OVERVIEW OF THE APPLICABILITY OF H.264/MVC FOR REAL-TIME LIGHT-FIELD APPLICATIONS

Péter Tamás Kovács [1, 2], Zsolt Nagy [1], Attila Barsi [1], Vamsi Kiran Adhikarla [1,3], Robert Bregović [2]

[1]Holografika, Budapest, Hungary
[2]Department of Signal Processing, Tampere University of Technology, Tampere, Finland
[3]Pazmany Peter Catholic University, Faculty of information Technology, Budapest, Hungary

ABSTRACT

Several methods for compressing light-fields (LF) and multiview 3D video content have been proposed in the literature. The most widely accepted and standardized method is the Multi View Coding (MVC) extension of H.264, which is considered appropriate for use with stereoscopic and multiview 3D displays. In this paper we will focus on light-field 3D displays, outline typical use cases for such displays, analyze processing requirements for display-specific and display-independent light-fields, and see how these map to MVC as the underlying 3D video compression method. We also provide an overview of available MVC implementations, and the support these provide for multiview 3D video. Directions for future research and additional features supporting LF video compression are presented.

Index Terms **—** light-field, 3D video, compression, multi-view coding, MVC, H.264

1. INTRODUCTION

Future 3D displays will go far beyond stereoscopic and multi-view, as demonstrated in currently existing prototype and commercial 3D displays [1][2][3]. Some of the existing displays aim to reproduce light-fields having both horizontal and vertical parallax, while others omit vertical parallax in order to provide better resolution and higher number of viewing directions horizontally, typically resulting in wider horizontal Field Of View (FOV) for the same number of light rays.

Wide-angle LF displays may have hundreds of viewing directions, but typically only in the horizontal direction (Horizontal Parallax Only, HPO). To achieve wide field-of-view and still maintain a reasonable resolution, these displays operate with large pixel counts (nowadays, up to 100 megapixels). The storage, compression, transmission and rendering of light-fields of this size is a major challenge, which needs to be solved to pave the way towards the wide adoption of such advanced 3D display technologies.

There have been a lot of effort directed towards supporting 3D displays with effective 3D video compression standards [4][5]. In this paper we give an insight into the computational background of LF displays, and analyze how the results of standardized 3D video coding technology can be exploited. Based on this analysis, we identify areas that need attention in future research in 3D LF video coding. In this paper we focus on H.264/MVC, since that is the current accepted standard for coding 3D video data, and is more likely to have mature implementations than work-in-progress 3D HEVC.

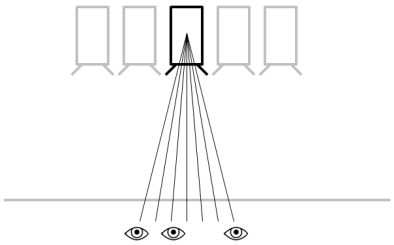

Figure 1. Light rays emitted by a single projection module are spread over screen positions and viewing directions, thus cannot be seen from a single viewing position

2. LF DISPLAY ARCHITECTURE

We focus our discussion on HoloVizio light-field displays [1], but the results presented in this paper are directly applicable to any LF display that is driven by a distributed projection and rendering system. Considering the gap between pixel / light ray counts and the rendering capacity available in a single computer / GPU, using a distributed rendering system for these systems is a necessity today and in the foreseeable future. Therefore LF displays are typically driven by multiple processing nodes.

LF displays are capable of providing 3D images with a continuous motion parallax on a wide viewing zone, without wearing glasses. Instead of showing separate 2D views of a 3D scene, they reconstruct the 3D light field as a set of light rays. In most LF displays this is achieved by using an array of projection modules emitting light rays and a custom made holographic screen. The light rays generated in the projection modules hit the holographic screen at different points and the holographic screen makes the optical transformation to compose these light rays into a continuous 3D view. Each point of the holographic screen emits light rays of different color to various directions.

Light rays leaving the screen spread in multiple directions, as if they were emitted from points of 3D objects at fixed spatial locations. However, the most important characteristic of this distributed projection architecture is that the individual projection modules do not correspond to discrete perspective views, in the way *views* are defined in a typical multi-view setting. What the projection modules require on their input depends on the exact layout of the LF display, but in general, a single projection module is responsible for light rays emitted at different screen positions, and in different directions at all those positions. The whole image projected by a single projection module cannot be seen from a single viewing position, as shown on Figure 1. As such, one projection module represents a *LF slice*, which is composed of many image fragments that will be perceived from different viewing positions.

