LOSSLESS FMO AND SLICE STRUCTURE MODIFICATION FOR COMPRESSED H.264 VIDEO

Wai-tian Tan, Eric Setton, and John Apostolopoulos

Streaming Media Systems Group
Hewlett-Packard Laboratories, Palo Alto, CA

ABSTRACT

We introduce a scheme to losslessly modify pre-compressed H.264 video to enable at streaming time (1) modification of slice sizes to fit transport packet size, and (2) introduction of an error resilience feature, namely Flexible Macroblock Ordering. By lossless we mean the reconstructed video from the modified bitstream is identical to that from the original compressed bitstream. We outline how this transcoder operates, and discuss some of its restrictions. The bit rate overhead for operating on the pre-compressed, rather than directly encoding the original video with the desired characteristics, is analyzed. Simulation results show 1 to 2 dB improvement when our scheme is applied to QuickTime generated H.264 videos transported over a lossy packet network.

Index Terms— Error resilience, H.264, MPEG-4 AVC.

1. INTRODUCTION

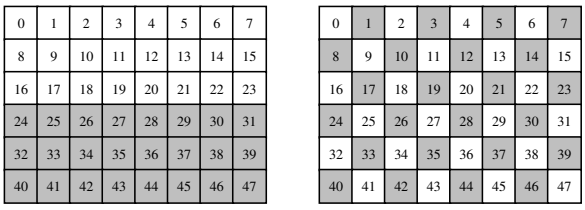
Slices are independently-decodable collections of macroblocks that compose a coded picture. It is well known that the error resilience of a compressed video depends significantly on its slice structure. The reason for the dependency is twofold. First, since they are coded independently, the use of more slices improves error resilience at the expense of lower coding efficiency. Second, to improve the effect of error concealment, H.264 [1] has a *Flexible Macroblock Ordering (FMO)* feature that allows a slice to contain an arbitrary set of macroblocks that need not be in raster-scan order. FMO also involves a tradeoff between error resilience and compression efficiency [2]. An illustration of FMO is given in Fig. 1.

Generally, the slice structure of a compressed video is determined at content-creation time and is not modified when streaming. For video transmission over a channel with few or no losses, it is efficient to have one slice per picture. When losses are more frequent, however, it is preferable to employ one slice per transport packet and FMO [3]. It remains difficult to make the appropriate choice at content creation (encoding) time, since channel characteristics at streaming time may be difficult, if not impossible, to predict, and the content may also be used in multiple settings with different loss characteristics. Instead, a better solution is to allow the slice structure to be modified while streaming according to observed channel conditions.

There are different techniques for transcoding compressed video during a streaming session, and these may or may not incur degradation in visual quality depending on the specific technique used [4]. Generally, transcoding techniques involving requantization tends to incur significant "re-encoding loss" of signal quality, and can incur high computational cost.

In this paper, we introduce a low-complexity scheme in which compressed H.264 video can be *losslessly* transformed into another H.264 video with a different slice structure. By lossless, we mean the input and output videos decode to reconstructed pictures with identical pixel values. Such a scheme is desirable in two scenarios. First, many content creation tools such as QuickTime Pro do not allow user to choose the slice structure or employ FMO. Our scheme allows error resilience to be introduced post-encoding. The possible PSNR improvement achieved by our scheme when transmitting such content over lossy networks is presented in Section 4. Second, even for encoders that allow arbitrary slice structures, our scheme eliminates the need for predicting channel conditions at content creation time.

The remainder of this paper is organized as follows. The procedures of the proposed scheme are discussed in Section 2. The overhead of adding FMO to a compressed video compared to encoding it directly is characterized in Section 3. The proposed scheme is then applied to H.264 video content generated by QuickTime, and simulation results under various packet loss rates are presented in Section 4.



(a) Example without FMO (b) "Checkerboard" with FMO

Fig. 1. Instead of raster-scan only, FMO allows slices (shown in different shades) to contain an arbitrary set of macroblocks.

