



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALIREZA ZARE
ANALYSIS AND COMPARISON OF MODERN VIDEO
COMPRESSION STANDARDS FOR RANDOM-ACCESS LIGHT-
FIELD COMPRESSION
Master of Science Thesis

Examiner: Prof. Atanas Gotchev
Péter Tamás Kovács
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical
Engineering
on 7th June 2017

ABSTRACT

ALIREZA ZARE: Analysis and Comparison of Modern Video Compression Standards for Random-access Light-field Compression
Tampere University of Technology
Master of Science Thesis, 65 pages, 2 Appendix pages
June 2017
Master's Degree Programme in Information Technology
Major: Signal Processing
Examiner: Professor Atanas Gotchev, and Péter Tamás Kovács

Keywords: video coding, AVC/H.264, HEVC/H.265, random access in video coding, slice, tile, light-field compression

Light-field (LF) 3D displays are anticipated to be the next-generation 3D displays by providing smooth motion parallax, wide field of view (FOV), and higher depth range than the current autostereoscopic displays. The projection-based multi-view LF 3D displays bring the desired new functionalities through a set of projection engines creating light sources for the continuous light field to be created. Such displays require a high number of perspective views as an input to fully exploit the visualization capabilities and viewing angle provided by the LF technology. Delivering, processing and de/compressing this amount of views pose big technical challenges. However, 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 trait could be exploited by partial decoding of pictures to help providing less complex and thus real-time operation.

However, none of the recent video coding standards (e.g., Advanced Video Coding (AVC)/H.264 and High Efficiency Video Coding (HEVC)/H.265 standards) provides partial decoding of video pictures. Such feature can be achieved by partitioning video pictures into partitions that can be processed independently at the cost of lowering the compression efficiency. Examples of such partitioning features introduced by the modern video coding standards include slices and tiles, which enable random access into the video bitstreams with a specific granularity. In addition, some extra requirements have to be imposed on the standard partitioning tools in order to be applicable in the context of partial decoding. This leads to partitions called self-contained which refers to isolated or independently decodable regions in the video pictures.

This work studies the problem of creating self-contained partitions in the conventional AVC/H.264 and HEVC/H.265 standards, and HEVC 3D extensions including multi-view (i.e., MV-HEVC) and 3D (i.e., 3D-HEVC) extensions using slices and tiles, respectively. The requirements that need to be fulfilled in order to build self-contained partitions are described, and an encoder-side solution is proposed. Further, the work examines how slicing/tiling can be used to facilitate random access into the video bitstreams, how the number of slices/tiles affects the compression ratio considering different prediction structures, and how much effect partial decoding has on decoding time.

Overall, the experimental results indicate that the finer the partitioning is, the higher the compression loss occurs. The usage of self-contained partitions makes the decoding operation very efficient and less complex.

PREFACE

This research work was carried out at the Department of Signal Processing at Tampere University of Technology (TUT) during the year 2014-2015. The work is conducted in 3D Media Group at TUT in collaboration with Holografika's research team in Budapest. The work was further developed in Nokia Technologies in Tampere.

I would like to express my gratitude and appreciation to my supervisor Prof. Atanas Gotchev not only for his supports and guidance but also for providing the strong collaboration between TUT and industries. Through that, in my Master thesis, I had chance to solve a real-world problem for industry. I would also like to express my deep appreciation to Péter Tamás Kovács for his supervision and guidance. Throughout my thesis, the significant factor was his ease of access for answering my questions and helping me. I would like to extend special tanks to my colleagues at Nokia specially my supervisors Miska Hannuksela and Alireza Aminlou for their support and precious guidance.

I would be remiss if I did not thank my friends in Tampere specially Khazar Khorrami, Farid Mehrabkhani, Reza Araghi, Gabriel Torres, Jakub Esner, Mojtaba Sarooghi, Amir Shokouhi, Farshad Ahmadi, Sajjad Nouri, and Ramin Ghaznavi.

My acknowledgments would not be completed with heartfelt expression of gratitude to my parents, Yousef and Zaman, and my sisters, Marzieh, Razieh, and Hanieh. Thank you all for unwavering support.

Tampere, 25.8.2015

Alireza Zare

To war children worldwide.

CONTENTS

ABSTRACT	I
PREFACE	II
LIST OF FIGURES	VI
LIST OF TABLES	VII
LIST OF SYMBOLS AND ABBREVIATIONS	VIII
1. INTRODUCTION	9
1.1 Case study: Light-field 3D displays	10
1.2 Related work	12
1.3 Contribution and publication.....	14
1.4 Thesis structure	15
2. BACKGROUND	16
2.1 Basis of video compression.....	16
2.2 Video compression standards.....	18
2.3 Selected aspect of AVC/H.264.....	21
2.3.1 H.264 coding structure	21
2.3.2 H.264 intra and inter-picture prediction.....	22
2.3.3 H.264 frame partitioning feature.....	24
2.3.4 H.264 bitstream syntax	25
2.4 Selected aspect of HEVC/H.265	26
2.4.1 HEVC coding structure	27
2.4.2 HEVC intra- and inter-picture prediction schemes.....	30
2.4.3 HEVC partitioning feature	31
2.4.4 MV and 3D extensions of HEVC	34
2.4.5 HEVC and its 3D extensions bitstream syntax	35
3. PROPOSED METHOD (PARTIAL DECODING).....	38
3.1 Partial decoding implementation possibilities.....	38
3.2 Enabling the self-contained slices in the AVC standard	39
3.2.1 General case	39
3.2.2 Self-contained Slices	41
3.3 Enabling the self-contained tiles in the HEVC standard.....	42
3.3.1 General case	42
3.3.2 Self-contained tiles in conventional HEVC	44
3.3.3 Self-contained tiles in 3D extensions of HEVC.....	46
3.4 Slice/tile-based extractor	46
4. RESULTS AND PERFORMANCE EVALUATION	50
4.1 Experimental setting.....	50
4.2 Test sequences.....	50
4.3 Partitioning arrangement	51
4.4 Results analysis	52
4.4.1 Compression performance	52

4.4.2	Complexity and latency reduction	55
5.	CONCLUSIONS.....	57
6.	REFERENCES.....	58
	APPENDIX A: H.264 TRACING PATH OVERVIEW	63
	APPENDIX B: DOWNLOAD LINKS	64

LIST OF FIGURES

Figure 1.	<i>Holografika multi-view 3D display [3]</i>	<i>12</i>
Figure 2.	<i>YUV 4:2:0 video data structure.....</i>	<i>16</i>
Figure 3.	<i>Block diagram of a video encoder.....</i>	<i>18</i>
Figure 4.	<i>Chronology of international video coding standards [25].....</i>	<i>19</i>
Figure 5.	<i>Intra- and inter-prediction modes and motion vector concept.....</i>	<i>22</i>
Figure 6.	<i>Hierarchical organization of the AVC/H.264 bitstream</i>	<i>26</i>
Figure 7.	<i>HEVC coding tree: a) CTU partitioning, b) Corresponding tree</i>	<i>27</i>
Figure 8.	<i>PU partitioning modes</i>	<i>28</i>
Figure 9.	<i>CU types or coding schemes within a typical video frame [34].....</i>	<i>28</i>
Figure 10.	<i>Tile-based and raster scan order of CTUs within a frame.....</i>	<i>34</i>
Figure 11.	<i>Overview of the system structure for 3D video transmission [32].....</i>	<i>35</i>
Figure 12.	<i>HEVC bitstream structure</i>	<i>36</i>
Figure 13.	<i>3D-HEVC access unit structure</i>	<i>36</i>
Figure 14.	<i>Two different slice arrangements in H.264, drawn with [41].....</i>	<i>39</i>
Figure 15.	<i>Left: A correctly decoded frame, right: A decoded frame of a bitstream in which every other slices were dropped</i>	<i>40</i>
Figure 16.	<i>Standard slices in H.264 [34]</i>	<i>41</i>
Figure 17.	<i>Tile partitioning, Balloons sequence with resolution 1024x768 [41].....</i>	<i>43</i>
Figure 18.	<i>Spatio-temporal positions of the merge and AMVP candidates [32].....</i>	<i>45</i>
Figure 19.	<i>Bottom-most PU within a tile</i>	<i>45</i>
Figure 20.	<i>Delimiters in a typical bitstream.....</i>	<i>47</i>
Figure 21.	<i>Frame partitioning and the extracted partitions.....</i>	<i>51</i>
Figure 22.	<i>BD-rate over number of tiles.....</i>	<i>54</i>
Figure 23.	<i>Decoding speed in AVC (frame per sec.)</i>	<i>56</i>
Figure 24.	<i>Decoding speed in HEVC (frame per sec.)</i>	<i>56</i>

LIST OF TABLES

Table 1.	<i>A typical super-multi-view video data specification</i>	<i>18</i>
Table 2.	<i>Configuration parameters for two different tiling configuration approaches</i>	<i>42</i>
Table 3.	<i>Configuration parameters to obtain the highest possible number of tiles</i>	<i>43</i>
Table 4.	<i>The required information for parsing tiles from bitstream</i>	<i>48</i>
Table 5.	<i>Test sequences description</i>	<i>50</i>
Table 6.	<i>Storage performance comparison between Intra profile and Baseline, and Random Access Main profiles in AVC and HEVC, respectively (delta-bitrate %).....</i>	<i>52</i>
Table 7.	<i>AVC slice overhead (BD-rate %).....</i>	<i>52</i>
Table 8.	<i>HEVC slice and tile overhead (BD-rate %).....</i>	<i>53</i>
Table 9.	<i>HEVC tile overhead for different vertical partitioning schemes (BD-rate %).....</i>	<i>54</i>
Table 10.	<i>HEVC tile and slice overhead for different vertical partitioning schemes (BD-rate %).....</i>	<i>54</i>
Table 11.	<i>HEVC tile overhead for different horizontal partitioning schemes (BD-rate %).....</i>	<i>54</i>
Table 12.	<i>HEVC tile and slice overhead for different horizontal partitioning schemes (BD-rate %).....</i>	<i>55</i>
Table 13.	<i>Decoding speed in AVC (frame per second)</i>	<i>55</i>
Table 14.	<i>Decoding speed in HEVC (frame per second)</i>	<i>56</i>

LIST OF SYMBOLS AND ABBREVIATIONS

2D	two-dimensional
3D	three-dimensional
AMVP	advanced motion vector prediction
ASO	Arbitrary Slice Order
AVC	Advanced Video Coding
BD-rate	Bjøntegaard Delta-rate
CB	coding block
CTB	coding tree block
CTU	coding tree unit
CU	coding unit
DIBR	depth-image-based rendering
FMO	Flexible Macroblock Ordering
FOV	field of view
FRExt	Fidelity Range Extension
FTV	free-viewpoint television
HEVC	High Efficiency Video Coding
HTM	HEVC Test Model
HVS	human visual system
JM	Joint Model
LF	light-field
MB	macroblock
MCP	motion-compensated prediction
MPEG	Moving Picture Experts Group
MTU	maximum transmission unit
MV	multi-view
MVC	Multiview Video Coding
MVD	motion vector difference
MVD	Multiview Video plus Depth
MVP	motion vector predictor
NAL	Network Abstraction Layer
PB	prediction block
PPS	Picture Parameter Set
PSNR	peak signal-to-noise ratio
PU	prediction unit
RAP	random access point
RDO	rate-distortion optimized
RGB	red, green, and blue
ROI	Region-of-interest
SCC	screen content coding
SEI	Supplemental Enhancement Information
SPS	Sequence Parameter Set
SVC	Scalable Video Coding
TU	transform unit
VCEG	Video Coding Experts Group
VCL	Video Coding Layer
VPS	Video Parameter Set
WPP	wavefront parallel processing

1. INTRODUCTION

In many video applications, an important feature for users is to be able to jump to specific parts of the video (e.g., in personal video recorder: forward/backward playback function) and to switch between different channels (e.g., in TV broadcasting: changing channels at any time) with minimum delay. This feature is referred to as random access to bitstream (i.e., coded video), where decoder is capable to start decoding a bitstream at a point other than the beginning of the bitstream and reconstruct an exact or approximate representation of the source video. This capability must be also supported at the encoder side, where at some points, known as random access points (RAP), the video pictures (i.e., coded frames) are coded such that all dependencies of the pictures following the RAPs, inclusive, to the pictures prior to RAPs are removed.

In the abovementioned applications, random access to video pictures is required. However, in some video applications random access to a portion of a picture is desired in addition to the picture itself, for example, in partial decoding in which only parts of the video pictures are presented in the decoder side. This is enabled by using picture partitioning tools which divide a picture into multiple partitions. In fact, picture partitioning tools facilitate efficient random access to a portion of a coded frame, which is necessary in partial decoding applications. In the recently emerged Advanced Video Coding (AVC)/H.264 and High Efficiency Video Coding (HEVC)/H.265 standards, the frame partitioning tools so-called slices (supported in both) and tiles (introduced in HEVC) are utilized to divide a video picture into multiple partitions.

The most straightforward alternative to the partial decoding solution is to crop the desired partitions from source videos, in the pixel domain, and encode them separately. This approach requires an extra post-processing step and a large number of bitstreams to be handled synchronously. Hence, the spatial approach is not desirable in this work.

In order to realize partial decoding of a video picture, the constructed partitions have to be self-contained (i.e., independently decodable). This kind of partitions is coded/decoded in such way that any kind of dependency on other partitions within the current picture and also non-co-located partitions in the reference pictures is prohibited. This way, only necessary partitions within a video picture can be decoded while skipping the remaining partitions. Despite of support for picture partitioning tools, partial decoding of video pictures is missed from the recent video coding standards.

In general, in a block-based hybrid video coding standard (e.g., AVC/H.264 and HEVC/H.265) the independently decodable property is extended to all three coding components of such standard including predictive, residual, and entropy coding [1]. In terms of residual and entropy

coding, the standard slices/tiles (i.e., what is introduced by the standards) are independent of each other since entropy coding engine is reinitialize at the beginning of each slice/tile. However, the standard slices/tiles may not be independently decodable in terms of predictive coding. While standard slices/tiles restrict intra-picture prediction within slice/tile boundaries, in inter-picture prediction slice/tile boundaries are disregarded in the motion compensation process.

This means some extra requirements have to be imposed on the standard partitions in order to be self-contained and thus applicable in the context of partial decoding. To fulfill these requirements, the motion compensation process has been modified in order to restrict inter-picture prediction to within co-located slices/tiles. In addition, regular partitioning as another requirement for enabling partial decoding must be fulfilled. Regular partitioning refers to partitioning the video pictures into the same number of partitions such that each partition occupies the same spatial region over all frames. This is achieved by configuring partitioning arguments appropriately.

The abovementioned requirements are related to the encoder side. The partial decoding must be supported in the decoder side as well. Enabling partial decoding can be conducted either by extracting and transmitting the coded elements corresponding to the necessary partitions, or transmitting the whole bitstream as it is. In the latter case, a dedicated decoder which is capable to skip decoding of the unnecessary partitions is needed. In contrast, in the former case partial decoding can be achieved with a confirming decoder, however, some bitstream manipulations are required in order to make the corrupt bitstream (i.e., containing only necessary partitions) standard compliant. Moreover, the bandwidth required to transmit a bitstream that is disposed of un-necessary partitions will be much lower. As the coded partitions may be interleaved with other coded data in the bitstream and as parameter sets and headers (e.g., slice segment header) are for the entire bitstream, a dedicated decoding process is defined for decoding particular partitions, while dropping the decoding of other partitions.

The main contribution of this research is to enable partial decoding of video pictures in the AVC/H.264 and HEVC/H.265 standards. In this study, the problem of creating self-contained partitions are studied and requirements that need to be fulfilled are described. An encoder-side solution is proposed in order to build self-contained partitions. An extractor is designed to construct a full-picture-compliant bitstream corresponding to the desired partitions such that a standard decoder can cope that. Further, the advantages of self-contained partitions in the context of a light-field (LF) rendering application called projection-based LF 3D displays are examined.

1.1 Case study: Light-field 3D displays

In this section, a big picture over 3D display technologies is provided. The projection-based multi-view/LF 3D displays, especially the one introduced by Holografika [2], which is considered as the case study in this work, are then described briefly. However, in-depth

discussions of these broad topics are beyond the scope of the present work. The reader is referred to [3] for an overview of the state-of-the-art 3D display technologies. Finally, the advantages of partial decoding in the projection-based LF 3D displays are discussed.

The 3D display technologies are categorized into two fundamentally different categories, namely, binocular stereo and autostereoscopic technologies. The first one relies upon special eyeglasses worn by viewers, while the second type of 3D displays is glasses free, in which viewers can preserve a 3D sensation via their naked eyes. The autostereoscopic 3D display technologies are classified into three broad categories: 1) multi-view 3D display, 2) volumetric 3D display, and 3) digital hologram display. In each category there exist several 3D display mechanisms. The projection-based multi-view 3D displays fall in the multi-view 3D display category, which is based on an approximation of light field of the captured scene.

