SELF-CONTAINED SLICES IN H.264 FOR PARTIAL VIDEO DECODING TARGETING 3D LIGHT-FIELD DISPLAYS

Alireza Zare[1], Péter Tamás Kovács[1,2], Atanas Gotchev[1]

[1]Department of Signal Processing, Tampere University of Technology, Tampere, Finland
[2]Holografika, Budapest, Hungary

ABSTRACT

Some video applications work with parallel bitstreams however only parts of them are required. Decoding the streams entirely might be prohibitive due to high number of video streams and / or high resolution or bitrate of the streams. In such cases, a random access also referred as to partial decoding has to be supported. It can be achieved with a conforming decoder, if the video is separated into individual slices, and the unnecessary slices are dropped from the bit stream. Such a feature is essential in 3D light-field displays in which partial decoding of bitstreams helps providing less complex and thus real-time decoding and processing.

This paper studies the problem of creating self-contained (i.e., independently decodable) slices in an H.264 stream. The requirements that need to be fulfilled in order to build self-contained slices are described, and an encoder-side solution is proposed. The advantages of self-contained slices are examined in the context of a light-field rendering application that relies on partial decoding.

Index Terms—H.264/AVC, 3D light-field displays, video slicing, partial decoding, random access

1. INTRODUCTION

3D light-field (LF) displays [1][2][3][4] represent a major step forward in 3D visualization compared to stereoscopic or multiview displays in terms of smooth motion parallax, wide Field Of View (FOV) and depth range. LF displays can show imagery to a multiple naked-eye freely moving observers, providing walk-around capability, much like holograms. This viewing freedom however comes at a cost in terms of complexity and required amount of visual data to properly represent a scene from many angles. When visualizing image-based content (that is, a scene that is represented with a regular set of captured / rendered light rays with no information about scene geometry), the number of views necessary to properly capture sufficiently wide FOV with the angular resolution supported by the display, can be up to hundreds of views. Delivering this amount of views is a challenging task, however decompressing and processing them is an even bigger challenge. In fact, decoding a certain number of views of a given resolution on a single computer can become prohibitive, considering the video resolution and decoder speed. However, decoding all views completely is not even necessary. As it has been shown in [5], when processing light fields in a distributed system, access patterns in ray space are quite regular, some processing nodes do not need all views, moreover the necessary views are used only partially. This could be exploited to save decoding time, thus enabling real-time operation. While skipping the decoding of whole views is straightforward, partial decoding of frames is typically not supported by video decoders.

In the H.264 standard, similar to earlier standards, a video is described as a series of frames. A frame can be split into several slices, where a slice is a group of macroblocks which are formed by blocks of pixels. An encoder can be configured to use only one slice per frame (in which case all macroblocks of the frame constitute a slice), or to divide the frame into an arbitrary number of slices, with various possible arrangements. If some slices are lost or discarded during transmission, a conforming decoder can still decode the remaining slices. This behavior can be used for achieving partial decoding of video frames by relying on off-the shelf hardware decoders (e.g. those included in GPUs), without changing the decoder, and only manipulating the bitstream. If one intentionally loses slices that do not need to be decoded, a conforming decoder is supposed to be able to decode the regions of the frame that are kept. Thus, slicing facilitates partial decoding which is instrumental for feeding LF displays with live imagery.

In the assumed setting, slicing is applied to subdivide frames into vertical stripes, thus enabling partial decompression of frames and resulting in reduced decoding time. Breaking a video stream into slices thus enables random access into portions of the video stream with a specific granularity.

2. SLICING

2.1. Slicing in H.264

Slices contain varying number of macroblocks which are processed in raster scan order or some other scanning pattern such as interleaved and checker-board slices when Flexible Macroblocks Ordering (FMO) is enabled. Slices are named according to coding type and can be coded using intra-frame and/or inter frame prediction. An I slice contains only I macroblock type, a P slice contains P and I macroblock types and a B slice contain B and I macroblock types.