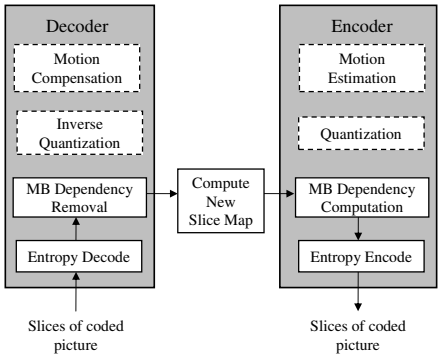


Fig. 2. Modification of FMO usage is achieved using low-complexity operations in the compressed domain only.

2. CHANGING SLICE STRUCTURE LOSSLESSLY

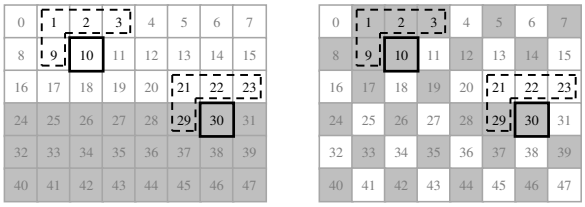
The main components of our scheme are outlined in Fig. 2. The procedure is applied to P-slices only, and has been tested on many video sequences coded using (1) JM 10.1 [5], baseline profile, and (2) QuickTime Pro 7.1.3. I or IDR-slices are not modified. Considering B-slices may be the subject of future work. The procedures operate in the compressed domain and have low computational complexity.

First, in the `Entropy Decode` step of Fig. 2, our scheme reverses the entropy coding of P-slices of a compressed picture to obtain all the symbols. These include the macroblock type and the quantization parameter (QP) difference, for each macroblock, motion vector differences, and the number of non-zero coefficients, for each 4×4 block. In the step `MB Dependency Removal`, differential coding is reversed to obtain motion vectors and QPs for each macroblock.

2.1. Forming a New Slice Map

When a coded picture of the input video contains intra-coded macroblocks, the possible output slice maps consistent with perfect reconstruction is constrained. For example, consider Fig. 3 where macroblocks (MB) 10 and 30 are intra-coded. Due to possible intra-prediction, MB 1, 2, 3, 9 may be needed to decode MB 10, and therefore need to be assigned to the same output slice as 10, as shown in Fig. 3-(b). Similarly, MB 30 is dependent on MB 29 but not on 21, 22, 23 on the input. In the output slice map, we need to assign 29 and 30 to the same slice, and 21, 22, 23 to a different slice than 30. This is necessary as prediction in intra-macroblocks, when it is implicit, depends on the presence or absence of different neighbors of the macroblock in the decoded slice.

This constraint imposed by intra-blocks has two consequences. First, arbitrary mapping of macroblocks to slices is not possible. We can only freely assign non-intra macroblocks and macroblocks outside of "intra-neighborhoods" (shown dotted in Fig. 3). Therefore, in Sections 3 and 4, when we mention "checkerboard" and "alternate lines" assignments, we mean an approximate assignment after constraints of intra-blocks are satisfied. Second, for irregular slice maps, it is necessary to explicitly code the slice map. For the case of two slices per picture considered later in this paper, the cost is one bit per macroblock.



(a) Input picture slice map (b) Modified slice map

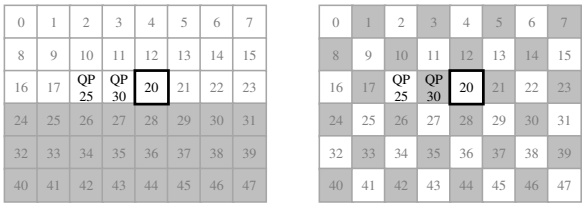
Fig. 3. The intra-coded macroblocks 10 and 30 have a neighborhood (shown dotted) which constrains the modified slice map and only allows an approximate "checkerboard".

2.2. MB Dependency Computation

After an output slice map is determined, quantities that are predictively coded need to be adjusted to the new slice structure. First, motion vector differences. In `MB Dependency Removal`, the motion vectors for every block (4×4, 8×8 or 16×16) are computed. New motion vector predictors under the new slice map are determined, and new motion vector differences are computed and coded. Please note that a skipped macroblock has a motion vector difference of zero. If the predictor changes under the new slice map, it is necessary to code formerly skipped blocks explicitly with the new motion vector difference (and a "coded block pattern" (CBP) of zero).