Nowadays, we observe emerging trends and utilizations of LF technology in a wide range of unconventional applications such as vision-correcting displays [4] and 3D displays [5] [6]. LF displays are expected to be the future of 3D displays. In contrast to typical auto/stereoscopic 3D displays, the new technology is believed to successfully convince the consumers who have not widely adopted 3D television. The LF 3D displays provide very high 3D quality experience through: offering significantly higher resolution and brightness, providing smoother motion parallax, wider field of view (FOV), and larger depth range [5], when compared with typical auto/stereoscopic 3D displays. There are, however, several inherent problems thus 3D image resolution and quality still are not comparable with that of high-end 2D displays.

Holografika introduces an approach to the 3D displaying based on the LF technology, known as HoloVizio technology described in [6]. The HoloVizio technology, see Figure 1, uses a specially arranged array of optical modules/projectors and a holographic screen. Each point of the holographic screen emits light beams of different color and intensity to the various directions. The light beams generated in the optical modules hit the screen points in various angles and the holographic screen makes the necessary optical transformation to compose these beams into a perfectly continuous 3D view. With proper software control, light beams leaving the pixels propagate in multiple directions, as if they were emitted from the points of 3D objects at fixed spatial locations. Contribution of the projections are proportional to their FOVs with respect to the display screen. This means the projections towards the edges of the setup contribute with a smaller number of rays due to the finite size of the screen [7].

In the projection-based multi-view/LF 3D displays, the abovementioned achievements require employing a high number of parallel views (i.e., a collection of video sequences showing a 3D scene from slightly different perspectives). The number of views can be up to hundreds in order to fully exploit the visualization capabilities and viewing angle offered by such displays. Delivering, processing and de/compressing this amount of views pose big technical challenges. Need of handling such large amount of data prevents this kind of 3D displays to operate in real-time. Hence, any technique which leads to processing of such amount of data effectively and efficiently is desired, without scarifying the promised realistic representing of 3D scenes.

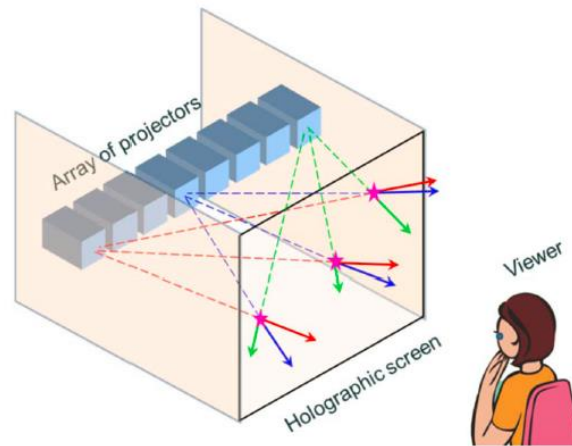


Figure 1. Holografika multi-view 3D display [3]

Even light-field 3D displays which are driven by a distributed computing system cannot be served by a single computer, because of the high pixel count. In addition, due to the nature of light-field rendering, different computing elements require access to separate parts of the encoded views, therefore random access into the bitstream is desirable. It is shown in [8] that when processing light fields in a distributed system, access patterns in ray space are quite regular. This means some processing nodes do not need all views. Moreover, the necessary views are used only partially. As a conventional approach, decoding the bitstreams entirely might be prohibitive due to high number of video streams and / or high resolution.

The above trait can be exploited to save decoding time and reduce display bandwidth, thus enabling real-time operation. While skipping the decoding of whole views is straightforward, partial decoding of video pictures is not supported by the recently emerged video coding standards. Enabling this functionality is instrumental for feeding LF displays with live imagery. In view of this need, this work is motivated with enabling partial decoding of video pictures to enable real-time operation.

1.2 Related work

In the modern video coding standards, different types of predictions are employed to exploit redundancies among video frames and thus to improve compression efficiency. However, predictive coding together with introduction of multiple reference frames brings complexity to other aspects of video coding, for example, in applications where random access to bitstream is necessary. Some tools which limit the aforementioned dependencies are video segmenting (e.g., slices and tiles), control over reference frames list selection, and restriction on the value of motion vectors. Some applications in which eliminating or removing the prediction dependency is advantageous include: parallel coding, error resilience coding, and partial decoding. This work aims at realizing partial decoding using independently decodable partitions in the video pictures.

In video coding community, independently decodable partitions within a video frame are often referred as isolated [9] [10], self-contained [11] [12], motion-constrained [13] [14], or simply independent [15] [16] partitions. The earliest independently decodable partition concept in video coding standards was introduced by independent segment decoding mode of H.263 in Annex R. When the mode is enabled, slice boundaries are treated as frame boundaries, where both intra-picture and inter-picture predictions are restricted within slice boundaries [9].

Self-contained partitions have been utilized for different applications and served various purposes in the literature. A review of some of these applications is provided as follows.

- Region-of-interest (ROI) applications, where only ROI needs to be decoded, for example for enabling stream-reassembling in video conferencing [14] [17].
- Real-time content editing in compressed domain, where video content needs to be modified partially [15].
- Scalable distribution of viewports on the same video sequence to a large amount of devices with different specifications and physical properties [11].
- Viewport-adaptive video streaming in virtual reality applications, where the viewer's current viewport is transmitted at higher quality at a time [13].
- Providing interactive on-demand service, for example, dynamic pan/high-quality zoom into a portion of the video (e.g., ClassX e-learning system of Stanford University [18]).
- Smart surveillance, security and inspection applications, for example, in drone video communication partial decoding helps in providing a faster data transmission.

A naïve alternative solution to the self-contained partitions is presented in [19], which is based on segmenting the video frames in the pixel domain and encoding each segment as a separate sequence. In the end-user device, all transmitted segments are decoded and stitched in order to generate the final viewport. In this approach, a large number of bitstreams should be synchronously transmitted. In contrast, using the proposed self-contained slices/tiles, a single bitstream can be send for each viewport, which requires a single decoder instance in the end-user device.

In [11] and [15] the authors utilize AVC/H.264 slice partitioning feature to avoid decoding and re-encoding of transmitted video sequence in the end-user device, in order to enable real-time operation. Decoding and re-encoding of video sequences, which refers to as transcoding, can be a complex task and also introduces some quality degradation, especially in case of high-resolution video sequences. In [15], the authors focused on real-time editing of videos, where region replacement is performed in the compressed domain. The proposed approach splits each video frame into several slice groups using Flexible Macroblock Ordering (FMO) tool which enables flexible slices (i.e., slices that can contain any predefined block locations). Slices within a slice group are made independent of other slice groups. The presented results indicate that editing video content by means of independent slices outperforms transcoding in terms of compression efficiency and complexity reduction. The proposed encoder restrictions to build

H.264 self-contained slices are similar to those described in [15]. Contrary to [15], in this work only slice is used and no slice header re-encoding is performed.

The authors in [11] benefit from self-contained slices to distribute viewports on the same video sequence to a number of devices with different specifications (e.g., processing power and bandwidth capacity) and physical properties (e.g., display size). In this approach, the desired slices relative to each user are cropped out from the segmented video bitstream. Afterwards, the bitstream is re-structured by replacing the dropped slices with artificially generated slices of minimal size. The resultant bitstream is standard compliant with lower bitrate, when compared with the original bitstream. In contrast, in the H.264 part of this work, no bitstream manipulation is performed. The unnecessary slices are dropped from the original full-picture bitstream using a designed extractor and the necessary slices are arranged in a sub-bitstream which can be coped using the FFmpeg x264 decoder [20]. This means the FFmpeg decoder allows decoding specific self-contained slices, regardless of the fact that other slices in the frame have been dropped.

The works presented in [10] and [16] use AVC slices and HEVC tiles, respectively, to enable ROI coding, where only ROI regions are restricted. In this work, the restriction is imposed to all slices/tiles within a frame. This provides a full degree of flexibility in the decoder side by selectively decoding any slice/tile of interest. However, it drops compression efficiency compared to the case where the restriction is imposed only to the necessary partitions.

Another study in [14] utilizes HEVC tiles in order to enable bitstream reassembling operation in video conferencing. In This work, similar encoder restrictions as proposed in [14] and [21] are used. The proposed method in this work uses the standard HEVC decoding, unlike [14], which requires modifications to the candidate list derivation for HEVC motion vector prediction.

The work presented in [13] uses HEVC motion-constrained tiles to enable adaptive streaming of panoramic videos for virtual reality applications. Authors in [21] and [17] use motion-constrained tiles to stich multiple video streams in the compressed domain in video conferencing and video surveillance applications. This work aims to benefit from self-contained slices/tiles in LF 3D displays.

To my knowledge this is the first research involving enabling partial decoding in MV and 3D extensions of HEVC.

1.3 Contribution and publication

The contributions of this work can be summarized as follows:

- A full instruction on enabling slicing and tiling in the AVC and HEVC standards, respectively, with any arbitrary arrangement is presented.

- The requirements for building self-contained partitions that are needed to enable partial decoding of video pictures are formulated.
- JM AVC/H.264 reference software version 18.6 was modified to enable partial decoding using slices.
- HEVC HTM reference software version 14.0 was modified to enable partial decoding in conventional HEVC/H.265 and its multi-view and 3D extensions using tiles.
- The effect of the introduced self-contained partitions on compression efficiency and decoding complexity is evaluated in terms of rate-distortion curve and decoding speed, respectively.

The following publications resulted from the work conducted during this thesis:

- Péter Tamás Kovács, **Alireza Zare**, Tibor Balogh, Robert Bregović, and Atanas Gotchev, “Architectures and codecs for real-time light field streaming,” *Journal of Imaging Science and Technology (JIST)*, Dec. 2016.
- **Alireza Zare**, Péter Tamás Kovács, Alireza Aminlou, Miska M. Hannuksela, and Atanas Gotchev, “Decoding complexity reduction in projection-based light-field 3D displays using self-contained HEVC tiles,” *3DTV-Conference 2016*, July, 2016, Hamburg, Germany.
- **Alireza Zare**, P.T. Kovacs, and Atanas Gotchev, “Self-contained slices in H.264 for partial video decoding targeting 3D light-field displays,” *3DTV-conference 2015*, July, 2015, Lisbon, Portugal.
- P.T. Kovacs, A. Fekete, K. Lackner, V.K. Adhikarla, **A. Zare**, T. Balogh, “Big Buck Bunny light-field test sequences,” provided by Holografika for MPEG FTV AHG, (c) copyright 2008, Blender Foundation / www.bigbuckbunny.org, Feb., 2015, Geneva, Switzerland.

1.4 Thesis structure

The rest of the thesis is organized as follows. In Section 2, the related background concepts are explained. In this section, an introductory part of video coding is followed by a definition of the emerging video coding standards. Further, an overview of the related aspects of the H.264 and H.265 standards are presented. The proposed method for building self-contained partitions in both standards is explained in Section 3. The experimental condition and test video sequences are described in Section 4. The section continues with evaluation of the proposed method and analyzing the results. Finally, the thesis is concluded in Sections 5.

2. BACKGROUND

This section presents the related background information. An introduction to the video coding topic is followed by a brief definition of the emerging video coding standards. Further, an overview over the relevant aspects of the AVC/H.264 and HEVC/H.265 standards is provided.

2.1 Basis of video compression

Video compression is the process of squeezing or compacting video data into a format suitable for transmission or storage. Compression system composes of a complementary pair of components so-called encoder and decoder. Encoder converts video data into a compressed form (i.e., bitstream) and decoder reconstructs a representation of the source video data from the encoder output with a specific fidelity. The encoder/decoder pair is often referred as codec.

Video compression relies on two main strategies: removing irrelevant information (i.e., information that not perceived by human visual system) and redundant information (i.e., information that is repeated) from source video data. Irrelevancy reduction can be achieved by introducing a model for representing video data, which is compatible with human visual system (HVS) characteristics. The HVS is more sensitive to brightness (luminance) than color (chrominance). In order to benefit from this fact, the luminance information can be separated from the color information which can be then represented with a lower resolution [22]. For this purpose, video data in the RGB color space is generally converted to the YUV space.

Therefore, the source video data is generally given in YUV with 4:2:0 color subsampling format, in which chrominance components have half resolution vertically and horizontally relative to luminance. Note that this is already a simple compression. Furthermore, this allows encoder to process luminance and chrominance components separately. The structure of video data in YUV 4:2:0 format is shown in Figure 2. For example, a video frame with 1280x768 resolution and 8 bits pixel depth in the RGB requires $3 * (1280 * 768)$ bytes, ≈ 2.949 MB space while the same frame in YUV 4:2:0 space is stored by $(1280 * 768) + 0.25 * (1280 * 768) + 0.25 * (1280 * 768)$ bytes ≈ 1.475 MB which is half as much space as occupied in the former case.

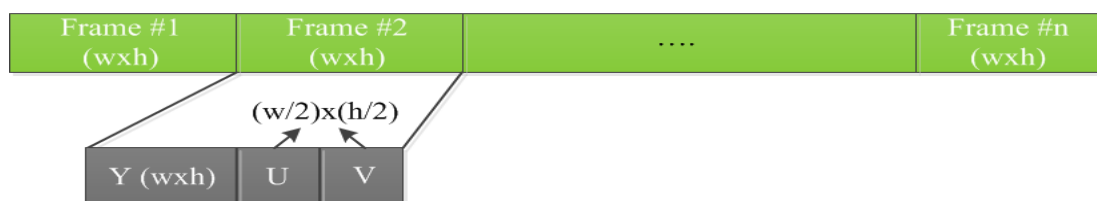


Figure 2. YUV 4:2:0 video data structure

Redundancy reduction aims to remove information that is repeated in the source video data and not necessary for faithful reproduction of the data. The modern video compression standards exploit four different kinds of redundancies called spatial, temporal, statistical, and frequency to achieve compression. In the spatial domain (i.e., within a video frame) the neighboring pixels are correlated (i.e., similar to each other). There is usually a high correlation among successive video frames in time order, which is referred as to temporal redundancy. The statistical redundancy is exploited by so-called entropy encoder which takes advantage of patterns in the data and represents more frequently occurring data more efficiently.

Transferring video data (e.g., a frame) into frequency domain reveals that many of the coefficients in the higher frequencies, which correspond to details within the frame, are close to zero. It is possible to achieve compression by removing these insignificant coefficients. This is referred as to removing frequency redundancy which benefits from HVS behavior as it is more sensitive to lower frequencies.

These redundancies are exploited by means of coding techniques such as intra prediction (i.e., exploiting spatial redundancy), motion-compensated prediction (i.e., exploiting temporal redundancy), entropy coding (i.e., exploiting statistical redundancy), de-correlating linear transform and scalar quantization (i.e., exploiting frequency redundancy) [22].

The modern video compression standards operate based on block-based hybrid video coding principle. Following this principle, each video frame is partitioned into blocks of samples. Each block is then coded by means of a coding technique which consists of a combination of prediction techniques, transformation of the prediction error and entropy coding. This is known as hybrid video coding technique. In the following the main functional units of a hybrid video encoder are described.

A general video encoder block diagram is shown in Figure 3. The prediction model receives uncompressed video data as input and attempts to reduce spatial and temporal redundancies by predicting the current block video data from neighboring video frames and/or neighboring samples within the same frame. It means the prediction is constructed by intra-picture and inter-picture predictions. The prediction model outputs a residual frame which is created by subtracting the prediction from the actual current block. The spatial model receives the residual frame and aims to de-correlate the residual samples further. This is typically carried out by applying a transform to the residual samples and quantizing the transform coefficients. As a result of this process, the residual samples in the transform domain can be represented with a small number of significant coefficients. The prediction model and spatial model parameters are represented in a more compact form by removing statistical redundancy using entropy encoder. The output bitstream consists of coded prediction parameters, coded residual coefficients and header information.

At the end of the section, the essence of video coding for both storage and transmission of a typical super-multi-view video data utilized in a 3D projection-based LF display is described.