The main purpose of introducing slices is to add robustness to the encoded bitstream in the presence of transmission errors. Slice structure coding improves transport efficiency and error resilience by providing small decodable chunks for transmission and supporting Arbitrary Slice Order (ASO), FMO and redundant slice tools. Smaller network packets and fixed-size network packets are achieved by keeping the number of bytes per slice roughly constant.

Besides the advantages of slicing mentioned above, the increased number of slices reduces PSNR to bitrate ratio because

978-1-4673-8090-4/15/$31.00

of some extra packetization overhead and prediction limited to slice boundaries, which causes lower level of spatial and temporal correlation [6]. Figure 2 illustrates variation of PSNR with respect to different number of slices. In addition, the usage of standard slicing is limited in some applications such as real time editing bitstream [7] and partial decoding [9], specifically in 3D LF displays. Hence, some extra requirements must be imposed to standard slices in order to be applicable in the abovementioned applications. In the following sections those requirements are introduced and the resulting slicing is discussed in more detail.

2. 2. Self-Contained Slices

A slice is self-contained in the sense that its syntax elements can be parsed from the bitstream and it contains all data which is required on the decoder side. This is true, by considering the fact that correctly decoding of slice requires sequence and picture parameter sets to be received by decoder. Although slices are self-contained, they do depend on the previous frame(s) to be correctly decoded because of inter-frame prediction. In other words, the correct decoding of one slice depends on decoding of previous frames so that they must be entirely available. Losing one slice affects the decoding of all following slices in which case their motion vectors point to some regions of previous frames, which have lost. Furthermore, losing of an entire slice makes the first coded macroblock number, defined by the "first_mb_in_slice" field, unavailable. As a result, the synchronization between the encoder and the decoder gets lost and the errors propagate in the subsequent decoded bitstream caused by the H.264 variable length coding property [9]. Figure 3 shows the decoding result of a video, in which some slices are lost when motion vectors are not restricted to slice boundaries. In this case, some requirements have to be imposed at the encoder in order to eliminate the dependency between the current slice and the previous frames and to mitigate the error propagation.

The first requirement is restricting intra-frame prediction to within current slice boundaries. The usage of slices already restricts intra-frame prediction. The second requirement is using regular slice partition so that each slice occupies the same spatial region over all frames. This is obtained by partitioning each frame with the same number of slices and defining slice argument to a fix number of macroblocks leading to regular rectangular slicing. Our goal of specifying independently decodable slices can be achieved by meeting these two requirements provided all frames are selected as I-frames at the expense of considerable increase in bitrate. As a third requirement, inter-frame prediction must be also limited to within slice boundaries in order to provide both spatially and temporally independently decodable regions [7]. In this case, it is guaranteed that motion vectors only point to the same spatial area as current slice in reference frames. We modified the reference encoder to restrict motion estimation to the same spatial area in the previous frames.

These kind of self-contained slices have several advantages. Most notably, in contrast to standard slices they can be encoded and decoded independently. Hence, losing (skipping) one or more slices leaves the other areas of a frame unaffected and the remaining slices can still be decoded. In addition, frames of a video sequence can be partitioned to slices to enable partial and parallel decompression of frames.

3. PARSING PROCESS

In case of partitioning each frame into multiple slices, one can consider the slices as self-contained units, in the sense that they are independently decodable, if all requirements introduced in the previous section are fulfilled. In this case, each slice depends on co-located slices in reference frames. In order to illustrate the scenario, we parse the bitstream for encoded slices and drop bitstream elements corresponding to slices that we are not intending to decode. The remaining bitstream is then decoded. The parsing process requires understanding of the bitstream syntax. A bitstream can be seen as a collection of Network Abstract Layer (NAL) packets in decoding order where each NAL packet contains a start code prefix. In the next section, the bitstream format and the usage of start code in slice parsing process are explained.