Second is the number of nonzero coefficients for 4×4 blocks. Similar to motion vector differences, these values are simply re-encoded using the new prediction associated with the output slice map.

The QP difference is re-encoded for macroblocks with such a field. For macroblocks with no nonzero coefficients (skipped or with a CBP of zero), the QP difference is not coded. In Fig. 4-(a), MB 18 and 19 have QP of 25 and 30, respectively; MB 20 is skipped and decoded with the same QP as the previous block, i.e., 30. For the slice map of Fig. 4-(b), a new QP of 25 is assumed if this block was coded as



(a) Input picture slice map (b) Modified slice map

Fig. 4. When macroblock 20 is coded as *skip*, a QP of 30 is assumed in (a) and a QP of 25 is assumed in (b). The QP for MB 20 must be 30 to ensure perfect reconstruction.

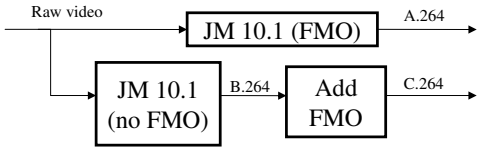


Fig. 5. Setup to analyze loss in coding efficiency by adding FMO to the compressed bitstream.

skip. Although MB 20 has no nonzero coefficients, a QP of 30 needs to be maintained since deblocking filter (thus, pixel values) depends on QP. The solution is to encode the presence of a coded block pattern (CBP), thereby enabling the encoding of the QP difference. As the presence of a CBP signals possible nonzero coefficients in four 4×4 blocks of the macroblock, we explicitly code that each 4×4 block contains no nonzero coefficients.

3. TRANSCODING OVERHEAD

In the previous section, we discuss the difficulty of imposing an arbitrary slice map to a pre-compressed video, which leads to some overhead in explicitly coding the macroblock map. We also discuss how extra bits are necessary to preserve QP for macroblocks that are skipped or have no CBP. This overhead is not necessary when a video is directly encoded with FMO. In this section, we analyze the overhead resulting from adding FMO to the compressed bitstream by analyzing the performance of the setup described in Fig. 5. Two FMO patterns are examined: "checkerboard" and "alternate lines" (where successive lines of macroblocks are assigned alternately to 2 different slices). Please note that videos *B.264* and *C.264* result in identical reconstructed pictures that are

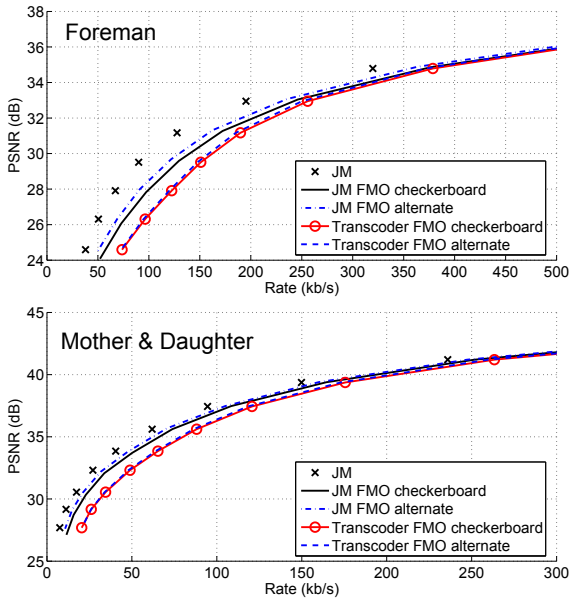


Fig. 6. Loss in coding efficiency by adding FMO in compressed domain.

different from those of *A.264*, and that *C.264* may only produce the desired FMO pattern approximately.