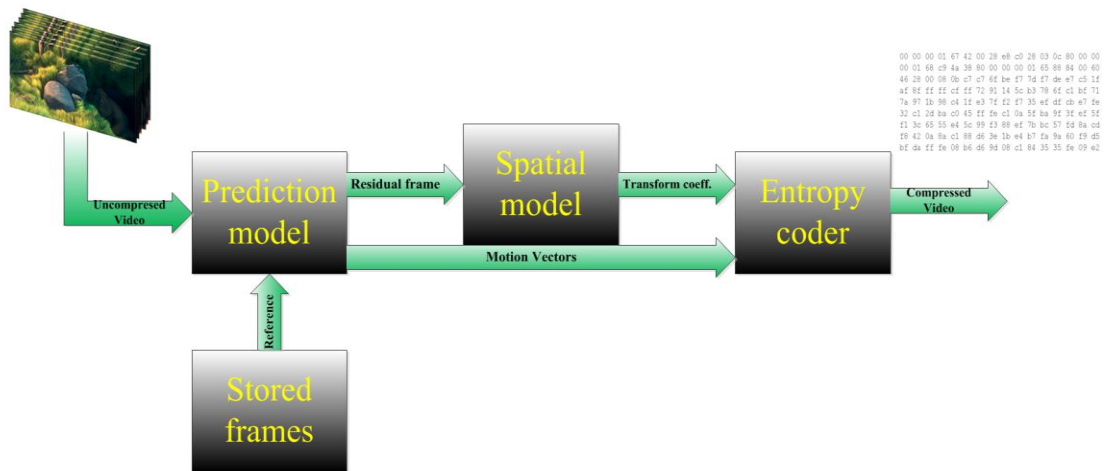


Figure 3. Block diagram of a video encoder

Table 1. A typical super-multi-view video data specification

Number of views	91
Number of frames	123
Frame rate	25 fps
Resolution	1280x768
Pixel depth	8 bits/component

Consider a multi-view sequence with the specifications described in Table 1. The required bandwidth for transmitting each view equals:

$$\left((1280 * 768 * 8) + 2 * (640 * 384 * 8) \right) * 25 \approx 281 \text{ Mb/s}$$

Based on Akamai 2015 ranking, average connection speed in Finland is 16.6 Mb/s. It means transmitting 1 sec of each original video takes ≈ 17 seconds. The required memory space for storing one minute of each view equals:

$$60 * 25 * \left((1280 * 768 * 8) + 2 * (640 * 38 * 8) \right) \approx 17.7 \text{ Gb}$$

It is clearly understandable the incredible demand on network and storage infrastructures without compressing, especially for real-time applications.

2.2 Video compression standards

A video compression standard defines the compressed format (i.e., bitstream syntax) and decoding process. The encoding process is not included in the standard in order to allow everyone/every manufacture having its own implementation. The only requirement is that the bitstream must be decodable with a standard confirming decoder.

For the compression standards, there are two main international organizations namely ITU-T and ISO/IEC. From ITU-T, the Video Coding Experts Group (VCEG) of Question 6 (Visual coding) of Working Party 3 (Media Coding) of Study Group 16 (Multimedia Coding, Systems and Applications) and from ISO/IEC, Moving Picture Experts Group (MPEG) of Joint Technical Committee Number 1 of Subcommittee 29 of Working Group 11 have dominated video compression standardization [23]. Figure 4 illustrates chronology of international video coding standards and contribution of these two organizations in developing them.

The H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) standard and its successor H.265 or High Efficiency Video Coding (HEVC) are currently the most commonly used standards in video compression. The goals of these standardization efforts were improving compression efficiency up to 50% and providing more network-friendly design compared to its previous standards, without sacrificing quality. The latest collaboration between the two standard bodies has resulted the HEVC/ H.265 standard. HEVC aims to support high resolution video contents up to 8K (8192x4320) and to improve parallel processing architecture. In addition, the new standard is intended to be applicable over a wide range of applications with respect to the need for interoperability of products built by different manufactures and potentially customized by different vendors [24]. HEVC is built on the concepts of the AVC standard. It inherits a number of features from AVC. Some features that exist in AVC were not included in HEVC and a number of new features are introduced in HEVC, which are not available to that of older standard. Both HEVC and AVC standards are designed based on block-based hybrid video coding principle.

Over the time, the capabilities of the original standard are extended by adding new coding features to support broader range of applications. While the high level syntax designs of AVC extensions are not fully aligned, HEVC extensions are designed as a high-level syntax only

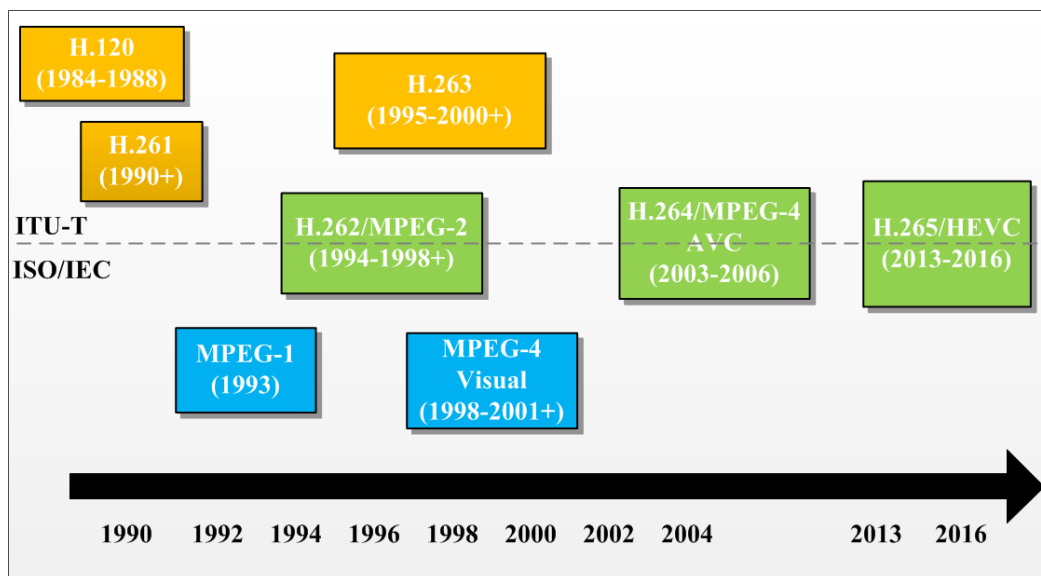


Figure 4. Chronology of international video coding standards [25]

extension to allow reuse of existing components. The extensions can be classified into two categories, based on the generated bitstream, as follows: single-layer extensions and multi-layer extensions. In the single-layer extensions, like in the original standards, the generated bitstream exhibits a single representation of the coded video. In contrast, the generated bitstream in the multi-layer extensions contains different representations of the coded video in terms of spatial resolution, temporal frame rate, quality, and view angle. In addition, a layer may correspond to the property of being texture or depth data in 3D video coding. In a multi-layer bitstream, a layer known as base/independent layer is followed with one or more enhancement/dependent layers. Regarding the abovementioned classification, there is an exception in case of HEVC, wherein temporal sub-layers corresponding to different temporal frame rates are embedded within a single layer. The development of the HEVC extensions can be traced from [26]. In the following, the existent extensions of both standards are briefly introduced. The irrelevant extensions to this work are also introduced for the sake of completeness.

Scalability extension (support for variety of video transmission systems with different capabilities: AVC Scalable Video Coding (SVC) and HEVC Scalability (SHVC) extensions): Scalable video coding aims to deliver multiple coded version of a video using a lower overall bitrate relative to simulcast coding (i.e., independent coding of each version). The term scalability refers to the removal of parts of the bitstream in order to adopt it to the various needs (e.g., network conditions and end user capabilities) in the application under consideration. An overview of the scalability extensions of AVC and HEVC are provided in [27] and [28], respectively.

Fidelity/format range extension (support for higher-fidelity video materials: AVC Fidelity Range Extension (FRExt) and HEVC Format Range Extension (RExt)): These extensions provide tools for color format and bit depth enhancement. They make a preferable choice for high-quality applications by introducing tools to support sample bit depths beyond 10 bits/sample and different chroma sampling structures, 4:0:0, 4:2:2 and 4:0:0, in addition to the common 4:2:0 chroma sampling format. In addition to support for extended sample bit depth and chroma format, these extensions contain other important tools to address the higher-quality demands. The reader is referred to [29] and [30] for detailed description of the FRExt and RExt, respectively.

3D extension (support for 3D video applications: AVC Multiview Video Coding (MVC), MV-HEVC and 3D-HEVC extensions): These extensions provide tools to improve coding of multi-view video and 3D (i.e., multi-view video plus depth) data captured by multiple cameras from the same scene. Examples of applications where this type of video data is employed include 3D displays (e.g., stereoscopic, auto-stereoscopic and LF displays) and free-viewpoint televisions (FTVs) in order to enable depth perception and interactive selection viewpoint and direction, respectively. A typical multi-view video contains a large amount of inter-view similarities in addition to the previously introduced correlations among frames of a single video data. These extensions provide support for exploiting such correlation among the views. In these extensions, one of the views is known as independent/base view, which is coded independently from other views called independent view. This means only the dependent views

use additional coding features introduced in these extensions of the conventional standards. This further means the base view is fully compatible with the conventional codecs. This property provides compatibility with existing 2D video applications.

Both multi-view extensions in the AVC and HEVC use the same design principles. For detailed description of these extensions the reader is referred to [31] and [30], respectively. In contrast to MV-HEVC, 3D-HEVC uses depth information for coding of video views. However, it is not restricted to use depth information; it can also be used for a multi-view video data without depth maps. Furthermore, 3D-HEVC includes some new coding tools for efficiently coding of depth maps. In [30] and [32], the 3D and MV extensions of HEVC are described in detail. These two extensions of HEVC are discussed later in this section.

Screen content coding (SCC) extension (support for efficient screen content coding) [26]: This extension includes tools to improve compression capability for video data that contains a mix of camera-captured natural video, computer-generated graphics and text. Example applications of this type of video data include wireless displays, control room video wall displays, screen desktop sharing and factory automation display.

2.3 Selected aspect of AVC/H.264

In this sub-section, a short introduction to H.264 coding structure and features that are relevant for this work is provided. In particular, frame partitioning tool, intra- and inter-picture predictions on the basis of block-based operation are discussed.

2.3.1 H.264 coding structure

The H.264 standard, similar to that of earlier standards, is known as a block-based video coding standard. Coding process is performed on the basis of 16x16-sample square regions within the source frame, called macroblocks (MB). A MB is considered the processing unit of H.264. A MB consists of one luminance 16x16-sample block and two chrominance 8x8-sample blocks, which are processed separately. This block-based partitioning allows applying different prediction models to different regions of a frame. Each block is coded by using either intra-picture or inter-picture prediction technique. The prediction techniques are described in the following sub-section.

A MB is named based on its coding type. MB types defined by H.264 include I, P, B, SI and SP. The two last types are skipped as they are not relevant to this topic. They facilitate efficient switching between video streams. An I MB is predicted using intra prediction from spatially neighbouring samples within a frame. P and B MBs are predicted by referring to samples in a past or future (in display order), previously coded frame(s). The only substantial difference between P and B MBs is that P MBs use exactly one reference picture but B MBs can be predicted from one or two reference picture (s). In other words, the predictor (i.e., prediction block) may be constructed from a single prediction region in a reference picture, for a P or B

MB, or from two prediction regions in reference pictures, for a B MB. In addition to the aforementioned MB coding types, there are two more coding modes, namely skip and direct modes which are special forms of the inter-coding mode. In the skip mode, no data about the MB (i.e., neither motion data nor residual) is coded, only a signalling bit (skip_bit) is sent to the decoder. The MB is decoded based on previously-coded data. For a direct-coded block, residual data is coded while motion data is derived from previously-coded neighbouring blocks. The direct mode is available for B MB only.

Supporting multiple block sizes helps to improve prediction efficiency. In H.264, a MB may be further partitioned to smaller blocks so-called MB partitions for the purpose of providing more effective prediction, since samples from two smaller blocks are more likely to be similar compared to the larger ones. Therefore, smaller prediction blocks are well suited for coding of parts of a frame with significant details. However they lead to increases in complexity, where more search operation and more signalling information to the decoder is required. Hence, there is always a trade-off between prediction efficiency and coding complexity.

2.3.2 H.264 intra and inter-picture prediction

The goal of the prediction techniques is to construct a prediction of the data and subtracting this prediction from the source data to lead to redundancy reduction. Intra prediction utilizes the fact that neighbouring samples within a frame are likely to be correlated. In contrast, inter prediction, also referred to as temporal prediction and motion-compensated prediction, tends to exploit temporal correlation. Hence, intra prediction is conducted without referring to neighbouring video frames, whereas in inter prediction the sources of prediction are prior decoded frames. The intra-prediction mode improves error resiliency and random access facility at the cost of lowering compression efficiency compared to the inter-prediction mode. Figure 5 illustrates concept of intra- and inter-prediction modes with respect to the MB types.

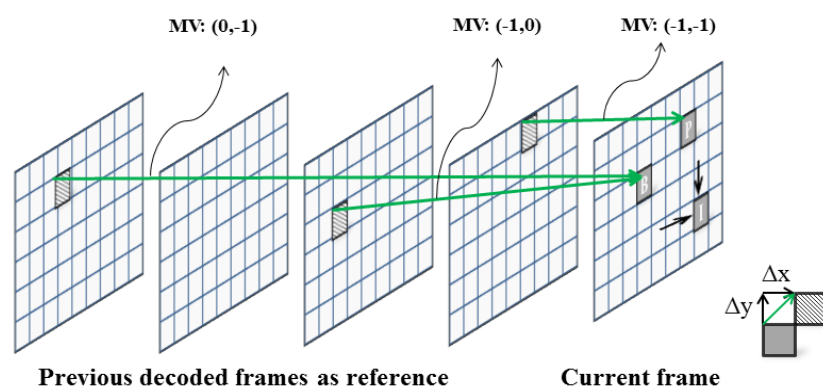


Figure 5. Intra- and inter-prediction modes and motion vector concept

In intra prediction, a block is predicted from previously-coded samples in the left and/or above adjacent blocks, as shown in Figure 5. Those samples are available at the encoding time of the current block by considering the fact that MBs are processed in raster scan order (i.e., from left to right and top to bottom) within a frame. The possible intra prediction block sizes for luminance samples are 4x4, 8x8 and 16x16, while the chroma 8x8-sample blocks are predicted without more splitting. Each block size is associated with some prediction modes which define how and in which direction the prediction is performed. A detailed description of prediction modes are presented in [22].

Inter prediction aims to compensate for motion among video frames. A practical and widely used technique is block-based motion compensation which intends to compensate for movement of rectangular blocks of the current frame. The technique is typically a two-stage process. First, a search is carried out within an area centred on the current block location in the reference frame to find a similar block to the current block. The search can be performed over all of the possible blocks or some of them within the search area. For each candidate block in the search area, a residual is formed by subtracting the candidate block from the current block. A candidate block with the minimal energy in the residual gives the best match. This process of finding the best match is known as motion estimation. As the second stage, so-called motion compensation, the candidate block that has been chosen to be as the best match is subtracted from the current block to form a residual. A translation vector (i.e., motion vector) is also associated to each candidate block. The motion vector is the spatial offset between the location of the current block and the predictor in the reference picture. To a block of type B two motion vectors may be assigned. The motion vector concept is depicted in Figure 5. The residual relative to the best block together with corresponding motion vector (s) is encoded and transmitted to decoder.

In inter prediction, each luminance 16x16-sample block may be split to two 8x16 partitions or two 16x8 partitions or four 8x8 partitions. An 8x8-sample block of luminance and chrominance components may further be broken to two 4x8, two 8x4 or four 4x4. Each partition can be predicted from an area of the same size from different reference picture (s) [8]. However, in the real world objects often move by a fractional number of pixel positions among frames, where the block-based motion compensation model is not efficient enough. In this case, the reference picture is interpolated to the sub-pixel position before searching for the best block match.

The H.264 standard utilizes a rate-distortion optimized (RDO) mode selection framework to find the best coding mode (e.g., prediction mode and MB mode) of a MB. It is relatively a sophisticated process for encoder to find an optimal coding mode for a MB in a rate-distortion sense. The reason is that an encoder has a huge potential space of coding options, including hundreds of possible combinations of prediction modes, partition sizes, reference pictures and motion vectors. This enforces the encoder to utilize a lower complex process relative to exhaustive mode selection process in which all possible coding modes are tested. The most

popular formulation of rate-distortion optimized mode selection algorithm is Lagrangian cost function that linearly combines bitrate cost and distortion cost as in equation (1).

$$J=D+\lambda R \quad (1)$$

Where, J is joint cost, D is distortion within a MB, R denotes bitrate, and λ is Lagrange multiplier which controls the trade-off between rate and distortion costs. For example, a larger λ value tends to minimize bitrate cost at the expense of a higher distortion cost. The mode selection of each MB is such that the joint cost is minimized.

2.3.3 H.264 frame partitioning feature

In video coding, the frame partitioning feature typically serves the following purposes:

- **Error robustness:** The frame is partitioned into smaller entities in order to enable decoder to re-synchronize the decoding process at the beginning of each partition, instead of each frame. This typically implies that coded partitions are self-contained in the sense that they contain all data which is required at the decoding time.
- **Parallel processing:** The partitions within a frame can be encoded and decoded in parallel, where the coding process is specified in multiple independent units. This enables real-time operation of a video codec in highly computationally demanding applications.
- **Efficient packetization:** The frame partitioning supports adaptation to the network constraint in video transmission applications by providing differently sized network packets with respect to the prevailing network throughput.
- **Low delay video transmission:** One use case of the frame partitioning is reduction in the end-to-end delay in low delay applications such as streaming, broadcasting and remote video, where partitions of video frames can be used as entry points for transmission and coding process.
- **Bandwidth/bitrate and/or decoding time reduction:** The frame partitioning can also enable partial decoding of video frames in order to reduce bitrate and/or decoding time of the coded video.