3. 1. Bitstream Format

The encoded stream in H.264 is represented in a clearly defined syntax. In the highest level: the sequence level, H.264 stream is seen as a series of NAL units starting with parameter sets. Parameter sets contain two NAL units, Sequence Parameter Set and Picture Parameter Set, which contain common control parameter and are necessary for the decoder to correctly decode the bitstream. In the next layer: the slice layer, there are coded slices. Each coded slice known as Video Coding Layer NAL unit consists of a slice header and slice data. The slice header section holds common information applicable to all macroblocks in the slice such as the number of first macroblock in the slice, slice type and frame number that slice belongs to. For example, slice_type "110" determines that each macroblock in the slice can be either of type I or P. The slice data consists of a series of coded macroblocks. The size of each encoded slice tends to vary depending on the number of macroblocks, the amount of motion and the details included in the slice spatial area. I slices are typically coded with higher number of bits because of the fact that intra prediction tends to be less efficient than inter prediction, followed by P slices and then B slices. In the lowest level of the hierarchical structure a coded macroblock is represented. Each coded macroblock section contains all elements such as macroblock type and prediction mode necessary to decode a single macroblock which makes up a spatial block of a frame.

Each NAL unit in the bitstream is separated by some specific start codes or delimiters that define boundaries between coded sections. Delimiters facilitate accessing coded units in level of frame or slice. In H.264 delimiters are unique sequences of four bytes equal to 0x00000001 and three bytes equal to 0x000001 in level of frame and slice respectivelly. A low level of a typical bitstream with delimiters is shown in Figure 1.

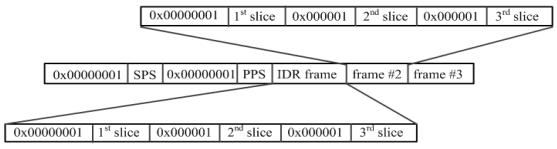

Figure 1. A typical H.264 bitstream

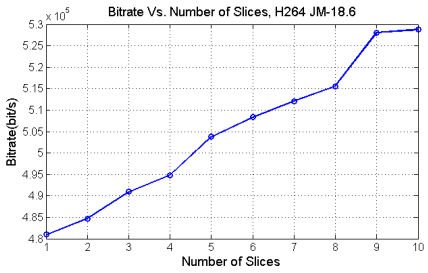

Figure 2. Bitrate vs. number of slices, Balloons sequence, 10 frames

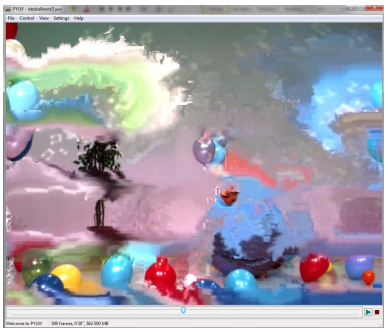

Figure 3. Decoded video after dropping every other slices from encoded stream using standard slices

3.2. Approach

In our experiment we target slices, which can be decoded independently by a conforming decoder. The requirements for self-contained slices as explained in Section 2.2 have been achieved by modifying the JM H.264 reference encoder (version 18.6). Each frame is divided to the same number of slices in which motion estimation is limited to the same spatial area covered by slice in reference frames. In order to restrict the motion estimation, slice boundaries are determined and every time after motion estimation and before motion to be compensated the validity of motion vector is examined with respect to slice boundaries. In case a motion vector points to somewhere out of slice boundaries, it is discarded. A new configuration parameter called "selfSlice" is defined so that when it is enabled the changes to obtain self-contained slices are applied otherwise the standard slice subdivision is performed.

To perform the slice dropping experiment, first the video is encoded using our modified encoder. Every other coded slice including a start code and multiple coded macroblocks is then parsed and cropped from the bitstream in order to obtain erroneous bitstream. The resulting bitstream is then decoded using standard decoder.