The results are shown in Fig. 6 for the 300-frame, 30 f/s, CIF sequences *Foreman* and *Mother and Daughter* coded using constant QP. Except for the first frame, all the frames are encoded as P-frames. We see that the performance penalty is negligible (about 3%) at medium to high bit rates of over 500 kb/s for *Foreman* and over 250 kb/s for *Mother and Daughter*. This is expected, as similar overhead is amortized over a higher bit rate. On the other hand, there is a loss of between 1.5 to 1.8 dB at lower bit-rates of about 100 kb/s for *Foreman* and 40 kb/s for *Mother and Daughter* due to the large percentage represented by overhead at those rates.

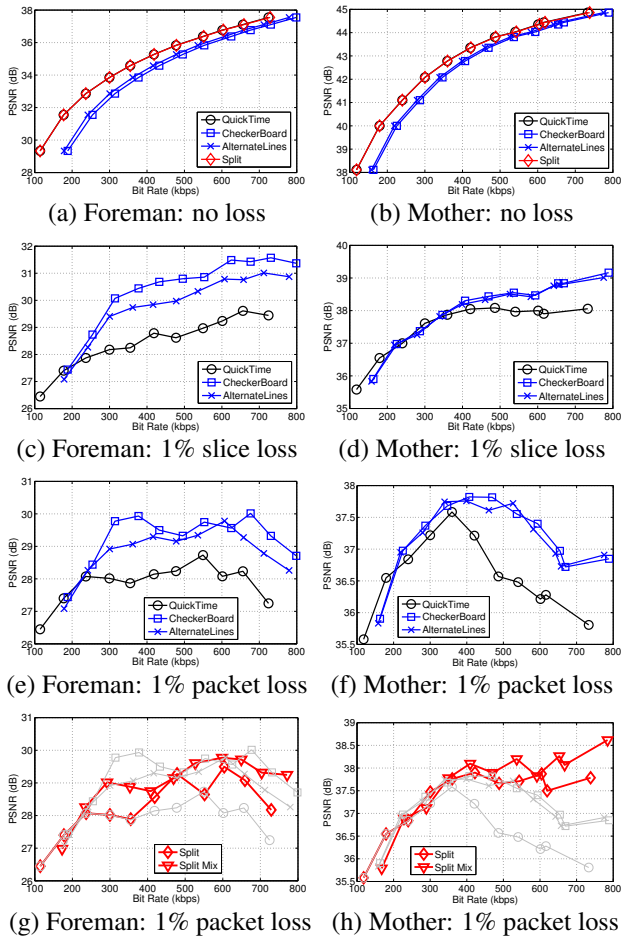
4. TRANSPORT IN CHANNEL WITH LOSSES

In this section, we consider the transport of compressed video over a lossy channel. We use QuickTime Pro 7.1.3 to encode the 300-frame CIF sequences *Foreman* and *Mother and Daughter* at different bit-rates. QuickTime does not allow the user to specify slice structure and by default splits every picture into two slices as depicted in Fig. 1-(a). Only the first frame is coded as intra. By using our scheme, we can add different flavors of FMO to improve the error resilience of the compressed video. Specifically, we employ the "checkerboard" and "alternate lines" schemes, and compare the resulting PSNR under different loss patterns.

The results are summarized in Fig. 7. The zero-loss rate-distortion characteristics are given in Fig. 7-(a) and (b) for *Foreman* and *Mother and Daughter*, respectively. Since the "CheckerBoard" and "AlternateLines" videos are losslessly modified from the original "QuickTime" video, the corresponding operating points achieve identical PSNR, though the FMO versions require more bits. In generating these graphs, we include a copy of picture parameter set with each slice to ensure the slices are decodable when received. This is a conservative approach since the picture parameter sets (PPS) can actually be reused for different pictures with identical slice maps, instead of transmitting multiple copies as in these experiments.

Figures 7 (c)-(d) show the PSNR when the slices of the different video are subject to random loss of 1% irrespective of their coded size. The figures represent average results over 40 loss traces. As expected, we see that a large PSNR improvement is achieved by employing FMO. Specifically, for *Foreman*, "CheckerBoard" and "AlternateLines" achieves a PSNR improvement over "QuickTime" of about 2 dB and 1.3 dB, respectively. For *Mother and Daughter*, the gain is smaller, between 1 dB and 0.5 dB, due to little motion which is easily concealed by the motion-compensated concealment of JM 10.1.