Similar to earlier standards, H.264 has ability to partition a video frame into regions so-called slices. Multiple MBs within a frame are aggregated to form a slice. The slice feature improves bitstream access with specific granularity by providing random access to the lower level than coded frames. Generally, coded frames are referred to as access units and slices as subsets of access units.

In the H.264 standard, a video frame can be constructed from several slices. An encoder can be configured to use only one slice per frame, in which case all MBs of the frame constitute a slice, or to divide the frame into an arbitrary number of slices, with various possible arrangements. Slices within a frame are processed in raster scan order. Similarly, MBs within

a slice 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 [22]. Slices are named according to the coding type (i.e., intra- and/or inter-picture predictions), namely, I slice, P slice, B slice, SP slice, and SI slice. The slice coding types are described as follows:

- I slice: all MBs of I slice are I MBs which are coded using intra-prediction mode.
- P slice: MBs of P slice can be either I MBs or P MBs which are coded using inter-prediction mode with at most one motion vector per block
- B slice: in addition to P MBs, B slice also may contain some MBs predicted using inter-prediction mode with two motion vectors. The two vectors may have the same temporal direction or may not.
- SP slice: facilitates efficient switching between different streams encoded at different bitrates
- SI slice: allows switching between different streams with no correlation using only intra prediction

In addition to the bitstream access improvement which facilitates partial decoding, slices add robustness to the bitstream in the presence of transmission errors. This is actually the main purpose of introducing slices. Slice structure coding improves transport efficiency and error resilience by means of providing small decodable chunks for transmission, 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 low and roughly constant. As mentioned above, this can be achieved by the second slice configuration approach.

Besides the advantages of slicing tool mentioned above, the increased number of slices reduces peak signal-to-noise ratio (PSNR) to bitrate ratio. This mainly occurs because of reduction in spatial correlation since intra-picture prediction is restricted within the slice boundaries. In addition, header information is appended to each coded slice, which arises some extra bitrate overhead. This overhead of slice header also contributes to the reduced rate-distortion performance.

2.3.4 H.264 bitstream syntax

The bitstream format in H.264 is represented in a clearly defined syntax. The bitstream is a sequence of access units divided into Network Abstraction Layer (NAL) units (i.e., packets containing coded H.264 data). The bitstream is arranged in a hierarchical structure with three NAL, slice and MB layers, as shown in Figure 6.

In the highest level, the sequence level, H.264 bitstream is seen as a series of NAL units. There are two conceptual types of NAL unit so-called Video Coding Layer (VCL) and non-VCL NAL unit. VCL NAL units contain coded video data and non-VCL NAL units include metadata

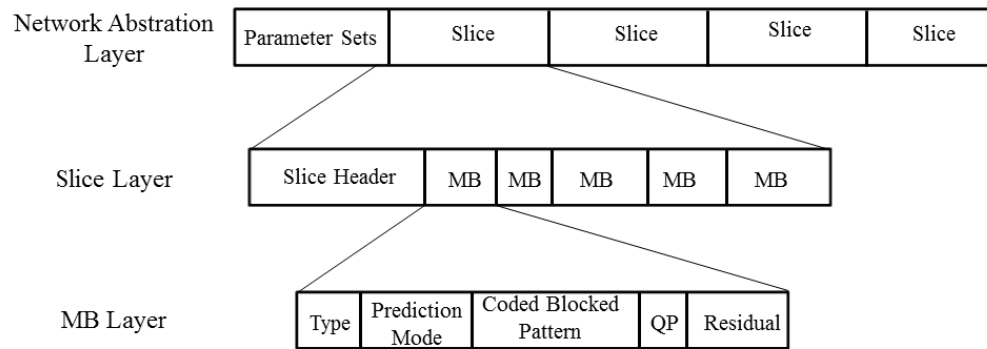


Figure 6. Hierarchical organization of the AVC/H.264 bitstream

such as Parameter Sets, Supplemental Enhancement Information (SEI) parameters, and delimiters. The bitstream starts with parameter sets which contain two NAL units, namely, Sequence Parameter Set (SPS) and Picture Parameter Set (PPS). The SPS and PPS contain common control parameters which are necessary for the decoder to correctly decode the bitstream. In the next layer, the slice layer, there are coded slices which are described as VCL NAL units. A VCL NAL unit consists of a slice header and slice data. The slice header section holds common information applicable to the all MBs in the slice such as the number of first MB in the slice, the slice type, and frame number that slice belongs to. For example, slice type “110” determines that each MB in the slice can be either of type I or P. The slice data consists of a series of coded MBs. The size of each encoded slice tends to vary depending on the number of MBs, 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-picture prediction tends to be less efficient than inter-picture prediction, followed by P slices and then B slices. In the lowest level of the hierarchical structure, a coded MB is represented, which contains all elements such as MB type and prediction mode necessary to decode a single MB which makes up a spatial block of a frame.

2.4 Selected aspect of HEVC/H.265

This section presents an overview of the HEVC coding structure. The newly introduced coding tree units (CTU), coding units (CU), prediction units (PU), and transform units (TU) block partitioning concepts are described. The coding features of HEVC and its 3D extensions relevant to this work are discussed, particularly prediction and frame partitioning tools. This is followed by a full instruction on enabling tiling with any arbitrary arrangement. Further, the multi-view and 3D extensions of HEVC are introduced. A description of HEVC and its 3D extensions bitstream structure is provided. The section further explains how tiles can be utilized to facilitate partial decoding of video frames.

2.4.1 HEVC coding structure

The HEVC standard adheres to block-based hybrid video coding paradigm. Along with this adherence, a number of innovative features are introduced and most parts known from previous standards are improved and generalized. Particularly, in HEVC frame partitioning and block partitioning concepts are considerably improved in comparison with of those in AVC/H.264. These improvements enable HEVC to be broadly flexible and optimized for various contents, applications and devices.

In the new standard, similar to the earlier standard, a video frame is coded using three separate color planes. Further, each plane is partitioned into squared-shaped regions. Each region is called coding tree unit (CTU) which represents the basis of coding process. This is analogous term to the macroblocks in AVC. A CTU consists of three components so-called coding tree blocks (CTB): one luma CTB and two associated chroma CTBs. In the YUV 4:2:0 format, each luma CTB covers a square area of $2^N * 2^N$ luma samples and each of the two corresponding chroma CTBs covers the area of $2^{N-1} * 2^{N-1}$ chroma samples. This results in the same number of CTBs in both luma and chroma components. The parameter N is chosen by the encoder among the values N=4, 5, and 6.

The CTU is further partitioned into multiple coding units (CU) based on a tree partitioning structure, namely quad-tree. This quad-tree is also referred to as coding tree. In the proposed coding tree structure, an internal node (i.e., a region) can be recursively divided exactly into four smaller regions of equal sizes. An example of the CTU partitioning is illustrated in Figure 7. In this example, a CTU of size 64x64 is split into 16 CUs with different sizes from 32x32 to 8x8. The same splitting is used for both chroma and luma CTBs. The numbers in Figure 7 (a) indicate the coding order of the CUs, which is a depth-first traversing in the tree structure drawn in Figure 7 (b). The CU composes three CBs, where a similar relation exists as for CTU and CTBs. The introduced quad-tree structure allows for splitting a CTB into wide range of differently sized sub-blocks. This is suitable for capturing versatile characteristics of a typical video. In a typical encoder setting, the maximum size of CU is equal to 64x64 (i.e., the CTU can be a single CU) and the minimum size of CU is equal to 8x8 (e.g., partition number 3 in

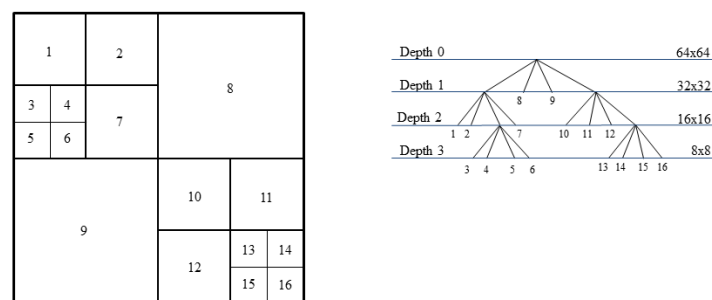


Figure 7. HEVC coding tree: a) CTU partitioning, b) Corresponding tree

Figure 7 (a)), and the minimum size of CTU is equal to 16×16 . In this case, the splitting can be started from depth 0 to depth 3.

For further partitioning, instead of splitting one depth more, prediction unit (PU) concept is introduced. The CU can be seen as the root for the PU splitting. The PU concept enables CU partitioning but differently from a general quad-tree structure. Unlike the CU partitioning, the PU may only be split once. The CU can be divided into one or two or four PUs according to the PU partitioning modes, as illustrated in Figure 8. HEVC supports two PU splitting modes for the intra coded CU: PART_2Nx2N, where a single PU is specified for each CU, and PART_NxN, where each CU is divided into 4 smaller PUs. For inter coded CU, in addition to the similar partitioning modes introduced in the intra coding, 6 more PU splitting modes are supported as shown in the second and third rows of Figure 8.

It is worth mentioning that the modes shown in Figure 8 are not supported for all CU sizes. For example, intra PART_NxN mode is only allowed when the CU size is equal to the minimum allowed CU size. More detailed information in regards to the PU concepts and partitioning modes can be found in the chapter 3 of [33].

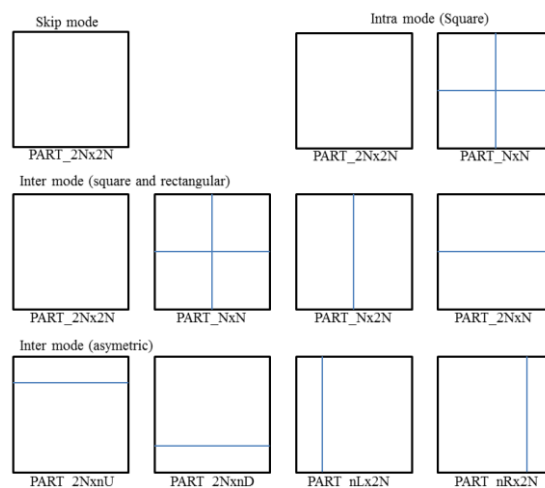


Figure 8. PU partitioning modes

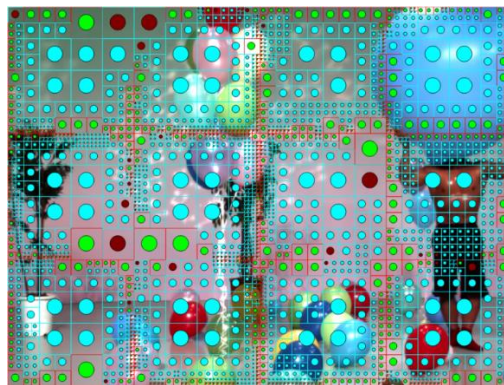


Figure 9. CU types or coding schemes within a typical video frame [34]

HEVC, similar to AVC, supports skip and direct modes coding. In HEVC, these two modes are improved and considered as special forms of newly introduced inter-prediction block merging technique. The technique is described in the following sub-section. The skip mode is well suited to code static regions in video data, where residual block tends to be close to zero. While in the skip mode no residual data is presented in bitstream, in the direct mode residual data is included in bitstream. In both modes, motion data for the current block is derived from motion data of spatial and temporal neighboring blocks. Figure 9 depicts CU types within a typical video frame. The Green, light blue and red colors indicate intra coded, skipped coded and inter coded CUs, respectively.

The AVC standard utilizes macroblock to specify prediction scheme (i.e., all macroblock partitions are coded with the same prediction scheme), whereas HEVC allows adaptation of prediction schemes for CUs within a CTU. This means that for each CU a prediction scheme (i.e., intra or inter) is specified by the encoder. Within a CU, a single set of motion parameters is signaled for each PU, which is used for motion-compensated prediction of the luma and chroma prediction blocks (PB). The PUs can further share motion information through a merge mechanism. The motion information is signaled after the final coding structure is formed.

To be more precise, a nested quad-tree partitioning mechanism is used in HEVC. This nested quad-tree structure exists since another quad-tree is rooted from the leaf nodes (i.e., from the CU regardless of the PU partitioning) of the aforementioned coding tree. The new quad-tree is called transform tree which recursively splits residual blocks into multiple units. The leaf nodes in the transform tree (i.e., transform units (TU)) form the basis of transform and quantization operations. An illustration of this nested quad-tree scheme is presented in Fig 3.7 from [33]. More detailed description of transform tree and associated concepts can be seen in [33].

The introduced block partitioning mechanism is one of the main sources of the coding efficiency improvements and the adaptability seen with HEVC. This provides a highly flexible and efficient block partitioning structure for supporting a wide range of applications from low resolution to high resolution and low motion to high motion contents. The size of CTU and maximum hierarchical depth can be configured in order to be optimized to the targeted content. For example, for contents that contain only slow motions, a CTU size of 64 and depth of 1 may be an appropriate choice. For more general contents which may also contain complex motions, a CTU size of 64 and maximum depth of 3 would be preferable. The usage of large block sizes together with small block sizes leads to a significant coding efficiency improvement. Larger block sizes lead to capturing the increased spatial correlation presented in a higher resolution video content. On the other hand, smaller block sizes provide adaptation of the block partitioning to the local properties of a video data. In consequence, encoder's freedom is increased by supporting a high number of possibilities for coding a video frame. However, it has to come along with increase in computational complexity, memory requirement and coding delay. Various aspects of the HEVC block partitioning were studied in [35].

2.4.2 HEVC intra- and inter-picture prediction schemes

The intra- and inter-prediction schemes have been demonstrated as the primary tools employed in current video coding standards. In HEVC, these schemes follow the basic ideas as in AVC but they are made far more elaborated and flexible such that the trade-off between coding efficiency and computational requirements is improved.

In HEVC, the AVC intra-prediction principle is further extended such that it is able to accurately model a wide range of textural and structural information in various types of content. HEVC supports 35 intra-prediction modes which can be classified in two groups. The first group is called angular intra-prediction mode containing 33 distinct modes numbered from 2 to 34 as illustrated in Fig.6 of [36]. These modes are intended to efficiently explore different structures with directional edges. In contrast, the modes in the second group, namely planar and DC modes numbered 0 and 1 respectively, provide predictors suitable for smooth regions. For detailed description of the intra-prediction modes, the reader is referred to chapter 4 from [33]. The Intra-prediction scheme, depending on the block size and prediction mode, may utilize pre-filtering of reference samples, prior to prediction process, and/or post-filtering of predicted samples in order to remove blocking and contouring artifacts. Contribution of individual intra prediction tools to the overall coding efficiency is discussed in detail in [37]. The enhancements of HEVC intra-prediction compared with that of AVC are described as follows:

- Large amount of different block sizes and wide range of prediction modes: The HEVC 35 distinct intra-prediction modes, compared with 9 modes in AVC, can be performed in different block sizes ranging from 4x4 to 64x64.
- The whole 35 prediction modes are available for both luma and chroma blocks, compared with limited number of intra-prediction modes for chroma blocks in AVC, where only 4 modes are allowed.
- Availability of the whole intra-prediction modes regardless of missing of the neighboring reference samples, using a reference sample substitution process. Under such a situation in AVC only DC mode is used.
- The abovementioned post-filtering stage is newly introduced by HEVC, which is not employed in AVC.

HEVC follows the general principles of the AVC inter-prediction scheme to drive a motion-compensated prediction (MCP) for a block of samples based on a translational motion model. Whereas the new inter-prediction scheme is not a revolutionary whole new design, all components are steady enhanced and additionally some new tools are introduced, when compared with that of AVC. Specifically, for exploiting correlation among motion data of neighboring blocks, HEVC introduces two new techniques so-called advanced motion vector prediction (AMVP) and inter-prediction block merging.

The AMVP technique is one of the most important tools introduced in HEVC, which significantly improves coding efficiency in terms of bitrate. It is based on the fact that motion vectors of spatial and temporal neighboring blocks are often correlated, since they are likely to belong to the same moving object with similar motion. In consequent, instead of directly encoding the motion vectors of the current block, a list of motion vector predictors (MVP) are constructed using the motion vectors of spatial and temporal neighboring blocks. Afterwards, a component wise motion vector difference (MVD) between the motion vector of the current block and the best MVP, in terms of rate-distortion optimization, from the list is formed. The best MVP is chosen from the list by a technique known as motion vector competition [38] which explicitly signals the index of the selected MVP for motion vector derivation. At the end, the MVD together with the best candidate index is signaled. In this way, the size of the signaled motion vector is reduced. Although the MVD concept is not new, already known from AVC, this kind of motion vector signaling is newly introduced by HEVC. In AVC, instead of having such list of candidate MVPs and explicitly signaling the best candidate index, MVPs are implicitly derived from already decoded motion vectors of neighboring blocks [24].