4. RESULTS AND COMPARISON

We initially consider partial decoding of a compressed video in which frames are split into 6 standard slices in a situation where every other slice is dropped before decoding. As it is illustrated in Figure 3, lost NAL units cause decoding of other regions of the frame incorrectly. This is mainly due to the motion compensation, which is pulling in garbage from the undecoded / lost image regions. This occurs because no restriction on motion prediction is imposed at the encoder. Motion vectors point to some region in reference frames, which do not exist at the decoding time, and are filled by the decoder with bogus pixels. The error propagation is growing as higher number of frames is being decoded.

The required functionalities mentioned in Section 2.3 are implemented in the reference encoder to obtain independent slices. In the case of independent slices, each slice is decoded without requiring existence of other slices (of course, in other spatial regions than what is covered by the slice) from reference frame(s). Figure 4 shows partial decoding of the same compressed video but this time containing independent slices. It can be seen that losing bitstream elements corresponding to the dropped slices leave the decoding of the remaining slices unaffected.

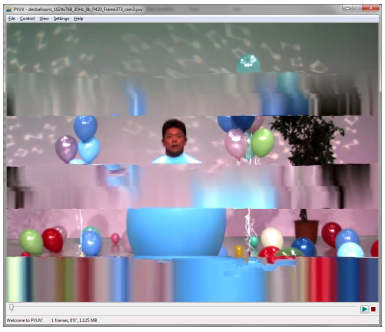

Figure 4. Decoded video after dropping every other slices from encoded stream using self-contained slices

A comparison has been made between the H.264 standard slicing and the proposed independent slicing to evaluate the effect of independent slices to the compression efficiency. The video sequences are selected from different levels of object motion. The RD curves in comparison are shown in Figures 5, 6 and 7 for three different video sequences containing low, medium and fast motions respectively. The results show that the reduction in compression efficiency depends on the video content and the level of motions within the video. In other words, it is determined by the amount of temporal correlation lowering caused by the independent slicing in contrast with standard slicing. A bitrate numerical comparison is also made for the three video sequences for QP value 28 in Table 1. Although, the Balloons sequence contains more complex object motion self-contained slices cause more bitrate increment in the Pantomime sequence. That is because of higher reduction of temporal correlation in the latter sequence.

Table1. Bitrate (MB/s) comparison, frames 1-10, QP = 28

Sequence	Resolution	standard slice (MB/s)	Self-contained slice (MB/s)
Big Buck Bunny	1280x768	2.395	2.396
Pantomime	1280x960	3.20	4.95
Balloons	1024x768	1.66	1.81

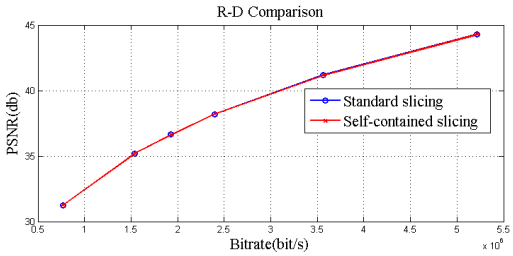

Figure 5. Rate-Distortion Comparison, Big Buck Bunny sequence, 10 frames

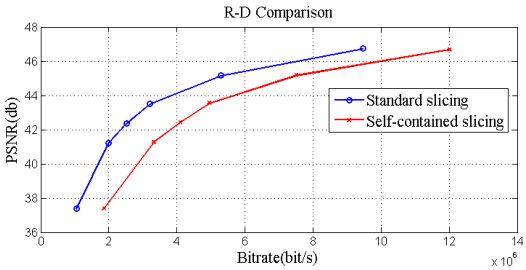

Figure 6. Rate-Distortion Comparison, Pantomime sequence, 10 frames

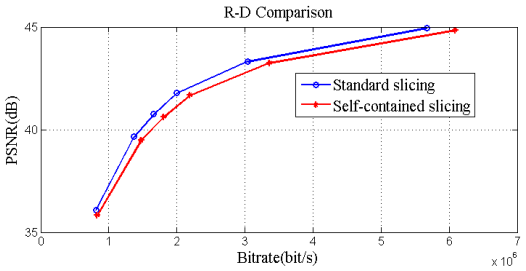

Figure 7. Rate-Distortion Comparison, Balloons sequence, 10 frames

The results indicate that the introduced self-contained slices have a low impact on the compression efficiency so that it is worth employing them in applications where partial decoding or random access are desired, with a minor increase in bitrate (depending on video content and number of slices).