The above slice-loss results do not consider the effects of packetization when a coded slice is too large to be transported within a single transport packet. In the following, we



(a) Foreman: no loss (b) Mother: no loss

(c) Foreman: 1% slice loss (d) Mother: 1% slice loss

(e) Foreman: 1% packet loss (f) Mother: 1% packet loss

(g) Foreman: 1% packet loss (h) Mother: 1% packet loss

Fig. 7. Comparison under different loss conditions for QuickTime generated videos with/without losslessly added FMO, and with slices modified to fit transport packet sizes.

assume a maximum packet size of 1450 bytes so that larger coded slices are fragmented into multiple transport packets, and that a slice is decodable only when all of its constituent packets are received. The PSNR for the different videos at 1% *packet* loss rate are shown in Fig. 7-(e) and (f). We see that the gain in PSNR of "Checkerboard" and "AlternateLines" over "QuickTime" is comparable to that under slice-loss, but we also observe that PSNR tends to *decrease* with increasing bit-rate at the higher bit-rates. This "paradoxical" effect results from the larger number of packets needed to transport one slice, which reduces the probability that the complete slice is received and therefore decodable. This suggests that at higher bit rates, it is desirable to modify the slice map so that each coded slice is roughly the size of the maximum packet size. This is achieved by employing our scheme to losslessly form slices smaller than a specified size, and the results are shown as "Split" in Fig. 7-(g) and (h), where curves in Fig. 7-(e) and (f) are shown in grey. We see that for *Mother and Daughter*, the PSNR of "split" does not decrease with increasing data rate as expected. For *Foreman*, certain parts of the video sequence contain a large region of spatially contiguous intra-coded MBs. As discussed in Section 2.1, this prevents us from breaking that region into multiple slices, causing a dip in PSNR at high data rates, as observed in other schemes. Further improvement is given by the "split-mix" scheme of Fig. 7-(g) and (h) under which we attempt to use one row of MB as a slice. A transport packet is then formed by aggregating multiple non-contiguous slices. We see that "split-mix" outperforms "split" and "AlternateLines" as it improves error concealment over the former and reduces the use of slices larger than the maximum packet size when compared to the latter.

5. CONCLUSIONS

We present a scheme that losslessly modifies the slice structure of compressed H.264 video. The scheme allows trading off error resilience and compression-efficiency at streaming time if desired, rather than when the content is created. Notably, this transcoder allows FMO to be added to a compressed bitstream. Such an operation involves recomputing and re-encoding different quantities, such as quantization parameters or motion vectors that are coded differentially in the bitstream. We show that for CIF video at 30 f/s our approach results in negligible overhead, compared to direct encoding of FMO, at medium to high bit-rates, but represents over 1 dB loss in compression efficiency at low bit-rates. Our experiments over a packet erasure channel show that adding FMO to a compressed bitstream may improve the decoded video quality by over 2 dB in PSNR. We also show that by adjusting the size of a coded slice to fit in a transport packet, improvements over FMO schemes can be achieved. We demonstrate that the two strategies above can be combined for further improvement. We believe that the ability to efficiently and losslessly add error-resilience to a previously encoded H.264 video can be useful in a variety of situations.

6. REFERENCES

[1] ITU-T and ISO/IEC JTC 1, *Advanced Video Coding for Generic Audiovisual services, ITU-T Recommendation H.264 - ISO/IEC 14496-10(AVC)*, 2003.

[2] P. Lambert, W. De Neve, Y. Dhondt, and R. Van de Walle, "Flexible macroblock ordering in H.264/AVC," *J. of Visual Communication & Image Representation*, vol. 17, pp. 358 – 375, 1 2006.

[3] S. Wenger, "H.264/AVC over IP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 645–656, July 2003.

[4] G. Reyes, A. Reibman, S. Chang, and J. Chuang, "Error-resilient transcoding for video over wireless channels," *Journal on Selected Areas in Comm.*, vol. 18, no. 6, 2000.

[5] "H.264/AVC Reference Software," *http://iphome.hhi.de/suehring/ tml/download/, seen on Aug. 28 2005*.