The HEVC inter-prediction block merging technique is initially proposed in [39]. The technique is introduced to address the inherent disadvantage of over-partitioning caused by the HEVC block partitioning structure, which leads to redundant signaling of motion parameters and consequently increase in bitrate. For example, if a given block is divided into 4 sub-blocks, all sub-blocks are separately coded even though they share the same motion parameters. This scheme forms a merged region sharing all motion information, where ineffective segmentation (i.e., dividing regions of equal motion parameters) is removed. Following this concept, a block reuses the exact same motion information of the neighboring blocks. Like AMVP, a list of candidate motion parameters is constructed from spatial and temporal neighboring blocks which are selected as merge candidates. Afterward, index of the candidate to be used for inferring the motion parameters of the current block is signaled. In contrast to AMVP list, which contains candidate motion vectors, the merge candidate list contains candidates of all motion information.

Consequently, there are two different ways for signaling of motion data in HEVC: it can be done using either the merge mode or explicitly encoded using the AMVP technique. In both AMVP and merge techniques, the filling of candidate lists are standardized and candidate order is fixed such that the signaled index would refer to the same candidate to the list at the encoding and decoding time. Chapter 5 from [24] describes in detail how the merge candidate and AMVP lists are constructed.

2.4.3 HEVC partitioning feature

Besides the slice partitioning feature, as already known from the previous video coding standard, HEVC improves frame partitioning concepts by introducing three new frame partitioning schemes. The novelties brought with HEVC are dependent slices, tiles, and

wavefront parallel processing (WPP). The novel tools mainly aim at facilitating bitstream access for error robustness, parallel processing, efficient packetization, and ultra-low delay (i.e., sub-frame delay) processing purposes. The introduced frame partitioning tools play a major role in the substantial performance gains which promised by the emerging standard. In the following, these partitioning tools are briefly introduced and their contributions for achieving the aforementioned goals are described. Meanwhile, the applicability of each tool for enabling partial decoding is discussed.

An integer number of CTUs is aggregated into a slice to form frame partitioning. The HEVC slice concept is conceptually equivalent to what we know as slice from the AVC standard. Additionally, in HEVC, the practical weaknesses of the traditional slices (i.e., low flexibility to the partitioning, bitrate overhead caused by slice headers, and strict break of intra-picture prediction) are addressed by adding novel functionalities to them. Similar to the AVC slices, the HEVC standard slices can be also utilized for enabling partial decoding. However, the same requirements to achieve self-contained slices in AVC must be fulfilled for constructing self-contained slices based on the HEVC standard slices.

Optionally, each slice can be fragmented into one or more slice segments at the CTU boundaries. The first slice segment of a slice, in raster scan order, is called independent slice segment. The subsequent slice segments, if any, are known as dependent slice segments. The independent and dependent slice segments differ by their headers. The dependent slice segment includes shorter segment header since it reuses information from the independent slice segment's header within the same slice. As another difference, in intra-picture prediction, dependent slice segment boundaries are disregarded. This prevents the usage of slice segments in partial decoding applications. However, the slice segment is an appropriate support for low delay applications (e.g., streaming, broadcasting, and remote video), adaptation of NAL unit sizes to the network constrains of maximum transmission unit (MTU) and parallelization, where slice segments may be used as entry points. In addition, slice segments improve error resiliency by limiting propagation of an error on the whole video frame, at least from a spatial perspective, without incurring substantial coding efficiency losses [24].

Each coded slice segment in turn can be divided into multiple subsets of coded CTUs, which are referred to as slice segment subsets or sub-streams. The idea of sub-stream concept is to divide coded slice data into several coded fragments, without the use of header data, instead of including each spatial partition in an individual NAL unit. This allows a high number of frame partitions/coded fragments such that they can be concurrently encoded/decoded (i.e., parallelization) without incurring extra header data. By using only slice segment subsets, it is not possible to associate each coded fragment to the related spatial region in the video frame. Hence, they are not practical in the partial decoding context. Slice segment subsets are used together with other partitioning tools (e.g., tile or WPP) as entry points for parallelization purpose only [24].

In order to make the video coding process able to be efficiently parallelized, both predictive and entropy coding steps have to be parallelized accordingly. The predictive coding can be parallelized since a block can be predicted as soon as some of its top-left, top, top-right and left blocks are available. However, entropy coding prevents the coding process from being completely parallelized. This is because, entropy coding of one block needs entropy coding state of its precedent block. Hence, entropy coding of one block can only be started after entropy coding of its preceding block has finished. The WPP mechanism is brought up to address this problem by making entropy coding able to be parallelized. The WPP divides a video frame into CTU rows such that they can be concurrently processed. The WPP approach is based on preserving the above mentioned dependency, in entropy coding, to a large extent. In this approach, each CTU row is processed relative to its preceding CTU row with a delay of two consecutive CTUs. When WPP is enabled, the partitions (i.e., CTU rows) break any dependencies except for entropy coding. These dependencies between partitions along with the rigid row-wise partitioning make WPP useless in the partial decoding context. It is a purely parallel processing approach [24].

The last frame partitioning option is tiles. An overview of tiles in HEVC is provided in [16]. The afforded partitions are primarily designated as entry points for effective parallelism and packetization purposes. In addition, this kind of partitioning has been demonstrated to facilitate region of interest (ROI) based applications [16]. The tiling approach is similar to the situation where AVC slice group tool with map type 2 is enabled together with flexible macroblock ordering (FMO) tool, but with considerably lower complexity. Unlike slices, tiles are always rectangular and do not contain any headers.

Figure 10 illustrates a typical tiling of a video frame using vertical and horizontal tile boundaries aligned with CTU boundaries, where a frame containing 192 CTUs is divided into 15 tiles. The tile boundaries are marked with bold lines. When tiles are enabled, the regular raster scan order of CTUs changes to a tile-based raster scan order. In Figure 10, the green line shows the order of tile processing, and the red line shows the order of CTU processing within a tile.

As discussed, slice and tile are two alternative options that can be utilized for enabling video frame partial decoding. They are at the same level in hierarchical structure of bitstream. When these tools are enabled, a video frame is divided into partitions such that each constructed partition is independently decodable from other partitions within the same frame, where decoding refers to entropy, residual, and intra-picture prediction decoding. Due to this breaking of intra-picture prediction and entropy dependencies, which requires re-initialization of entropy coding state at the beginning of each partition, both standard tile and slice come along with cost of increase in bitrate.

Tiles appear to be more efficient than slices on a number of aspects. Tiles provide more flexibility to the partitioning, which brings any arbitrary partitioning arrangement. In contrast to slices, tiles incur fewer penalties of header data. Furthermore, tiles can be arranged in a more

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Figure 10. Tile-based and raster scan order of CTUs within a frame

compact shape compared to the slices [16]. This leads to less reduction of correlation between pixels within a tile, when compared to slice partitioning. As a result, among the possible frame partitioning tools, tiles are well suited for partial decoding purpose. Hence, in this work tiles are chosen for enabling partial decoding.

2.4.4 MV and 3D extensions of HEVC

The 3D video data provides a visual experience with depth perception, which composes of multiple videos, potentially with associated depth data, showing the same scene from multiple angles. This kind of representing video materials is called Multiview Video (MV) or Multiview Video plus Depth (MVD) format when presented without or with depth data, respectively. The 3D HEVC extensions MV-HEVC and 3D-HEVC are emerged to provide efficient coding of the 3D video content, supporting MV and MV/MVD data, respectively. The fundamental implementation principle of the MV extension is to re-use the coding tools of the HEVC standard while introducing only high-level changes. On the contrary, 3D-HEVC introduces new block-level dedicated 3D coding tools with the aim of improving compression efficiency compared to the MV extension. Hence, MV-HEVC is inferior in terms of R-D performance, however, it has easier implementation and provides more flexibility data format and scalability [40].

Figure 11 illustrates an overview of an exemplary system for transmission of the 3D video data format. In the encoder side, a small number of captured views as well as corresponding depth maps are encoded and transmitted. In the decoder side, the received 3D video bitstream is decoded and then necessary intermediate/virtual views suitable for any 3D display are generated using view synthesis technique as a decoder-side post-processing step, for example by means of depth-image-based rendering (DIBR) techniques. The camera parameters are also included in the bitstream for the purpose of view synthesis.

One of the views is referred to as base/independent view which is coded independently of the other views and the depth data by using a conventional 2D HEVC codec. The other views are

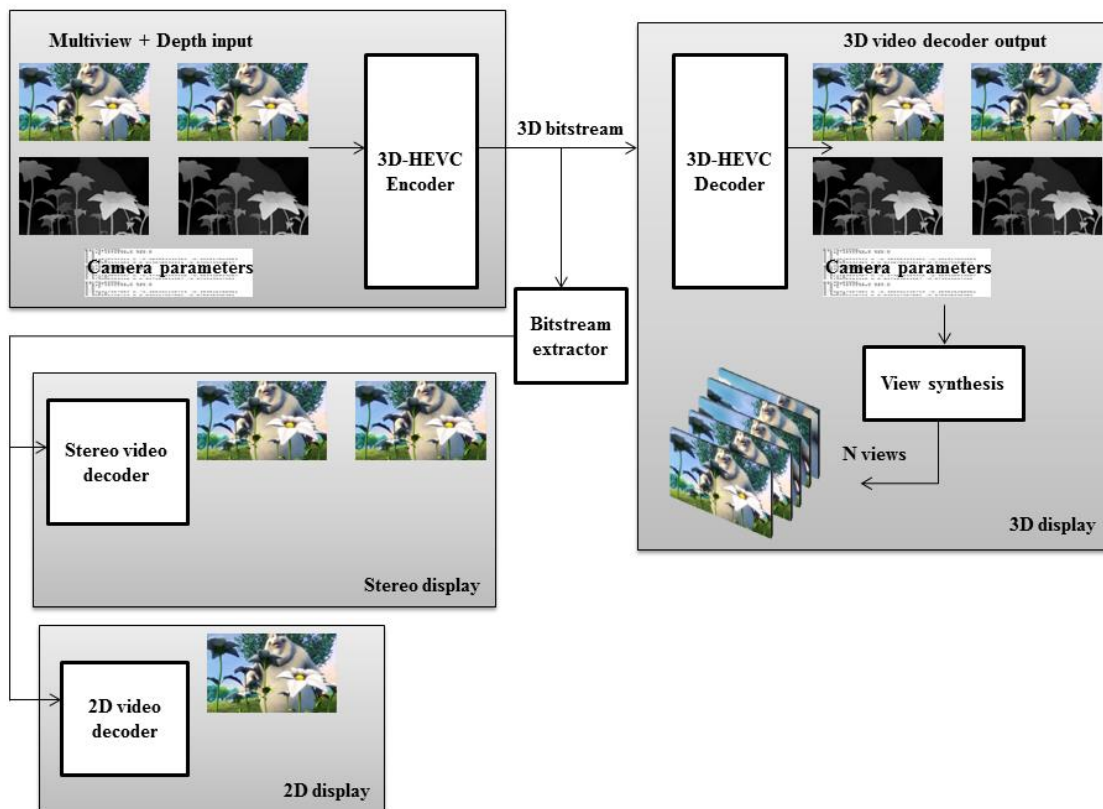


Figure 11. Overview of the system structure for 3D video transmission [32]

referred to as the dependent views which are coded using the new coding tools introduced in the 3D-HEVC extension. The base view is coded using an unmodified HEVC codec. For the dependent views, additional coding tools have been integrated into the HEVC codec, which exploit correlation among the video views. The proposed framework provides random access to a sub-bitstream representing subset of the video views. Thus, a 3D video bitstream can be partially decoded without to decoding the entire bitstream. It is possible to extract the sub-bitstream representing the base view from a 3D video bitstream and decode it using a conventional HEVC decoder in order to display on a conventional 2D display. Alternatively, a sub-bitstream representing two views can be extracted and decoded as well for feeding stereo display.

2.4.5 HEVC and its 3D extensions bitstream syntax

HEVC inherits hierarchical NAL unit based bitstream structure from AVC and uses parameter set concepts similarly to AVC. NAL units are byte aligned and each NAL unit consists of a NAL unit payload and a NAL unit two-byte header identifying the kind of payload, VCL or non-VCL NAL unit. The HEVC bitstream structure is shown in Figure 12, in the case where each frame is coded as a single slice. In addition to the parameter sets known from AVC (i.e., SPS and PPS), in HEVC parameter sets include a novel concept so-called Video Parameter Set

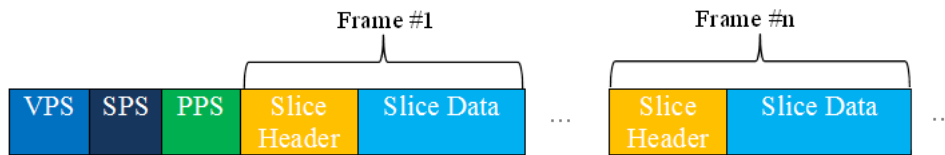


Figure 12. HEVC bitstream structure

(VPS). The bitstream starts with the new VPS which applies to all layers in layered extensions such as MV- and 3D-HEVC. Each coded frame together with the associated non-VCL NAL units is called an HEVC access unit. From this perspective, an HEVC bitstream can be seen as a series of access units. In the case of 3D video coding, an access unit refers to all video pictures and depth maps, if present, that correspond to the same time instance. Figure 13 shows access unit structure and coding order of view components in the MVD representation format for encoding of a typical MVD data comprises n views with each view having k frames.

In Figure 13, the video pictures and depth maps related to a particular view are indicated by a view order index, which does not necessarily represent the arrangement of the camera array. All video pictures and depth maps that belong to the same view have the same value of view index. In general, the coding order of access units is not necessarily identical to the capture or display order. For the base view, the video pictures are always coded before the associated depth maps. However for predicted views, depth map may be also coded before the associated video picture. Within an access unit, first the video picture and depth map with viewId 0 are coded, followed by the video picture and depth map with viewId 1, etc. A video picture and depth map with viewId equal to x are transmitted after all video pictures and depth maps with viewId less than x . The 3D video bitstream is then formed by multiplexing resulting NAL units.



Figure 13. 3D-HEVC access unit structure

The parameter sets (i.e., VPS, SPS, and PPS) were defined with a slight extension to that of conventional HEVC in order to support signalling of the new parameters, for example, the view order index and depth flag. All of the video and depth sequences are associated with a VPS. Each video sequence and depth sequence is associated with a SPS and PPS.

In 3D-HEVC, the base view and depth data are coded using new block-level video coding tools that exploit statistical dependencies between video picture and depth map. For example, the new inter-component prediction techniques employ already coded data within an access unit to exploit such dependencies. The inter-component prediction can be configured such that video pictures can be decoded independently of the depth data [32]. This feature is useful in the context of this work since only the video pictures are intended to be partially decode. In order to provide full-picture depth data, depth data is decoded fully.

3. PROPOSED METHOD (PARTIAL DECODING)

This section includes the discussion of implementation of the video picture partial decoding. The proposed method and modifications to build self-contained partitions in the JM AVC/H.264 and HEVC/H.265 HTM reference software packages are described.

3.1 Partial decoding implementation possibilities

It is a responsibility of the encoder to build self-contained regions which prohibit any kind of dependency on other partitions within the current picture and also non-co-located partitions in the reference pictures. After that, partial decoding can be performed either using a conforming or non-conforming decoder. In the case of using a conforming decoder, the encoder must ensure that all conditions for bitstream conformance are fulfilled, otherwise, a dedicated decoder has to be designed.

The decoder may be provided with a full-picture bitstream containing all partitions or a sub-bitstream containing only the necessary partitions. In the former case, the decoder must be capable of skipping decoding of unnecessary partitions. This capability is generally accomplished by providing the decoder with the knowledge of necessary partitions' indices using signalling information techniques (e.g., SEI messages). In the case of conforming decoder, however, the decoder behaves naturally, provided that the sub-bitstream is standard compliant. This allows the use of the off-the shelf hardware decoders (e.g., those included in GPUs), without need to changing the decoder. Thus, partial decoding may require signalling information to decoder or may not.

An alternative to the partial decoding solution is to crop out pixel data corresponding to necessary regions in the pixel domain. It results a region-rearranged video data in which some regions are removed and the remaining regions append together. The region-rearranged video is then encoded with a standard encoder (i.e., with no restriction on the partitions). The resulted bitstream can be decoded with a conforming decoder. The advantage of this approach is that no encoder change is required. The extra pre-processing stage is required for cropping out the necessary partitions, which may not be efficient for any partitioning arrangement. This leads this approach not to be applicable in our study case.