5. CONCLUSION AND FUTURE WORK

This paper has focused on the creation of self-contained slices in H.264 and its application in 3D light-field displays. H.264 has been chosen of this experiment, as hardware-based H.264 decoders are readily available in computers/ GPUs today, which can be leveraged for real-time implementation of the decoder side of the proposed technique. The paper has formulated the requirements for self-contained slices that are needed to enable partial decoding of the bitstream. The effect of the introduced slices on compression efficiency has been evaluated. The results indicate that the proposed self-contained slices facilitate partial and parallel decoding with low overhead which is caused by the extra packetization and the imposed restriction on motion prediction. The simulation results have shown that the required restrictions leading to self-contained slices do not cause significant performance penalty or visible coding artifacts.

In our future work we will focus on the emerging High Efficiency Video Coding (HEVC) in order to implement partial decoding by using tiles. Tiles have been introduced as spatially independent units to the encoding process. In order to make them self-contained in our context, motion vectors for Coding Tree Units (CTUs) have to be also constrained inside tile boundaries to remain in the co-located tiles in the reference frames. Therefore, some modifications of both standard encoder and decoder are expected in order to obtain partial decoding for feeding 3D LF displays. So far off-the-shelf decoders fail to expose an interface for decoding specific tiles, thus a bitstream manipulation in a decoder modification might be necessary in order to achieve this goal.

6. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the PROLIGHT-IAPP Marie Curie Action of the People programme of the European Union's Seventh Framework Programme, REA grant agreement 32449.

7. REFERENCES

[1] T. Balogh, P. T. Kovács and Z. Megyesi, "Holovizio 3D display system," in *proc. IMMERSCOM 2007*, 2007.

[2] Wetzstein G. and et al., "Tensor displays: compressive light field synthesis using multilayer displays with directional backlighting," in *proc. SIGGRAPH 2012*, 2012.

[3] Lee J.-H. and et al., "Optimal projector configuration design for 300-mpixel light-field 3D display," *Optics Express*, vol. 21, issue 22, pp. 26820-26835, 2013.

[4] Kawakita M., et al., "Glasses-free 200-view 3D video system for highly realistic communication," in *proc. Digital Holography and Three-Dimensional Imaging*, *OSA Technical Digest*, paper DM2A.1, 2013.

[5] P. T. Kovács, Z. Nagy, A. Barsi, V. K. Adhikarla and R. Bregović, "Overview of the applicability of H.264/MVC for real-time light-field applications," in *proc. 3DTV-CON 2014*, Budapest, Hungary, 2014.

[6] Jegadish Devadoss, Varun Singh, Jörg Ott, Chenghao Liu, Ye-Kui Wang, Igor D. D. Curcio, "Evaluation of error resilience mechanisms for 3G conversational video," *Tenth IEEE International Symposium on Multimedia*, pp. 378-383, California, USA, Dec. 2008.

[7] Vehkapera Janne and Tomperi Seppo, "Replacing picture regions in H.264/AVC bitstream by utilizing independent slices," in *proc. IEEE 17th International Conference on Image Processing*, Hong Kong, Sept. 2010, pp. 3397-3400.

[8] Thomas Wiegand, Gray J. Sullivan, Gisle Bjontegaard and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560-576, July 2003.

[9] Peter Quax, Fabian Di Fiore, Panagiotis Issaris, Wim Lamotte and Frank Van Reeth, "Practical and Scalable Transmission of Segmented Video Sequences to Multiple Players using H.264," *Motion in Games 2009 (MIG09), Lecture Notes in Computer Science LNCS series, LNCS 5884*, pp. 256-267, 2009.

[10] Kiran M. Misra, C. Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth and Minhua Zhou, "An overview of tiles," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 969-977, Dec. 2013.