EFFICIENT MOTION ESTIMATION IN H.264 REVERSE TRANSCODING

Chang-Hong Fu, Yui-Lam Chan and Wan-Chi Siu

Centre for Multimedia Signal Processing
Department of Electronic and Information Engineering
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ABSTRACT

In this paper, we propose a fast reverse motion estimation algorithm with efficient mode decision for reverse transcoding of H.264 bitstream. By analyzing the motion vectors and modes decoded from the forward bitstream, the best mode and motion vector for each backward transcoded macroblock are estimated. A remarkable reduction of computational complexity involved in reverse motion estimation can be achieved by the proposed algorithm with only negligible impact on the rate-distortion performance.

Index Terms: Video cassette recording (VCR), transcoding, H.264 video, reverse motion estimation.

1. INTRODUCTION

In order to provide fast and user-friendly browsing of video contents, some research works have been investigated to implement the full video cassette recording (VCR) functionality in MPEG video system [1-4] recently. Reverse playback is one of the VCR functions. However, the motion-compensated predictive technique that is adopted in various standards for compression severely complicates this operation. One conventional approach for performing reverse playback on compressed video is to decode the compressed video, store all uncompressed frames in the decoder, and display them in reverse order. This approach requires a significant amount of memory in playback devices. In [1-2], a reverse-play transcoder has been proposed to convert I–P frames into another I–P bitstream with reverse order in the server. When a playback device decodes this reverse-encoded bitstream, reverse playback can be achieved. For the dual bitstream system that was proposed in [3] for providing the full VCR functionality, transcoding the existing bitstream to a new one in reverse direction is also required. To expedite the reverse transcoding process, a method of estimating the reverse motion vectors for the new I–P bitstream based on the forward motion vectors of the original I–P bitstream has been suggested in [2]. Nowadays, the H.264 standard [5] achieves considerably higher coding efficiency than the previous standards. It is also highly desirable to provide reverse playback in H.264. However, the variable block size motion estimation adopted in H.264 introduces mode decision section in the encoding process and this further complicates the process of reverse transcoding.

In this paper, we exploit the relationship among various modes and motion vectors in the original forward bitstream to speed up the reverse motion estimation process. The paper is organized as follows. Section 2 describes the reverse transcoding process of H.264 bitstream. Section 3 illustrates the proposed method. Simulation results are presented in Section 4 and finally some conclusions are given in Section 5.

2. REVERSE TRANSCODING OF H.264 BITSTREAM

Figure 1 shows a scenario in which frame n is backward encoded with frame $n+1$. An important problem that arises in this reverse transcoding is to calculate the reverse motion vector (RMV), which is extremely computational expensive. In particular, the H.264 supports motion estimation using different block sizes. Each 16×16 pixel macroblock (MB) can be divided into MB partitions of sizes 16×8, 8×16, or 8×8. In each 8×8 partition, it can be further divided into sub-MB partitions of sizes 8×4, 4×8, or 4×4. These variable block sizes provide different modes that are actually arranged into two-level hierarchy. The first level (L1) contains modes of 16×16, 16×8 and 8×16 while the second level (L2) is defined as $P8\times8$ mode, of which each 8×8 block can be one of submodes such as 8×8, 8×4, 4×8, or 4×4. To perform reverse transcoding in the H.264, reverse motion estimation is firstly performed for all modes and submodes independently by minimizing the cost J_{motion}.

$$J_{motion}(RMV,\lambda_{motion}) = SAD(s,r) + \lambda_{motion}\cdot R_{motion}(RMV - PMV) \quad (1)$$

where PMV is the motion vector used for prediction, λ_{motion} is the Lagrangian multiplier for reverse motion estimation, $R_{motion}(RMV - PMV)$ is the estimated number of bits for coding RMV, and SAD is sum of absolute differences between the original block s and its reference block r.