978-1-4799-4758-4/14/$31.00

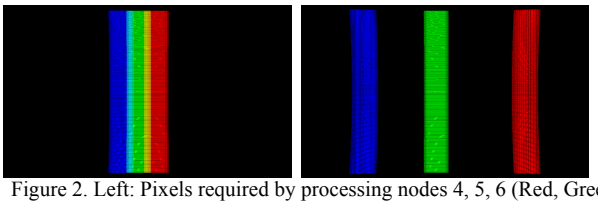

Figure 2. Left: Pixels required by processing nodes 4, 5, 6 (Red, Green and Blue channels). Right: Pixels required by processing nodes 0, 5, 9 (Red, Green and Blue channels)

Although these LF slices can be composed based on the known geometry of a multi-camera setup and the geometry of the LF display, this mapping is nonlinear and typically requires accessing light rays from a large number of views, even when generating the image for a single projection module.

The layout of the typical rendering cluster, made up of processing nodes (nodes for short), is such that a single computer is attached to multiple projection modules (2, 4, 8 or more), and as such, a single computer is responsible for generating adjacent LF slices. During LF conversion, individual nodes do not require all the views, nor all the pixels from these views. Although there is some overlap between the camera pixels required by nodes, those that are responsible for distant parts of the overall light-field require a disjoint set of pixels from the camera images.

To demonstrate this arrangement visually, Figure 2 shows which parts of the input perspective views are actually required for generating specific LF slices. A simulation has been run on a 45º large-scale light-field display with 80 projection modules, which has 10 processing nodes for generating the light-field. The display has been fed with 91-view input. What we can see is that adjacent processing nodes use adjacent, somewhat overlapping parts of the views, while processing nodes that are further away in the sense of LF slices will require completely different parts of the same view to synthesize the light field. These results are shown for the central camera, the pattern for other views is similar.

3. USE CASES

Two general use cases are defined to evaluate the applicability of specific 3D video coding tools, as the requirements imposed by these use cases are substantially different. The use cases identified by MPEG [6][7] can be classified into one of these, depending on whether the content is stored / transmitted in a display-specific or display-independent format. In both use cases, the requirement for real-time playback (as seen by the viewers) is above all other requirements.

The first and least demanding use case is *playback of pre-processed LF* content. In this case content has been prepared for a specific LF display model in advance, and must be played back in real time. In this setting the content is stored in *display specific LF* format. Display specific LF means the light rays are stored in a way that the individual slices of the full LF already correspond to the physical layout (projection modules) of the display on which the content should be played back. In other words, the LF in this case has already gone through the ray interpolation step that transforms it from camera space to display space. The implication is that the LF slices correspond to the layout of the distributed system driving the LF display, and as such, no ray interpolation is needed during playback, and no image data needs to be exchanged between nodes. As an example, in case of an 80-channel LF display, we may consider this data to be 80 separate images or videos making up a 3D image or video, for example 80 times WXGA (~78 MPixels).

The second use case we consider is *broadcast LF video* transmission, with the possibility to target different LF displays. 3D LF displays can differ in multiple properties, but spatial resolution and FOV have the most substantial effect on the content. The goal is to support different LF displays with the same video stream in a scalable way. In order to support different displays, we need to use *display independent LF*, which is not parametrized by display terms, but using some other terms (for example capture cameras), which is subsequently processed on the display side during playback. In this paper we consider this display independent LF to be a set of perspective images representing a scene from a number of viewpoints. Please note there are many other device-independent LF representations which lay between these two, however these two are the closest to practical hardware setups (camera rigs and LF displays).

The analysis that follows focuses on the decoder / display side, and does not consider encoder complexity.

4. PROCESSING DISPLAY-SPECIFIC LIGHT-FIELDS

In this case, as LF preprocessing is performed offline, the encoding process is not time critical, i.e. there is no real-time requirement for the encoder. Visual quality should be maximized wrt. bitrate, to be able to store the largest amount of LF video. On the decoding side, the goal is to be able to decompress separately the LF slices that correspond to the individual projection engines contained in the display, in real-time. The simplest solution to this problem is simulcoding all the LF slices independently using a 2D video codec (ie. H.264), and distribute the decoding task to the processing nodes corresponding to the mapping between processing nodes and projection engines. Take 80 optical engines and 10 nodes as an example: if all nodes are able to decompress 8 videos in real-time, simultaneously, we have a working solution (provided we can maintain synchronized playback). The complexity of H.264 decoding typically allows running several decoders on a high-end PC, and 25 FPS can be achieved. This solution is currently used in production LF displays.