In this work, the source video is encoded using an encoder which is capable of creating self-contained partitions. Using an intermediate extractor, the coded elements related to unnecessary partitions are dropped from the bitstream to form a sub-bitstream containing only coded elements related to necessary partitions. The extractor can be placed either in the encoder/server side or display side prior to decoding. The former case results in streaming/display bandwidth reduction. Consequently, in the display side any partition of interest can be selectively decoded since all partitions are independent of each other.

3.2 Enabling the self-contained slices in the AVC standard

3.2.1 General case

The first step toward enabling the self-contained slices is to enable the slicing tool provided by the standard. The slices can be either configured based upon number of MBs in the slice or number of bytes in the coded slice. In the first case slices are constructed with a fixed number of MBs. In contrast, in the second case slices may contain varying number of MBs. While the latter one is useful for transmission application by providing fixed-size network packet, the first case is much more practical in the context of partial decoding, since it provides regular spatial partitioning of a video frame. Regular partitioning means each slice occupies the same spatial region over all frames. Moreover, it forces all frames of a video sequence to have the same partitioning scheme. This is not the case in the second slice configuration approach where the number of slices per frame might change among frames. In addition, regular partitioning allows to instantly determine the spatial region in the frame that a specific slice occupies. This information is required in constructing self-contained slices. In Figure 14 two different slice arrangements are illustrated. The left picture is split into 6 slices with larger slice sizes compared to the right picture which is constructed from 52 slices.

Two configuration parameters are designed for slice configuration called *SliceMode* and *SliceArgument* parameters. Assigning 0 to the *SliceMode* parameter disables the slicing tool. *SliceMode* equals to 1 and 2 refers to the abovementioned slice configuration approaches. If *sliceMode* set to 1, the value assigning to the *sliceArgument* determines the largest possible number of MBs in a slice. For *sliceMode* equals 2, the value assigning to the *sliceArgument* determines the largest possible number of bytes in a coded slice. A regular rectangular slicing achieved by partitioning frames with the same number of slices and defining *sliceArgument* to a fix number of MBs. The *sliceArgument* is calculated according to the equation (2).

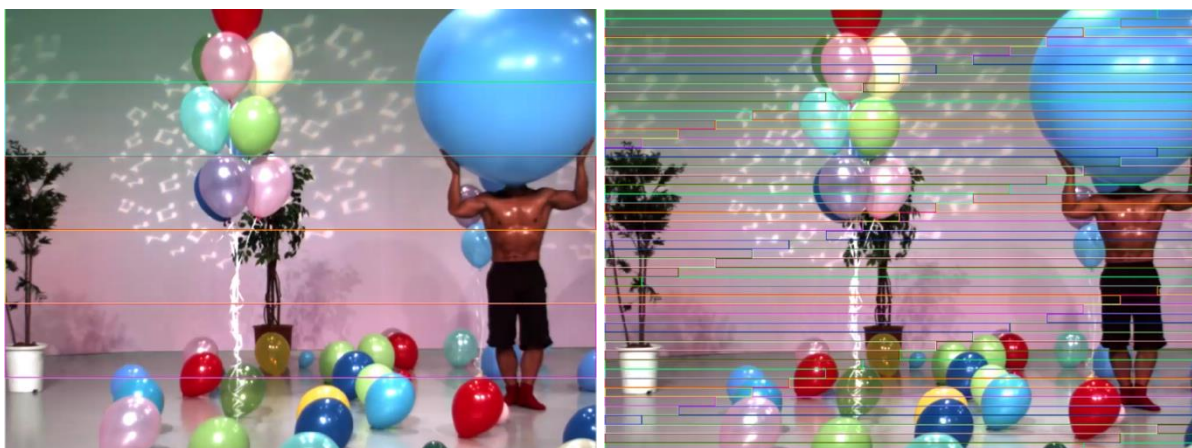


Figure 14. Two different slice arrangements in H.264, drawn with [41]

$$\text{SliceArgument} = \text{ceil}((\text{frame width} * \text{frame height}) / (16^2 * \text{numberOfSlices})) \quad (2)$$

Standard slices are self-contained in the sense that their syntax elements can be parsed from the bitstream and they contain all data which is required on the decoder side. This is true by considering the fact that correctly decoding of slices requires sequence parameter set (SPS) and picture parameter set (PPS) to be received in the decoder side. Nevertheless, in the context of partial decoding, standard slices cannot be called self-contained (i.e., independently decodable). This is due to the fact that correct decoding of a slice depends on previous pictures (i.e., reference pictures).

The usage of the standard slices (i.e., what is introduced by the standard) restricts intra-picture prediction. In addition, the standard slices are independent in terms of entropy coding since entropy coding engine is re-initialized at the beginning of each slice. Therefore, each slice can be decoded independently from other slices within the same frame. In inter-picture prediction, however, slice boundaries are disregarded, where motion compensation can cross slice boundaries. This further means decoding of each slice of a frame can depend on non-co-located slices in the other frames. Hence, the usage of standard slices is limited in partial decoding applications because of this dependency. Some extra requirements must be imposed to the standard slices in order to be applicable in partial decoding applications. In the following, those requirements are introduced and the resulting slicing is discussed in more details.

The correct decoding of one slice depends on decoding of previous frames so that they must be entirely available. For example, losing one slice affects the decoding of all following slices in which case their motion vectors point to some regions of the lost slice. Furthermore, losing of an entire slice makes the first coded MB 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 because of the H.264 variable length coding property.

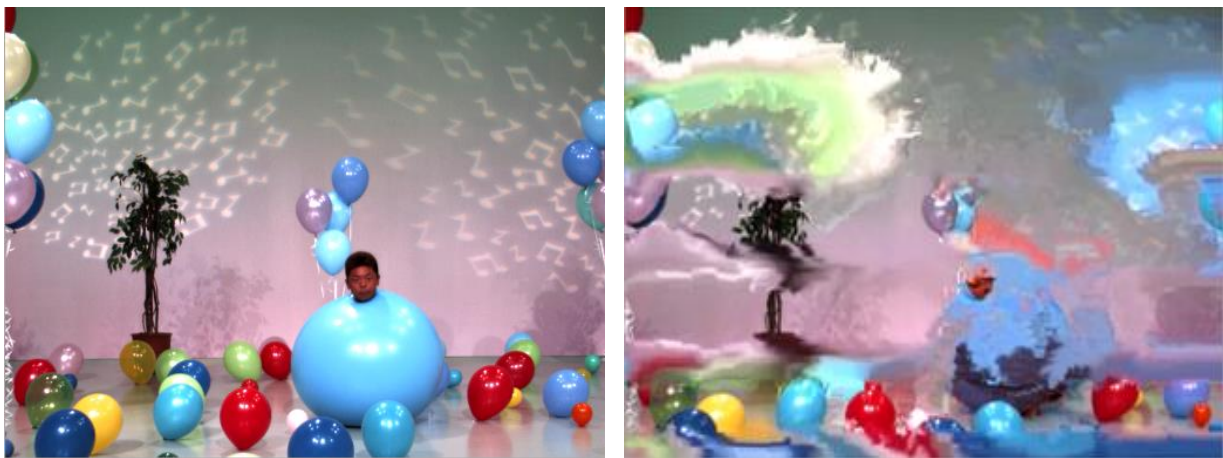


Figure 15. Left: A correctly decoded frame, right: A decoded frame of a bitstream in which every other slices were dropped

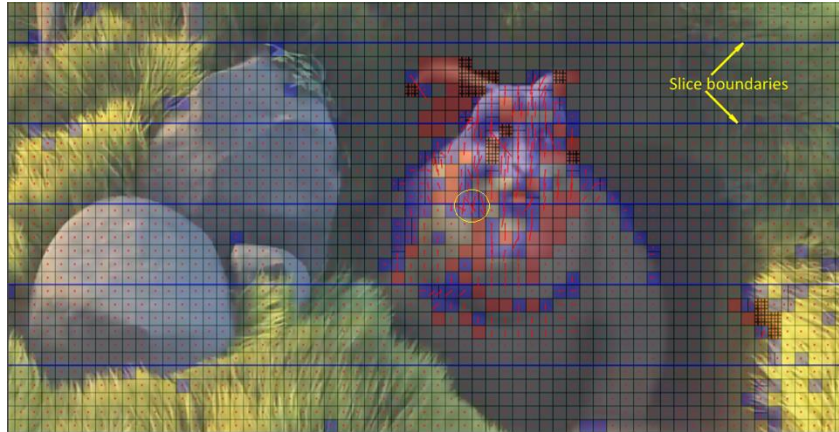


Figure 16. *Standard slices in H.264 [34]*

Figure 15 illustrates this dependency among slices. It shows the decoding result of a confirming bitstream and a bitstream in which some coded slices are dropped from. The error propagation is growing as higher number of frames is being decoded.

3.2.2 Self-contained Slices

It is the responsibility of the encoder to make slices self-contained and to ensure that each slice can be decoded regardless of the fact that other slices, in other spatial regions than what is covered by the slice, may have been lost. We formulate three requirements which have to be fulfilled in the encoder side.

- 1) Restrict intra-picture prediction to within current slice boundaries. As discussed above, the usage of standard slices already restricts intra-picture prediction.
- 2) Ensure regular slice partitioning. This means each slice occupies the same spatial region over all frames which have the same number of slices. This is realized by defining the *sliceArgument* parameter to a fix number of MBs. The 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.
- 3) Limit the inter-picture prediction to within slice boundaries in order to provide both spatially and temporally independently decodable slices. This guarantees that motion vectors only point to the same spatial area as the current slice in the reference pictures.

In Figure 16, some troublesome motion vectors that cross slice boundaries are shown inside yellow circle, for example. In contrast to the standard slices, self-contained slices can be coded independently. Hence, losing/skipping one or more slices leaves the other areas of a frame unaffected such that the remaining slices can still be decoded. Thus, self-contained slices facilitate partial decoding.

The main contribution of this thesis in the H.264 part is to prevent motion vectors from referring to non-co-located slices, with respect to the current slice, in the reference picture (s) using encoder restrictions. The JM H.264 reference encoder is modified to fulfill the third

requirement. For restricting motion vectors within slice boundaries, the search range is initially constrained to lie within the co-located slice by considering interpolation filter tap size, in case of sub-sample prediction, and the current block size. Each time that a predictor is estimated, its associated motion vector is examined to be within the restricted range. The examination is made based on the location of the current macroblock and the geometry of partitions within the frame. In case a motion vector breaks slice boundaries, the associated predictor is ignored.

3.3 Enabling the self-contained tiles in the HEVC standard

3.3.1 General case

This section defines the tile-related configuration parameters and describes how to enable tile partitioning in HEVC based on the HTM reference software version 14.0. Further, the implementation of the HEVC self-contained tiles is explained. The tiling is enabled by defining at least one of the related configuration parameters, namely, *NumTileColumnsMinus1* and *NumTileRowsMinus1* as non-zero values. The parameters determine number of tile columns and rows, respectively. The number of tiles within a frame is then determined using equation (3).

$$\text{number of tiles} = (\text{NumTileColumnsMinus1} + 1) * (\text{NumTileRowsMinus1} + 1) \quad (3)$$

The tile size can be defined in two ways which are specified by *TileUniformSpacing* configuration parameter. When *TileUniformSpacing* is set to 0, tiles are uniformly spaced based on the *NumTileColumnsMinus1* and *NumTileRowsMinus1* parameters. If *TileUniformSpacing* is set to 1, tile sizes are defined by two separate arrays, namely, *TileColumnWidthArray* and *TileRowHeightArray* which define a space or comma separated list of widths and heights, respectively, of each tile column or tile row. An example of the two tiling configuration approaches is described in Table 2, where both configurations result in the same tile arrangement shown in Figure 17 (c).

Table 2. Configuration parameters for two different tiling configuration approaches

Configuration parameter	Uniformly spaced tile configuration	Defining tile sizes using width and height arrays
<i>TileUniformSpacing</i>	1	0
<i>NumTileColumnsMinus1</i>	4	4
<i>TileColumnWidthArray</i>	-	3 3 3 3
<i>NumTileRowsMinus1</i>	2	2
<i>TileRowHeightArray</i>	-	4 4

Table 3. Configuration parameters to obtain the highest possible number of tiles

Configuration parameter	Value	Description
MaxCUWidth	16	Defines maximum coding unit width in pixel
MaxCUHeight	16	Defines maximum coding unit height in pixel
MaxPartitionDepth	2	The depth of CU tree
QuadtreeTULog2MaxSize	4	Defines maximum TU size in log2
QuadtreeTULog2MinSize	2	Define minimum TU size in log2
UniformSpacingIdc	1	Defines the mode used to determine tile sizes
NumTileColumnsMinus1	Width/16 -1	number of tile columns
NumTileRowsMinus1	Height/16 -1	number of tile rows

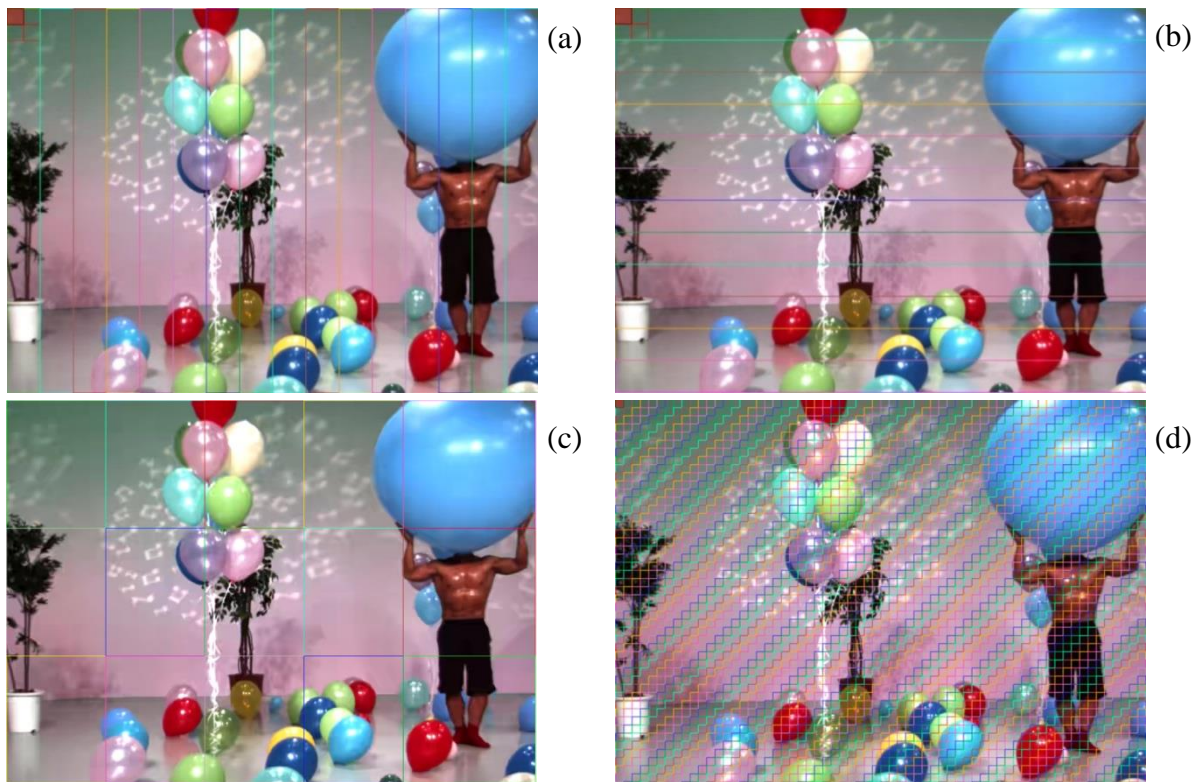


Figure 17. Tile partitioning, Balloons sequence with resolution 1024x768 [41]

a) Vertical partitioning

c) Vertical and horizontal partitioning

b) Horizontal partitioning

d) partitioning into maximum number of tile

Tiles provide horizontally and/or vertically partitioning of a video frame. Four different tile partitioning schemes are shown in Figure 17. A frame/slice may be partitioned into only one tile or multiple tiles. In the former case, we would have the tile with the biggest possible size as the frame/slice size, which is considered as slice rather than tile. This is obtained by defining the NumTileColumnsMinus1 and NumTileRowsMinus1 configuration parameters as zero.

The highest possible number of tiles within a video frame is achieved in the situation where each tile has the same size as the minimum CTU size (16x16). 0 shows the related configuration parameters' values for obtaining the highest possible number of tiles in a typical video frame. The concluded tiling is shown in Figure 17 (d) for a sequence at resolution 1024x768, where the frame is maximally partitioned into $\left(\frac{1024}{16}\right) * \left(\frac{768}{16}\right) = 3072$ tiles. Figure 17 (a) and (b) show partitioning of the same frame into maximum vertical and horizontal number of tiles for maximum CTU size of 64x64 using Main profile of HEVC, respectively.