After motion estimation for each mode, a rate-distortion (RD) optimization technique is used to get the best reverse mode and its general equation is given by:

$$J_{\text{mode}}(s,c,RMODE,\lambda_{\text{mode}}) = SSD(s,c,RMODE) + \lambda_{\text{mode}} \cdot R_{\text{mode}}(s,c,RMODE) \quad (2)$$

where λ_{mode} is the Lagrangian multiplier for mode decision, *RMODE* is one of the candidate modes during reverse motion estimation, *SSD* is sum of the squared differences between *s* and its reconstruction block *c*, and $R_{mode}(s,c,RMODE)$ represents the number of coding bits associated with the chosen mode. To compute J_{mode}, forward and inverse integer transform, and variable length coding are performed. The mode having the minimal cost is decided as the best reverse mode. Submode decision for *P8×8* is also carried out by using (2) with *RMODE* indicating a mode chosen from *8×8*, *8×4*, *4×8*, and *4×4*. The submode with the minimal cost is selected as the best reverse submode in this *P8×8* mode. By using brute-force search, it can guarantee to obtain the best RD performance, but the calculations of J_{motion} and J_{mode} incur a large amount of computational effort. In MPEG-2/4, RMVs can be obtained by negating the horizontal and vertical components of the corresponding forward motion vectors in the same spatial location of frame *n+1* [1-2]. This idea can be further extended to mode decision in H.264 in which the reverse modes of frame n can be directly copied from the modes of frame *n+1* in the corresponding spatial location. This is called "in-place" reverse motion estimation. However, a major disadvantage of this method is that it always obtains incorrect modes in the case of a scene with large motion activities or complicated spatial details.

3. PROPOSED TRANSCODING METHOD

In Figure 1, we assume that macroblock *k* of frame *n*, MB_n^k, is backward encoded using frame *n+1* as the reference. Let us represent the reverse mode and the set of RMVs of MB_n^k as $RMODE_n^k$ and RMV_n^k respectively. For the variable block size motion estimation, each RMV_n^k contains a number of motion vectors, i.e., $RMV_n^k = \{ RMV_n^{k,1}, RMV_n^{k,2} ..., RMV_n^{k,m} \}$, where *m* is the total number of partitions in MB_n^k. Besides, if $RMODE_n^k$ is *P8×8* mode, each of the four *8×8* blocks has its submode $RSUBMODE_n^k$. In the forward bitstream, the sets of forward motion vectors of MB_n^k and MB_{n+1}^k are denoted by $FMV_n^k = \{ FMV_n^{k,1}, FMV_n^{k,2} ..., FMV_n^{k,m} \}$ and $FMV_{n+1}^k = \{ FMV_{n+1}^{k,1}, FMV_{n+1}^{k,2} ..., FMV_{n+1}^{k,m} \}$ respectively. Their corresponding modes (submodes) are $FMODE_n^k$ ($FSUBMODE_n^k$) and $FMODE_{n+1}^k$ ($FSUBMODE_{n+1}^k$). To expedite mode decision of $RMODE_n^k$, our proposed algorithm exploits the relationship among FMV_n^k, FMV_{n+1}^k, $FMODE_n^k$ and $FMODE_{n+1}^k$.

In a natural video sequence, the homogeneous areas such as the background are always coded using *16×16* mode. On the contrary, a smaller partition is chosen in the non-homogeneous areas with strong edges. In general, $FMODE_n^k$ provides the spatial partition of MB_n^k. Therefore, $RMODE_n^k$ can directly use $FMODE_n^k$. However, $FMODE_n^k$ only reflects various motions of all objects inside MB_n^k by using frame *n-1* as the reference. But, in choosing the best $RMODE_n^k$, the reference frame is frame *n+1*. In the forward bitstream, $FMODE_{n+1}^k$ together with FMV_{n+1}^k of MB_{n+1}^k (the corresponding spatial location of MB_n^k in frame *n+1*) can indicate the correlation between modes used in frame *n* and frame *n+1*. Thus an improved way of determining $RMODE_n^k$ can be achieved by adaptively choosing $FMODE_n^k$ and $FMODE_{n+1}^k$.

Generally, the motion activity of MB_{n+1}^k, MA_{n+1}^k, provides the correlation between $RMODE_n^k$ and $FMODE_{n+1}^k$ and it is defined as:

$$MA_{n+1}^k = \frac{1}{m}\sum_{i=1}^{m}\left(\left|u_{n+1}^{k,i}\right| + \left|v_{n+1}^{k,i}\right|\right) \quad (3)$$