However, in this case we do not exploit similarities between the LF slice images which have similar features, like multiview imagery. On the other extreme, compressing all 80 LF streams with MVC would require that a single processing node can decompress all of them simultaneously in real-time, which is typically prohibitive. The complexity of MVC decoding is expected to increase linearly with the number of views in terms of computing power. Furthermore it also requires a larger Decoded Picture Buffer (DPB) depending on the number of views. Assuming that having enough RAM for the DPB is not an issue, decoding a 80-view MV stream on a single node in real-time is still an issue, especially as there is no real-time implementation available that can perform this task (see Section 7). Even considering parallelization techniques [8], decoding all views in real-time on a single node is out of reach.

A reasonable tradeoff is to compress as many LF module images that are mapped to a single processing element, and do this as many times as necessary to contain all the views. As an example, we may use 10 separate MVC streams, each having 8 LF slices inside. We can increase the number of views contained in one MVC stream as long as a single processing node can maintain real-time decoding speed.

5. PROCESSING DISPLAY-INDEPENDENT LIGHT-FIELDS

As discussed in Section 2, and in [9], not all views are required for interpolating a specific LF slice, and even from these views, only parts are required to generate the desired LF slice – some regions of the camera images might even be left unused.

FOV (degrees)	27	38	48	59	69	79	89
No. views used	42	44	46	48	50	52	54

Table 1. Number of views used overall for LF synthesis when targeting LF displays with different FOV.

To find out how much we can bound the number of views and pixels to be compressed, we may determine the images and image regions which are actually used during the LF interpolation process, and compress only those for the targeted display. However, assuming receivers with displays with different viewing capabilities makes such an approach impractical, and requires scalability in terms of spatial resolution and FOV. Difference in spatial resolution might be effectively handled by SVC, and is not discussed further here. The differences in FOV however have not been addressed, as studies on the effect of display FOV on the source data used for LF conversion have not been performed so far.

We have performed simulations to see how the FOV of the receiver's LF display affects the way the available captured views are used. We have modeled 7 hyphotetical LF displays, with the FOV ranging between 27º and 89º. Source data with 180 cameras, in a 180º arc setup, with 1 degree angular resolution has been used. Using the tool from [9] and analyzing the pixel usage patterns, we have analyzed how the display's FOV affects the number of views required for synthesizing the whole LF image. This analysis has shown that depending on the FOV of the display, the LF conversion requires 42 to 54 views as input for these sample displays, as seen in Table 1. Please note the actual number depends on the source camera layout (number and FOV of cameras), but the trend is clearly visible.

Looking at the images representing the pixels read from each view also reveals that for most views, only small portions of the view are used, which is especially true for side views. This can be intuitively seen if we consider a 3D display with a wide viewing angle, looking at the screen from a steep angle. In this case, we can only see a narrow image under a small viewing angle – this is also what we need to capture and transmit. This observation suggests that any coding scheme targeting multi-view video on LF displays should be capable of encoding multiple views with different resolution. In case of HPO LF displays, only the horizontal resolution changes. In full parallax setups, both horizontal and vertical resolutions change. Such flexibility is not supported by MVC.

Due to the fact that distributed processing nodes are responsible for different parts of the overall LF, these units require different parts of the incoming views (as seen in Section 2). Thus we may expect that the number of views necessary for one node is lower than for the whole display. Further analyzing pixel usage patterns and separating the parts required by distinct nodes, we can see that this number is indeed lower, however not significantly lower. For example, in case of the 89° FOV display, instead of the 54 views required for the whole LF, one node requires access to 38 views on average, which is still high - decompressing these many full views is a challenge.

As seen previously, not all pixels from these views are necessary to construct the LF. If we look at the patterns showing which regions of the views captured by the cameras are used for the LF conversion process when targeting LF displays with different FOVs, we can see that the area is pointing to the scene center, and is widening with the increased FOV, see Figure 3.

This property may be used to decrease the computational complexity of decoding many views, by decoding only regions of interest for the specific display. H.264 supports dividing the image into regions to distinctly decodable regions using slice groups, however this feature is typically targeted to achieve some level of parallelism in the decoding process. By defining individually decodable slice groups that subdivide the image into

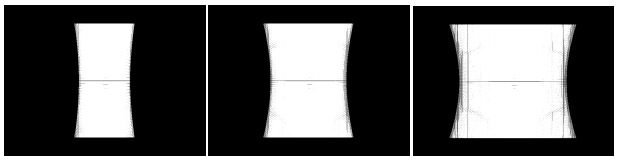

Figure 3. Image regions used from the central camera, by the 27º (left), 59º(center) and 89º (right) LF displays.

vertical regions, and decoding only those required, it is possible to decrease the time required to decode the views. Defining several slice groups would give enough granularity to target a wide range of displays with little overhead.

