of the IEEE International Conference on Image Processing, San Antonio, USA, September 2007.

[14] S. Moiron, S. Faria, P. Assuncao, V. Silva, A. Navarro, Inter mode conversion of H.264 to MPEG-2 video with reduced complexity, in: Proceedings of the IEEE International Conference on Consumer Electronics, Las Vegas, USA.

[15] ⟨http://iphome.hhi.de/suehring/tml/index.htm⟩, Suehring JM13.2 H.264.

[16] ⟨http://www.mpeg.org/MSSG⟩, MPEG2 v1.2.

[17] I. Ahmad, W. Xiaohui, Y. Sun, Y.-Q. Zhang, Video transcoding: an overview of various techniques and research issues, IEEE Transactions on Multimedia 7 (October 2005) 793–803.

[18] R. Marques, S. Faria, P. Assuncao, V. Silva, A. Navarro, Fast conversion of H.264/AVC integer transform coefficients into DCT coefficients, in: Proceedings of the International Conference on Signal Processing and Multimedia Applications, August 2006, pp. 5–8.

[19] T. Shanableh, M. Ghanbari, Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats, IEEE Transactions on Multimedia 2 (2) (June 2000) 101–110.

[20] J. Xin, A. Vetro, H. Sun, Converting DCT coefficients to H.264/AVC transform coefficients, Technical Report TR2004-058, Mitsubishi Electric Research Labs, 201 Broadway, Cambridge, MA, June 2004.

[21] J. Lee, K. Chung, Quantization/DCT conversion scheme for DCT-domain MPEG-2 to H.264/AVC transcoding, IECIE Transactions on Communications 88 (7) (2005) 2856–2863.

[22] T. Toivonen, J. Heikkila, Efficient method for half-pixel block motion estimation using block differentials, in: Lecture Notes in Computer Science—Visual Content Processing and Representation, vol. 2849, Springer, Berlin, September 2003, pp. 225–232.

[23] H.S. Malvar, A. Hallapuro, M. Karczewicz, L. Kerofsky, Low-complexity transform and quantization in H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology 13 (7) (July 2003) 598–603.

[24] M. Ghanbari, Video Coding: An Introduction to Standard Codecs, Institution of Electrical Engineers, Stevenage, UK, 1999.