where $u_{n+1}^{k,i}$ and $v_{n+1}^{k,i}$ are the horizontal and vertical components of $FMV_{n+1}^{k,i}$. If $FMODE_{n+1}^k$ is the *16×16* mode and its MA_{n+1}^k has a small value, it is most likely that $RMODE_n^k$ is also set to the *16×16* mode due to its temporal stationary. Besides, RMV_n^k is estimated by $-FMV_{n+1}^k$. On the other hand, a large value of MA_{n+1}^k signifies that $RMODE_n^k$ and $FMODE_{n+1}^k$ have only small correlation. In this case, only $FMODE_n^k$ can provide the best estimate of $RMODE_n^k$. In the actual situation, FMV_n^k solely represents the motion between frame *n* and frame *n-1*. While RMV_n^k refers to the motion between frame *n* and frame *n+1*, $-FMV_n^k$ cannot be directly used, but be taken as a good predictor by considering some kind of temporal smoothness of the motion trajectory. For MA_{n+1}^k with medium value, either $FMODE_n^k$ or $FMODE_{n+1}^k$ is possible to be selected. If they are the same and belong to L1 (*16×16*, *16×8* or *8×16*), there is very high chance that this mode can be used in $RMODE_n^k$. Otherwise, if both $FMODE_n^k$ and $FMODE_{n+1}^k$ belong to L2, the motion activity and spatial structure of MB_n^k is complex. It is necessary to select $RSUBMODE_n^k$ of each *8×8* partition from $FSUBMODE_n^k$ and $FSUBMODE_{n+1}^k$ by calculating their J_{mode}. The algorithm is illustrated as follows:

Step 1: Compute the motion activity of MB_{n+1}^k, MA_{n+1}^k, using (3).

Step 2: Compare MA_{n+1}^k with the threshold T_{high}. If $MA_{n+1}^k \geq T_{high}$, choose $FMODE_n^k$ as the final mode of $RMODE_n^k$. Set the initial $RMV_n^{k,i}$ to $-FMV_n^{k,i}$, for *i=1*, *2*,…, *m*, followed by a refined motion estimation with a significantly reduced search area to obtain the final $RMV_n^{k,i}$.

Step 3: Compare MA_{n+1}^k with the threshold T_{low} ($T_{low} < T_{high}$). If $MA_{n+1}^k \leq T_{low}$ and $FMODE_{n+1}^k = 16\times16$ mode, set $RMODE_n^k$ and $RMV_n^{k,1}$ to *16×16* mode and $-FMV_{n+1}^{k,1}$ respectively.

Step 4: Otherwise, choose the best of $RMODE_n^k$ from $FMODE_{n+1}^k$ and $FMODE_n^k$ based on the following three conditions.

C1: If $FMODE_n^k = FMODE_{n+1}^k$ and they both belong to L1, set $RMODE_n^k$ to $FMODE_n^k$ (= $FMODE_{n+1}^k$). Perform motion vector refinement by using both $-FMV_n^{k,i}$ and $-FMV_{n+1}^{k,i}$ as the initial vectors and the one with smallest J_{motion} is selected as $RMV_n^{k,i}$.
C2: If $FMODE_n^k = FMODE_{n+1}^k$ and they both belong to L2, set $RMODE_n^k$ to $FMODE_n^k$ (= $FMODE_{n+1}^k$). Determine $RSUBMODE_n^k$ and RMV_n^k according to the following steps:

- Perform motion vector refinement by using both $-FMV_n^{k,i}$ and $-FMV_{n+1}^{k,i}$ as the initial vectors for each *8×8* sub-MB partition.
- Choose the best submode of $RSUBMODE_n^k$ from $FSUBMODE_n^k$ and $FSUBMODE_{n+1}^k$ by calculating J_{mode} in (2) using their refined motion vectors, and set RMV_n^k to the corresponding refined motion vectors.

C3: If $FMODE_n^k \neq FMODE_{n+1}^k$, determine $RMODE_n^k$ with possible $RSUBMODE_n^k$ and RMV_n^k as follows:

- Refine motion vectors by using both $-FMV_n^{k,i}$ and $-FMV_{n+1}^{k,i}$ as the initial vectors for each MB partition or sub-MB partition.
- For both $FMODE_{n+1}^k$ and $FMODE_n^k$, calculate J_{mode} by using their refined motion vectors. If one of them is *P8×8* mode, reckon its corresponding submode in computing J_{mode}.
- Select the best mode with smaller J_{mode} as $RMODE_n^k$ and set the corresponding refined motion vectors as RMV_n^k. If $RMODE_n^k$ is *P8×8* mode, set also $RSUBMODE_n^k$ to the corresponding submodes ($FSUBMODE_{n+1}^k$ or $FSUBMODE_n^k$).