On the other hand, by separating views into vertical slices, we lose some coding gain due to motion estimation / compensation not going across slice boundaries. Some of this loss might be recovered by using prediction from the center of views to the sides, however such hierarchies are not supported. Exploiting this possibility is an area of future research.

6. NONLINEAR CAMERA SETUPS

With the emergence of LF displays with extremely wide FOV, it is more and more apparent that an equidistant linear camera array cannot capture the visual information necessary to represent the scene from all around. A more suitable setup is an arc of cameras, facing the center of the scene. Compressing such captured information with MVC should also be efficient, as the views captured in this manner also bear more similarity than views captured by a linear camera array.

However, the kind of pixel-precise inter-view similarity that MVC implicitly assumes only exist when using parallel cameras on a linear rig, and assuming Lambertian surfaces. It has been shown [10] that the coding gain from inter-view prediction is significantly less for arc cameras than for linear cameras.

Due to the emergence of wide-FOV 3D displays it is expected that non-linear multiview setups will be more significant in the future. Coding tools to support the efficient coding of views rotating around the scene center should be explored, and the similarities inherent in such views exploited for additional coding gains.

7. OVERVIEW OF MVC IMPLEMENTATIONS

The features discussed above can be embedded into the systems supporting LF displays if there exists implementations that support real-time operation.

MVC is the compression method of choice for 3D Blu-ray disks, where it is used for encoding the stereoscopic pair more efficiently than simulcasting the two views. Due to this widespread use of the Stereo High Profile of MVC, there are several implementations supporting it. However, support for real-time encoding and decoding of Multiview High Profile with more than two views is very weak, practically nonexistent.

JM 18.6 [11], the latest H.264/AVC reference software supports MVC, but only up to 2 views, which seems to be a hard coded limit. On the other hand it supports the specification of GOP structure explicitly, thus by interleaving frames from multiple views, it is possible to use it for inter-view prediction. It further allows the specification of arbitrary slice groups. Being a reference implementation however, its performance is typically below real-time. When running a single instance of the encoder / decoder, multiple CPU cores are not utilized, however it is possible to run parallel instances of the encoder / decoder during simulcoding, as in this case instances can run independently. Still, due to its low processing speed, this software cannot be utilized in real applications.

JMVC 8.5 [12], the latest H.264/MVC reference software naturally supports MVC with arbitrary number of views. Being a reference implementation, its runtime performance is low, similar to JM. Unlike JM however, depending on setup of inter-view prediction, encoder / decoder instances have to be executed sequentially for each view, and cannot be parallelized, as the dependent views rely on the reconstructed images output by the encoder in previous run. Parallelizing MVC encoding by partially delaying dependent views is possible [8], however this alone does not make JMVC real-time.

x264 [13] the popular, open source implementation of H.264 is considered the fastest pure-software H.264 codec. While it provides real-time encoding and decoding performance for high-resolution 2D videos, it does not support MVC, nor the specification of custom GOP structures to emulate inter-view prediction. Slicing is supported, but only for the purposes of parallel processing – the shape of slice groups cannot be defined externally.

NVENC [14] is a pure-hardware H.264 codec embedded in high-end Nvidia GPUs. It supports faster than real-time 2D video encoding / decoding for very high resolution videos, and it also supports MVC for up to two views. Nvidia does not have plans to extend it to multiple views. Using custom prediction structures and slicing along vertical blocks are not supported.

The DXVA MVC Specification [15] mentions support for the Multiview High Profile, however we have not seen any implementation of this in the latest Windows SDK.

As of commercial H.264 SDKs, we have found only one from MainConcept MVC/3D codec [16], which, according to the publicly available material supports decoding MVC for up to 10 views, but on the encoding side, only Stereo profile is supported.

IP cores (for embedding in hardware codecs in FPGAs or ASICs) have also been announced with MVC support, mostly for Blu-ray decoding. The announcement of the POWERVR VXD392 / VXE382 cores [17] explicitly mentioned Multiview High Profile, the Video Encoder / Decoder fact sheets however reveal that the final products support 2-view MVC.