3.3.2 Self-contained tiles in conventional HEVC

As one of the requirements for enabling partial decoding, regular tile partitioning (i.e., each tile occupies the same spatial region over all frames) is facile because of the provided partitioning flexibility by tiles.

Similar to the AVC standard slices, the usage of the HEVC standard tiles restricts intra-picture prediction and entropy coding within tile boundaries [16]. These two independencies among the standard tiles play a big role in facilitating tile-based partial decoding. However, the standard tiles suffer from inter-picture prediction, in the partial decoding context [13]. This issue prevents tiles from being independently decodable and further applicable in partial decoding applications. To remedy the drawback of the inter-prediction scheme, some restrictions in the encoder side has to be imposed. In this work, the HTM reference software is modified in order to provide self-contained tiles.

In order to construct self-contained tiles based on the introduced standard tiles, the same requirements as we had for creating self-contained slices in AVC have to be fulfilled in HEVC. This means restricting inter-picture prediction to within co-located tiles in the reference frames, with respect to the current tile. Compared to the AVC case, restriction of inter-picture prediction in HEVC is a more effort demanding task and requires more considerations to be dealt with. For that, the merge and AMVP lists described in Sub-section 2.4.2 are prevented to be filled with candidates in which motion vector points outside the current tile boundary.

Figure 18 depicts the most suitable spatio-temporal candidates for the merge and AMVP techniques [32]. The current and co-located PUs belong to the current and the reference pictures, respectively. In the derivation of spatial candidates, a candidate is considered as not available if the associated PU belongs to another tile, or is intra coded. As a consequence, in both motion data signalling methods, spatial candidates are selected such that they belong to

the current tile. The temporal candidate is selected between C_0 and C_1 positions. If PU at position C_0 is not available, beyond the current CTU row, or intra coded, position C_1 is selected [32]. This way of constructing the temporal candidates has to be restricted for the rightmost PUs within the current tile, in which C_0 falls in the next CTU column leading to breaking of the tile boundaries. Furthermore, both tile and slice partitioning tools suffer from the fact that motion vectors can freely point to any predictor beyond the current and co-located slice in the reference pictures, which may lead to breaking tile boundaries. The above description can be extended to the HEVC slice concept as well.

According to the HEVC standard, for the bottom-right-most PUs within the current tile the temporal merge and AMVP candidates are restricted to the co-located candidate (i.e., C_1 in

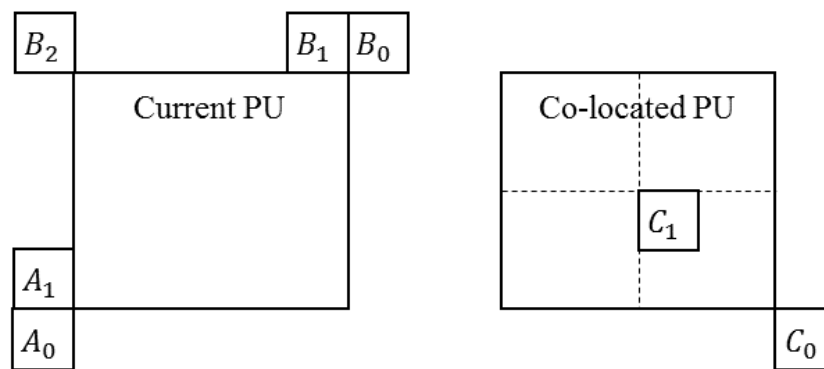


Figure 18. Spatio-temporal positions of the merge and AMVP candidates [32]

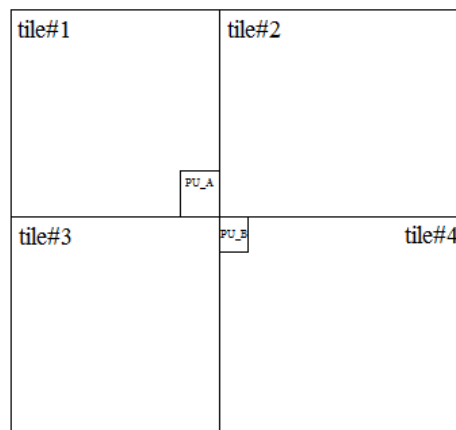


Figure 19. Bottom-most PU within a tile

Figure 18) in the reference pictures. In the exemplary tiling illustrated in Figure 19, this means for the prediction block PU_A in tile 1, its bottom-right neighbour (PU_B in tile 2) is set unavailable. Accordingly, PU_A is prevented to choose motion vector of PU_B as its temporal motion vector prediction (TMVP) candidate. However, the motion vectors of the remaining right-most PUs in the tile 1 maybe predicted from their bottom-right candidates, which fall in

the tile 2. Therefore, the encoder restriction is imposed only on the right-most PUs in which their TMVP candidate may fall in the tile 2.

3.3.3 Self-contained tiles in 3D extensions of HEVC

Since the MV-HEVC block-level coding tools are the same as those of conventional HEVC, no additional restrictions are required in order to enable partial decoding, with respect to the restriction introduced in the case of HEVC. However, in 3D-HEVC, there is one more restriction to be fulfilled for enabling partial decoding of video pictures. In the HEVC motion compensation process, the merge candidate list reconstruction does not involve motion information correlation between views, while the motion information of two associated blocks in base view and predicted view is likely to be the same. This is because they represent two different projections of the same scene. Therefore, in 3D-HEVC additional candidates for the reconstruction of merge list are derived, which are related to the temporal motion information and disparity motion information of already coded blocks in the base view. Therefore, the maximum number of candidates in the final merge list is increased to 6. In order to guarantee that motion vectors do not cross tile boundaries, the selection process of the additional candidate has to be restricted as well as the other five candidates which are selected the same as in HEVC [32].

3.4 Slice/tile-based extractor

In partial decoding applications, reduction in bandwidth is achieved by dropping coded elements related to unnecessary partitions from the bitstream. In this case, decoder receives a smaller bitstream containing only coded elements related to necessary partitions (e.g., ROI regions). As the coded slices/tiles are interleaved with other coded data in the bitstream and as parameter sets and headers (e.g., slice segment header) are for the entire bitstream, a dedicated decoding process is defined for decoding particular slices/tiles, while omitting the decoding of other slices/tiles. In case of self-contained slices/tiles, the decoder does not need the coded elements related to the unnecessary slices/tiles for correctly decoding of necessary slices/tiles. Nevertheless, they cannot be simply dropped from the bitstream because this results in a corrupt bitstream which cannot be copped with using a standard decoder. For that, prior to transmission/decoding, a slice/tile-based extractor is designed to construct a full-picture-compliant bitstream corresponding to the desired slices/tiles such that a standard decoder can cope with that. The extractor is designed based on the knowledge of bitstream syntax (e.g., NAL units' arrangements, slice header syntax, and slice/picture delimiters) as described in the Sub-sections 2.3.4 and 2.4.5 for AVC and HEVC, respectively.

Each NAL unit in the bitstream is separated by some specific start codes or delimiters that define boundaries between coded sections. The delimiters facilitate accessing coded units in level of frame or slice. In both AVC and HEVC, the delimiters are unique sequences of four

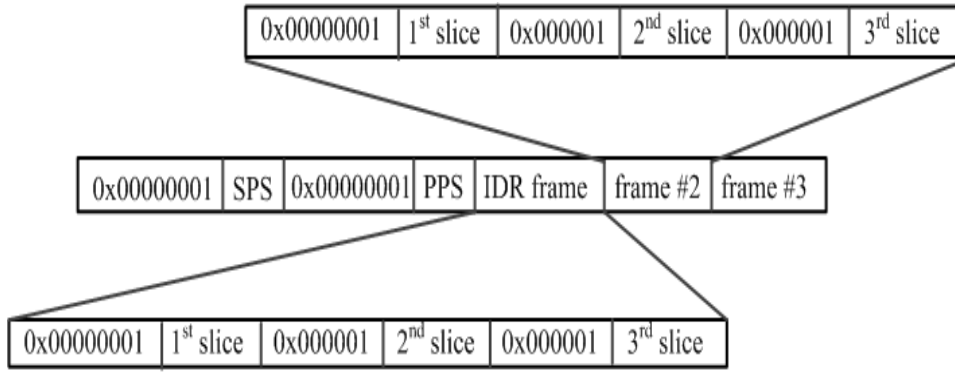


Figure 20. *Delimiters in a typical bitstream*

bytes equal to 0x00000001 and three bytes equal to 0x0000001 in level of frame and slice, respectively. A low level of a typical bitstream with delimiters is shown in Figure 20.

In case of tile partitioning, the entry points of tiles are signaled in PPS. In contrast to slices, no delimiters designated for tiles. Accessing each tile requires to read its entry point from the slice header in which the tile belongs. Since the tile signaling is included in PPS, it is possible to construct frames of a video sequence with different tiling arrangements (i.e., tile partitioning on a per frame basis).

The first coded tile immediately follows the slice header to which it belongs. The locations of the subsequent tiles, if any, are explicitly signaled in the slice header along with the tile index values ranging from 0 to the number of tiles in the video frame minus 1. The locations of first and last bytes of the k -th tile in the bitstream are calculated using equations (4) to (6) as follows:

$$firstByte[k] = startLoc + \sum_{n=1}^k (entry_point_offset_minus1[n - 1] + 1) \quad (4)$$

$$lastByte[k] = firstByte[k] + entry_point_offset_minus1[k] \quad (5)$$

Where $entry_point_offset_minus1[i] + 1$ specifies $(i + 1)$ th tile entry point offset in bytes and $startLoc$ is the entry point of the slice data, which is preceded by the associated slice header:

$$startLoc = location\ of\ last\ byte\ of\ slice\ header\ in\ bitstream + 1 \quad (6)$$

For example, consider a bitstream containing one coded frame which is split into three horizontal tiles. We can specify the location of each tile and extract it from the bitstream by using the information provided in Table 4. According to the equations (4) to (6) we have:

$$\begin{aligned} startLoc &= (entry\ point\ of\ slice\ NAL\ unit + NAL\ unit\ start\ code\ size \\ &\quad + NAL\ unit\ header\ size + slice\ header\ size) \\ &= 0x00000066 + 0x00000003 + 0x00000002 + 0x00000007 \\ &= 0x00000072 \end{aligned}$$

Table 4. *The required information for parsing tiles from bitstream*

Parameter name	Value
num_entry_point_offsets	2
entry_point_offset_minus1[0]	0x00000ce1
entry_point_offset_minus1[1]	0x00001898
Entry point of slice NAL unit	0x00000066
NAL unit start code (0x000001) size	3 bytes
NAL unit header size	2 bytes
Slice header size	7 bytes

The entry and last byte locations of the three tiles in the bitstream are calculated as below:

firstByte_location_0= startLoc,

lastByte_location_0= startLoc + 0x00000ce1= 0x00000d53

firstByte_location_1= 0x00000d54,

lastByte_location_1= 0x00000d54 + 0x00001898 = 0x000025ec

firstByte_location_2= 0x000025ed,

lastByte_location_2= The byte before the next slice NAL unit start code.

The designed slice-based extractor in the AVC part, simply drops the NAL units of unnecessary partitions from the bitstream. While it results in a corrupt bitstream which is not decodable using a standard decoder, an FFmpeg decoder can cope with that. The FFmpeg decoder [20] supports multiple standards and AVC is used as default. In this work, the FFmpeg codec is used to decode the AVC sub-bitstream only containing the desired slices. In this case, decoding time of the sub-bitstream is not reduced compared to decoding of the entire bitstream if the decoder performs the error concealment algorithm in case of the dropped slices. In order to improve decoding time, the FFmpeg error concealment algorithm is disabled while dealing with the dropped slices.

In case of HEVC, the tile-based extractor was implemented on high syntax level. The NAL units related to desired tiles are extracted from the bitstream containing the whole video and copied to the output sub-bitstream. In order to provide an HEVC conforming bitstream altering some parameters in the parameter sets are required. These parameters include slice segment address, dimension of the output video sequence, number of tile rows and columns, and tile enabled flag. The latter is disabled in case the output sub-bitstream contains only one tile. The operation results in a smaller bitstream which can be decoded using a standard decoder.

A sub-optimal alternative to the extractor is also presented in [11]. The proposed method in [11] replaces each unnecessary slice with a slice of minimal size containing spans of macroblocks in which the coding mode is set to skip mode. The standard decoder can decode

the resulted bitstream at the cost of higher bitrate compared to the case of using the proposed extractor in this thesis, which fully drops the unnecessary coded elements. The proposed solution in [11] not only provides a bitstream that can be handled by the standard decoder but also may cause reduction in decoding time. This is because of defining the coding mode of replacement regions as skip mode. Note that decoding time is not decreased if the decoder performs an error concealment algorithm with less complexity than the skip mode.

4. RESULTS AND PERFORMANCE EVALUATION

This section describes the experimental condition and provides some insights into the performance of the proposed partial decoding approach. The performance of the proposed self-contained slices/tiles is evaluated in terms of storage and decoding complexity, in the context of light-field 3D displays.

4.1 Experimental setting

The proposed encoder modifications were implemented in the AVC reference JM encoder version 18.6 [42] and HEVC reference software HM version 16.7 [26]. The designed slice-based and tile-based extractors were used to build the sub-bitstream containing desired slices and tiles for AVC and HEVC bitstreams, respectively. For AVC part the FFmpeg decoder and in the case of HEVC the standard decoder was used. The in-loop filtering across slices and tiles were disabled in AVC and HEVC, respectively. The simulation were performed using HEVC random access Main profile and AVC Baseline profile with 89 frames per sequence. The quantization parameters (QPs) are selected in the range of 22-34 with delta QP equal to 4. The decoding refresh type was set to Instantaneous Decoding Refresh (IDR) and Clean Random Access (CRA) pictures in AVC and HEVC, respectively, with period of 32. The reported results include compression efficiency in terms of Bjøntegaard Delta-rate (BD-rate) criterion [43]. Positive/negative values indicate how much the bitrate is increased/reduced for the same peak signal-to-noise ratio (PSNR). The decoding speed is reported in unit of frame per second.

4.2 Test sequences

The reported experiments were conducted on test sequences with different contents varying from low motion to high motion. The test sequences include five monoscopic video sequences, namely, Balloons, Pantomime, Kimono, BBB-flower, and Shark. While the first three sequences were captured in real world, the last two sequences are computer-animated sequences. The resolution of sequences along with a short description is provided in Table 5.

Table 5. Test sequences description

Sequence	Resolution (wxh)	Description
Balloons	1024x720	complex object motion, moving camera
Pantomime	1280x960	medium complex object motion, no camera motion
Kimono	1920x1280	Low object motion, no camera motion
BBB-flower	1280x768	Big Buck Bunny light field sequences [18], the Flowers scene, fast object motion, no camera motion
Shark	1920x1088	fast large object motion, no camera motion

4.3 Partitioning arrangement

In this experiment, the video frames are divided into four vertical partitions in a regular manner, such that each partition occupies the same spatial region over all frames. This kind of vertical partitioning is practical in the LF conversion process employed in the LF displays. Each partition covers about 25% of the whole frame. Figure 21 (a) illustrates the described partitioning arrangement for the Balloons video sequence. In AVC, slices are arranged in raster scan order in a video frame. In order to provide vertical slicing, the video frames need to be rotated 90° before encoding.

In case of HEVC, each tile contains exactly one slice which its boundaries match the tile's boundaries. The above-described partitioning could be also achieved by utilizing only tiles. The reasons for using a combination of tile and slice are to allow partitioning of video frames to vertical stripes, and low-complexity implementation of partial decoding operation. According to the HEVC standard, the end of a slice is indicated using the `end_of_slice_flag` which is the last encoded element for each CTU. It is set to one if the current CTU is the last CTU in the slice. In case of using tile alone as partitioning tool, only for the last CTU of the last tile (i.e., most bottom right tile) within a frame the flag is set. Assume that in a LF rendering node, only the pixel samples corresponding to the first tile is requested. Therefore, the extractor generates a sub-bitstream containing the first tile of each frame, in which the `end_of_slice_flag` equals to 0. The generated sub-bitstream is not decodable by a standard HEVC decoder, as the decoder does not meet end of slice condition while decoding the last CTU within the frame in the output sub-bitstream. The flag is entropy coded and cannot be modified without entropy decoding the slice data. The proposed partitioning helps to avoid such situation since the last CTU within overlapped slices and tiles are the same. Hence, the flag is set for each tile. However, it comes along with a cost of increase in bitrate. The results (see Table 8) show that the bitrate increment caused by slice header is less than 1% for the experimented video sequences. The amount of overhead is such small that can be negligible.