The motion estimation and mode decision process can be summarized by the flowchart in Figure 2. After that, based on the resulted motion vectors of RMV_n^k, a merging process could be taken to finish the reverse mode decision. The principle is simple but efficient. If there exists a common motion vector among any neighboring MB partitions (or sub-MB partitions), they can possibly be merged to a larger MB partition (or sub-MB partition) with that common motion vector.

4. SIMULATION RESULTS

The proposed reverse motion estimation algorithm is implemented based on the JVT JM 9.2 encoder [5] and was tested from five video sequences (Foreman QCIF, Carphone QCIF, Salesman CIF, Tabletennis CIF, and Flower CIF) with 5 quantization parameters, i.e., QP=*20*, *24*, *28*, *32*, and *36*. In our proposed algorithm, T_{high} and T_{low} were set to *256* and *32* respectively, and a search area of *±1* pixel was used for motion vector refinement. The results are tabulated in Table 1. In the table, $\Delta PSNR$, $\Delta bitrate$, ΔJ_{motion} and ΔJ_{mode} represent a PSNR change, a bitrate change in percentage, percentage changes in the numbers of J_{motion} and J_{mode} calculations when compared to the "brute-force" algorithm with a search range of *±8* pixels, respectively. The positive values mean increments whereas negative values mean decrements. From the results in Table 1, it is observed that the proposed algorithm can substantially reduce ΔJ_{motion} and ΔJ_{mode} over *99%* and *83%* respectively. Note that the "in-place" algorithm [1-2] does not need to compute ΔJ_{motion} and ΔJ_{mode} since it directly reuses the modes and motion vectors in the forward bitstream. However, the RD performance of the "in-place" algorithm is greatly deteriorated as depicted in Figure 3 where the RD curve of the "Foreman" sequence is shown. On contrast, the RD performance of the proposed algorithm can achieve similar result to the "brute-force" algorithm. Table 1 also shows that the proposed algorithm has consistent gain in coding efficiency for all video sequences as compared with the "in-place" algorithm. On average, the proposed algorithm can achieve tremendous speed up at a cost of *0.14* dB PSNR drop and *6.44%* bitrate increase compared to the "brute-force" algorithm.

5. CONCLUSION

A fast reverse motion estimation and mode decision algorithm for H.264 is proposed. The motivation is to exploit the relationship among various modes and motion vectors in the original forward bitstream. Experimental results show that the proposed algorithm can achieve remarkable speed up while maintaining a similar RD performance. This reduction in computational complexity helps the real-time VCR implementation of H.264 in reverse transcoding.

6. ACKNOWLEDGEMENTS

The work is partially supported by the Centre for Multimedia Signal Processing, EIE, PolyU and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU 5123/05E). C. H. Fu acknowledges the research studentships provided by the University.

7. REFERENCES

[1] S. J. Wee and B. Vasudev, "Compressed-domain reverse play of MPEG video streams," in Proc. SPIE Conf. Multimedia Systems and Applications, pp. 237-248, November 1998.
[2] S. J. Wee, "Reversing motion vector fields," in Proc. IEEE International on Conference Image Processing 1998 (ICIP98), pp. 209-212, October 1998.
[3] C. W. Lin, J. Zhou, J. Youn, and M. T. Sun, "MPEG video streaming with VCR functionality," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, No. 3, pp. 415-425, March 2001.

[4] Chang-Hong Fu, Yui-Lam Chan, and Wan-Chi Siu, "Efficient Reverse-Play Algorithms for MPEG Video With VCR Support," IEEE Trans. On Circuits and Systems for Video Technology, Vol. 16, No.1, pp.19-30, January 2006.
[5] JVT Reference Software JM9.2 downloaded from http://iphome.hhi.de/suehring/tml/download/

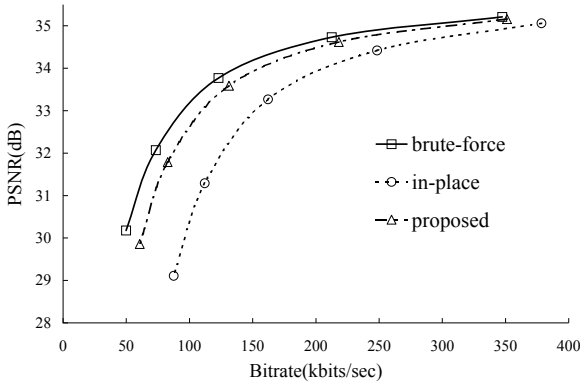

Figure 3. RD curves for the "Foreman" sequence.