There have been several attempts towards integrating MVC into open-source H.264 codecs into ffmpeg [18], and x264 [19] (the latter targeted only stereo), however none of these patches made it to the mainline development branch.

8. CONCLUSIONS AND FUTURE WORK

Based on the use cases and processing considerations described in this paper, we can formulate at least three aspects that need attention and future research when developing compression methods for LFs. First, we shall add the possibility to encode views having different resolution. Secondly, the ability to decode the required number of views should be supported by the ability to decode views partially, starting from the center of the view, thus decreasing the computing workload by restricting the areas of interest. Third, efficient coding tools for nonlinear (curved) camera setups shall be developed, as we expect to see this kind of acquisition format more in the future.

In the future, we will focus on including many-view MVC encoding / decoding into the x264 codec, which will allow us to exploit the possibilities of MVC (at least partially) in the use cases described. Also, the structure of image data and distributed processing requirements suggest that a novel display-independent representation for LFs should be developed, which gathers the necessary image data into a better localized format, instead of having the image data scattered all around views and compressed as such. We will also explore the SoA of HEVC 3D Extension, and how it can be applied to compress LF data.

9. 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.
The research leading to these results has received funding from the DIVA Marie Curie Action of the People programme of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement 290227.

10. REFERENCES

[1] T. Balogh, "The HoloVizio system," Proc. SPIE 6055, *SD&A XIII*, 60550U, 2006

[2] G. Wetzstein, et al, "Tensor Displays: Compressive Light Field Synthesis using Multilayer Displays with Directional Backlighting", In *Proc. SIGGRAPH 2012*

[3] M. Kawakita, et al, "Glasses-free 200-view 3D Video System for Highly Realistic Communication," in *Digital Holography and Three-Dimensional Imaging, OSA Technical Digest*, paper DM2A.1.

[4] A. Vetro, T. Wiegand, G.J. Sullivan, "Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard," *Proceedings of the IEEE*, vol.99, no.4, pp.626,642, April 2011

[5] H. Schwarz, et al, "3D video coding using advanced prediction, depth modeling, and encoder control methods," *Picture Coding Symposium*, 2012

[6] M. P. Tehrani, et al, "Use Cases and Requirements on Free-viewpoint Television (FTV)", *ISO/IEC JTC1/SC29/WG11 MPEG2013/N14104*, October 2013, Geneva, Switzerland

[7] P. T. Kovács et al, "Requirements of Light-field 3D Video Coding", *ISO/IEC JTC1/SC29/WG11 MPEG2014/M31954*, January 2014, San Jose, US

[8] Y. Chen, et al, "The Emerging MVC Standard for 3D Video Services", *EURASIP Journal on Advances in Signal Processing*, Vol. 2009, No. 1, January 2009.

[9] Adhikarla, V.K.; et al, "Fast and efficient data reduction approach for multi-camera light field display telepresence systems," *3DTV-Con 2013*

[10] K. Wegner, et al, "Compression of FTV video with circular camera arrangement", *ISO/IEC JTC1/SC29/WG11 MPEG2014/M33243*, April 2014, Valencia, Spain

[11] H.264/AVC Software Coordination, JM 18.6, http://iphome.hhi.de/suehring/tml/ (visited 14/06/2014)

[12] H.264/MVC Reference Software, JMVC 8.5, cvs:// garcon.ient.rwthaachen.de (visited 14/06/2014)

[13] x264, http://www.videolan.org/developers/x264.html (visited 28/03/2014)

[14] Nvidia Video Codec SDK, https://developer.nvidia.com /nvidia-video-codec-sdk (visited 14/06/2014)

[15] G. J. Sullivan, Y. Wu, "DirectX Video Acceleration Specification for H.264/MPEG-4 AVC Multiview Video Coding (MVC), Including the Stereo High Profile", http://www.microsoft.com/en-us/download/details.aspx? id=25200 (visited 14/06/2014)

[16] MainConcept Video SDK, http://www.mainconcept.com /eu/products/sdks/video/mvc3d.html (visited 14/06/2014)

[17] Imagination's POWERVR VXD392 and VXE382, http://www.imgtec.com/news/Release/index.asp?img_ccc=1 &NewsID=597 (visited 14/06/2014)

[18] J. Britz, "Optimized implementation of an MVC decoder", MSc thesis at Saarland University, 2013

[19] SoC 2011/Stereo high profile MVC encoding, https://wiki.videolan.org/SoC_2011/Stereo_high_profile_m vc_encoding (visited 14/06/2014)