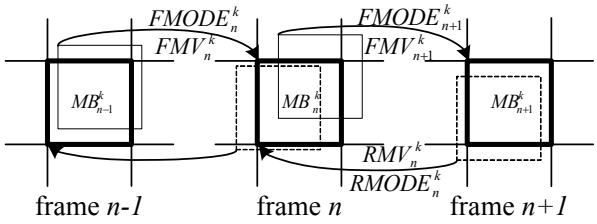

Figure 1. Reverse transcoding of MB_n^k .

Table 1. Performance comparisons.

	In-place		Proposed			
	ΔPSNR	Δbitrate	ΔPSNR	Δbitrate	ΔJ_{motion}	ΔJ_{mode}
Foreman	–0.50	+31.9%	–0.18	+6.7%	–99.4%	–88.0%
Carphone	–0.67	+40.8%	–0.24	+9.2%	–99.5%	–90.1%
Salesman	–0.23	+19.9%	–0.07	+5.1%	–99.8%	–96.3%
Tabletennis	–0.37	+27.7%	–0.17	+8.5%	–99.5%	–90.7%
Flower	–0.20	+12.5%	–0.06	+2.7%	–99.3%	–83.1%

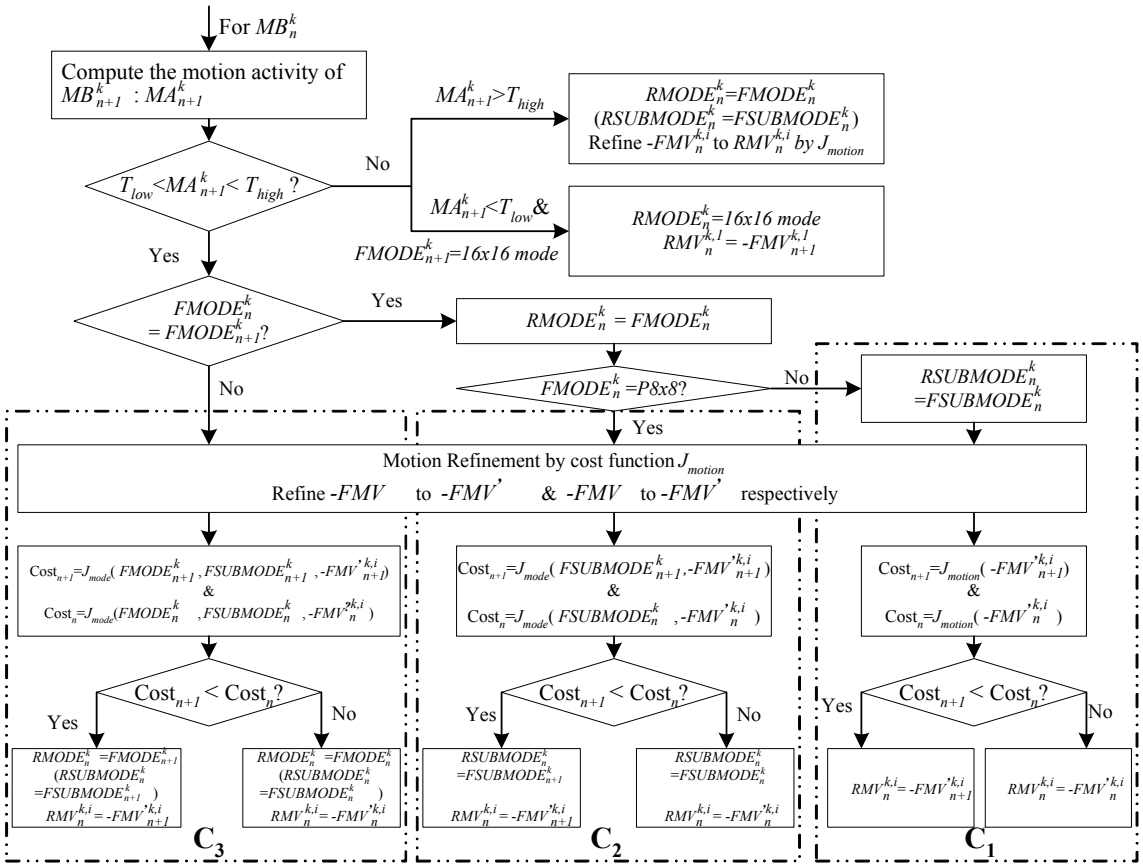

Figure 2. Flowchart of the proposed algorithm.