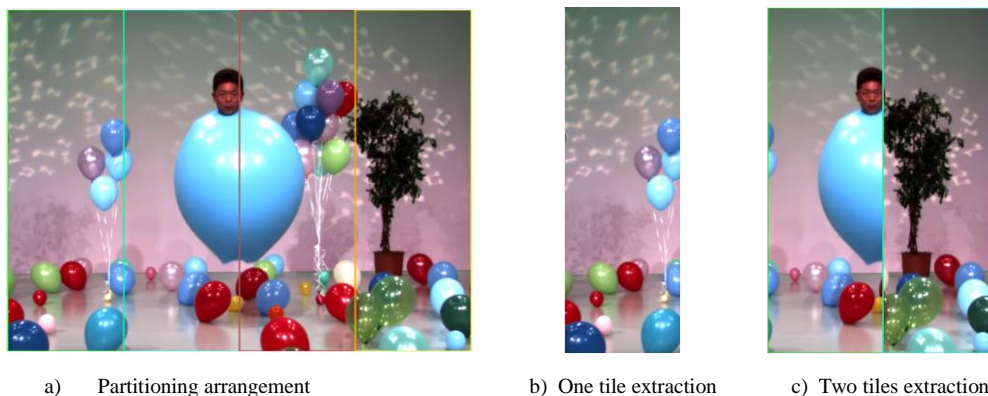


Figure 21. *Frame partitioning and the extracted partitions*

4.4 Results analysis

4.4.1 Compression performance

The standard slices/tiles can be considered self-contained when all pictures are intra coded at the expense of considerable increase in bitrate. Table 6 shows the storage performance of only intra coding scheme with respect to Baseline and Random Access Main encoding profiles in AVC and HEVC, respectively. The intra coding scheme increases bitrate about 142% and 363 % in AVC and HEVC, respectively, on average over the test sequences. Hence, it is not practical to achieve self-contained slices/tiles using intra coded pictures.

As discussed in Sub-section 2.4.3, the usage of tiles and slices comes along with compression efficiency penalty. Table 7 represents the compression loss caused by standard slices and the proposed self-contained slices in AVC for the described partitioning in previous section. The usage of the standard slices introduces 2% reduction in the compression performance relative to the case where no slicing is performed. The proposed self-contained slices introduce 33% compression loss which is high when compared with that of the standard slices. The low compression loss of self-contained tiles in HEVC, explained in the following, indicates that the proposed method in AVC is further capable of providing better storage performance by optimizing the proposed method.

Table 6. Storage performance comparison between Intra profile and Baseline, and Random Access Main profiles in AVC and HEVC, respectively (delta-bitrate %)

Sequences	AVC	HEVC
Balloons	297.24	669.42
Pantomime	70.02	129.39
Shark	7.05	83.94
BBB-flower	287.16	665.12
Kimono	52.26	267.46
Avg.	142.75	363.07

Table 7. AVC slice overhead (BD-rate %)

Sequences	Normal slices	Self-contained slices
Balloons	1.75	33.43
Pantomime	0.46	45.31
Shark	6.21	42.15
BBB-flower	1.28	34.38
Kimono	0.37	9.93
Avg.	2.01	33.04

Table 8. HEVC slice and tile overhead (BD-rate %)

Sequences	Tiles	Normal tiles & slices	Self-contained tiles & slices
Balloons	1.46	2.39	6.16
Pantomime	0.79	1.17	4.02
Shark	3.09	3.78	10.11
BBB-flower	2.00	2.78	2.09
Kimono	0.14	0.40	10.26
Avg.	1.50	2.10	6.53

In case of HEVC, Table 8 represents the compression loss for cases in which video frames are partitioned using only tiles, a combination of standard tiles and slices, and a combination of self-contained tiles and slices, for the partitioning scheme described in Sub-section 4.3. The BD-rate results shows that breaking of intra prediction and re-initialization of entropy coding engine (i.e., only tiles used) cause 1.5% reduction in compression efficiency in average, when compared with no partitioning case. Additionally, the usage of slice arises with 0.6% (= 2.10% – 1.5%, in Table 8) higher compression loss caused by slice header overhead. Moreover, the results indicate that the proposed modifications into the encoder, in order to provide self-contained tiles, drop compression efficiency by 4.43% (= 6.53% – 2.10%, in Table 8). The latter loss caused by restricting motion vectors within co-located tile boundaries in the reference pictures, which results in temporal correlation reduction in inter-prediction scheme. Overall, the proposed approach introduces 6.53% loss in compression efficiency for the employed partitioning scheme over the test videos.

Figure 22 shows the average BD-rate over number of slices/tiles corresponding to Table 9 and Table 10 for two vertical partitioning cases. In the first case only tile and in the second case a combination of overlapped tile and slice is employed. As can be seen, the finer the partitioning is, the higher the compression loss occurs. As the number of partitions increase, the difference between two curves is higher since the slice overhead becomes more visible. Moreover, it can be seen that tiles are more efficient than slices in terms of compression performance, when comparing the red (only tiling) and blue (tiling and slicing) curves. The drawn conclusion above can be observed for each test sequence represented in Table 9 and Table 10.

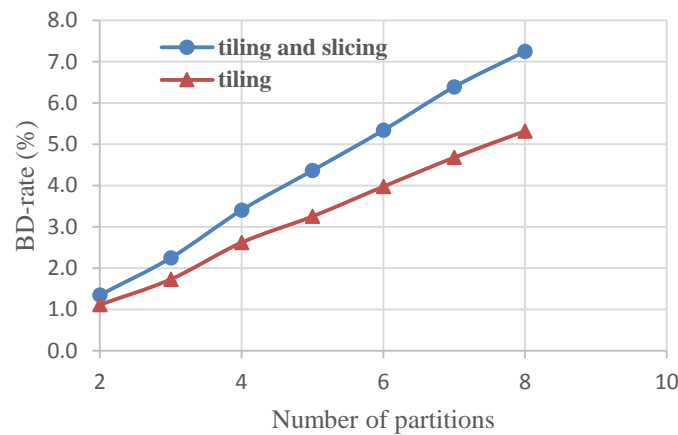
The compression loss introduced by slices/tiles also depends on the partitioning direction. 0 and 0 show BD-rate overhead for different tiling schemes with the same number of tiles as the first three columns in Table 9 and Table 10, respectively, but with horizontal tiling. Further horizontal tiling is not possible in some of the sequences, since the tile width in the HEVC standard is limited to 256 luma samples. Comparing Table 9 and Table 10 with 0 and 0, respectively, shows that horizontal tiling, performed in this work, incurs lower compression penalty on average over the test sequences.

Table 9. HEVC tile overhead for different vertical partitioning schemes (BD-rate %)

Sequences	1x2	1x3	1x4	1x5	1x6	1x7	1x8
Balloons	1.18	1.77	2.71	3.44	4.23	4.90	5.68
BBB_flower	1.06	1.81	2.64	3.71	4.37	5.15	5.74
Kimono	0.35	0.72	1.03	1.28	1.63	1.82	2.18
Pantomim	0.74	1.28	1.80	2.34	2.89	3.52	3.76
Shark	2.23	3.06	4.92	5.50	6.76	8.02	9.23
Avg.	1.11	1.73	2.62	3.25	3.98	4.68	5.32

Table 10. HEVC tile and slice overhead for different vertical partitioning schemes (BD-rate %)

Sequences	1x2	1x3	1x4	1x5	1x6	1x7	1x8
Balloons	1.50	2.55	3.96	5.10	6.34	7.34	8.65
BBB_flower	1.36	2.43	3.60	5.01	5.99	7.12	8.05
Kimono	0.48	1.06	1.48	1.90	2.32	2.76	3.30
Pantomim	0.83	1.53	2.20	2.91	3.59	4.32	4.75
Shark	2.57	3.67	5.80	6.91	8.46	10.40	11.50
Avg.	1.35	2.25	3.41	4.37	5.34	6.39	7.25

**Figure 22.** BD-rate over number of tiles**Table 11.** HEVC tile overhead for different horizontal partitioning schemes (BD-rate %)

Sequences	2x1	3x1	4x1
Balloons	0.70	0.93	1.46
BBB_flower	0.92	1.35	2.00
Kimono	-0.14	0.22	0.14
Pantomim	0.35	1.12	0.79
Shark	1.23	2.23	3.09
Avg.	0.61	1.17	1.50

Table 12. HEVC tile and slice overhead for different horizontal partitioning schemes (BD-rate %)

Sequences	2x1	3x1	4x1
Balloons	0.94	1.82	2.39
BBB_flower	1.18	1.98	2.78
Kimono	-0.02	0.52	0.40
Pantomim	0.49	1.33	1.17
Shark	1.55	3.00	3.78
Avg.	0.83	1.73	2.10

4.4.2 Complexity and latency reduction

In order to provide some insight in the decoder speed increment, four different scenarios were tested. They include the cases where a LF rendering node requests the area corresponding to: one tile, two tiles, three tiles, and four tiles (i.e., the whole frame). In the display side, only the requested tile(s) are received and decoded. Figure 21 (b) and Figure 21 (c) illustrate the cases where one tile and two tiles are decoded, respectively.

In Table 13 and Table 14, decoding speed related to the above test scenarios is represented in AVC and HEVC, respectively. The tables show average decoding speed over the test sequences for different QPs. In both AVC and HEVC, a significant increase in decoding speed is achieved by decoding only the desired region. Compared to AVC, in case of HEVC higher decoding speed is observed.

In AVC, the decoding speed increases 3.8, 2.1, and 1.4 times when decoding one, two and three tiles, respectively, compared to decoding the whole frame area. In HEVC, the decoding speed increases 4.6, 2.3, and 1.5 times when decoding one, two and three tiles, respectively. In Figure 24 and Figure 24, it can be seen that the decoding speed increment has almost a linear behavior in both AVC and HEVC, respectively.

Table 13. Decoding speed in AVC (frame per second)

QP	1 slice	2 slices	3 slices	Whole frame
22	41.26	20.51	12.53	9.33
26	45.01	24.41	17.98	11.09
30	48.00	27.89	18.05	13.84
34	49.32	29.64	21.34	14.22
Avg.	45.90	25.61	17.48	12.12

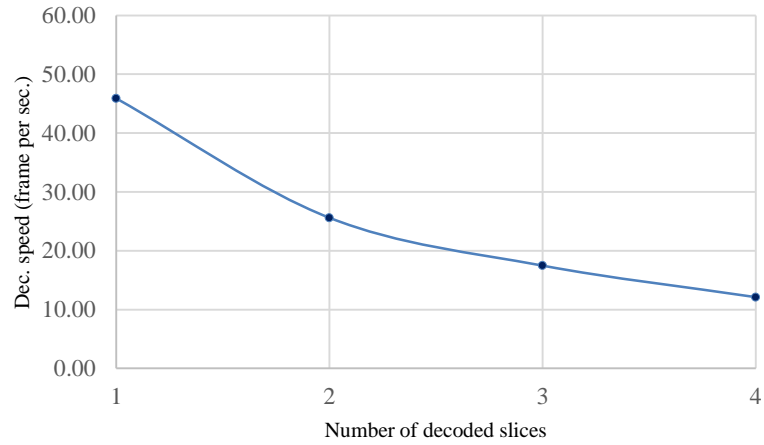


Figure 23. Decoding speed in AVC (frame per sec.)

Table 14. Decoding speed in HEVC (frame per second)

QP	1 tile	2 tiles	3 tiles	Whole frame
22	64.04	30.04	19.81	13.18
26	77.44	37.98	25.09	16.77
30	84.73	42.06	28.20	18.61
34	87.34	44.59	29.34	19.75
Avg.	78.39	38.67	25.61	17.08

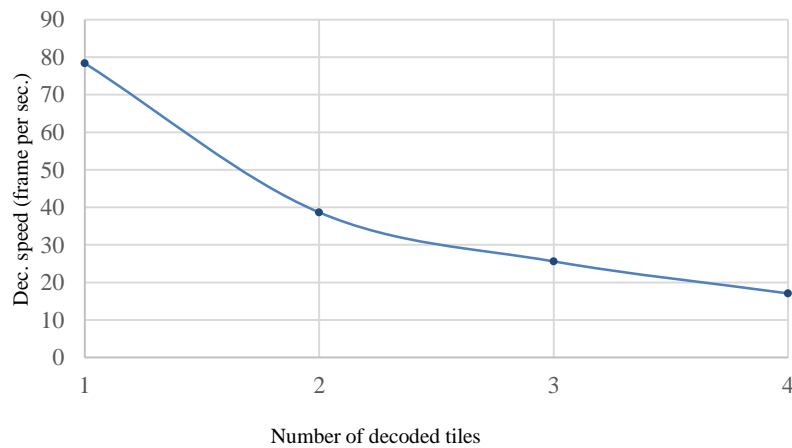


Figure 24. Decoding speed in HEVC (frame per sec.)

5. CONCLUSIONS

This thesis has provided an elegant solution for enabling partial decoding in both AVC and HEVC standards in applications where only a region of the video pictures is decoded. It has elaborated the creation of independently decodable partitions, so-called self-contained partitions, in the AVC and HEVC bitstreams using slices and tiles, respectively. The thesis has formulated the requirements that have to be imposed on the AVC standard JM and HEVC standard HTM encoders in order to enable self-contained partitions. More emphasis was given to the HEVC part since HEVC decoders are already widely available in a variety of devices.

The performance of the proposed self-contained slices/tiles was evaluated in the context of light-field 3D displays, in which partial decoding helps in enabling real-time operation. Naturally, the proposed approach is not limited to the light-field 3D displays. It can be utilized in any applications (e.g. ROI applications) where video frames can be reasonably split into separate partitions and only a sequence of the partitions need to be decoded.

The creation of self-contained tiles in the 3D extensions of HEVC (i.e., MV-HEVC and 3D-HEVC) was also described. Since the MV-HEVC block-level coding tools are the same as those of conventional HEVC, no additional restrictions are required in order to enable partial decoding than those described in the case of HEVC. However, in 3D-HEVC, there is one more restriction to be fulfilled for enabling partial decoding of video pictures. The implementation of self-contained tiles in 3D-HEVC is left for future work.

The results indicate that the usage of partitioning tools and the proposed encoder modifications yield a small negligible compression penalty. However, the proposed approach makes the decoding operation very efficient and low complex by reducing computational burden of decoding. It significantly increases decoding speed and thus helps enabling real-time processing. This allows to realize the proposed approach in real-world applications using hardware decoders available today in the market.

6. REFERENCES

- [1] H. Schwarz, T. Schierl and D. Marpe, "Block structures and parallelism features in HEVC," in *High Efficiency Video Coding (HEVC): Algorithms and Architectures*, Springer, 2014, pp. 49-90.
- [2] "Holografika, Holographic display technology," Holografika, [Online]. Available: <http://www.holografika.com/>. [Accessed 2016].
- [3] G. Jason, "Three-dimensional display technologies," *Advances in optics and photonics*, vol. 5, no. 4, pp. 456-535, 2013.
- [4] F.-C. Huang, G. Wetzstein, B. A. Barsky and R. Raskar, "Eyeglasses-free display: towards correcting visual aberrations with computational light field displays," *ACM Transaction on Graphics*, 2014.
- [5] G. Wetzstein, D. Lanman, M. Hirsch, W. Heidrich and R. Raskar, "Compressive light field displays," *IEEE Computer Graphics and Applications*, vol. 32, no. 5, pp. 6-11, 2012.
- [6] T. Balogh, P. T. Kovacs and A. Barsi, "Holovizio 3D display system," in *proc. IMMERSCOM 2007*, 2007.
- [7] R. Bregovic, P. T. Kovács, T. Balogh and A. Gotchev, "Display-specific light-field analysis," *Proc. SPIE-DSS: Three-Dimensional Imaging, Visualization, and Display*, 2014.
- [8] P. T. Kovacs, Z. Nagy, A. Barsi, V. K. Adhikarla and R. Bregovic, "Overview of the applicability of H.264/MVC for real-time light-field applications," in *proc. 3DTV-CON 2014*, Budapest, Hungary, 2014.
- [9] H. M. M., Y.-K. Wang and M. Gabbouj, "Isolated regions in video coding," *IEEE Transactions on Multimedia*, vol. 6, p. 259–267, 2004.

- [10] P. Lambert, D. D. Schrijver, D. V. Deursen and W. D. Neve, "A real-time content adaptation framework for exploiting ROI scalability in H.264/AVC," in *ACIVS'06 Proceedings of the 8th international conference on Advanced Concepts For Intelligent Vision Systems*, Springer-Verlag Berlin, 2006.
- [11] P. Quax, F. D. Fiore, P. Issaris, W. Lamotte and F. V. Reeth, "Practical and scalable transmission of segmented video sequences to multiple players using H.264," in *Motion in Games 2009 (MIG09), Lecture Notes in Computer Science LNC, LNCS 5884*, 2009, pp. 256-267.
- [12] A. Zare, P. T. Kovács and G. Atanas, "Self-contained slices in H.264 for partial video decoding targeting 3D light-field displays," in *Proc. 3DTV Conference 2015*, Lisbon, 2015.
- [13] A. Zare, A. Aminlou, M. M. Hannuksela and M. Gabbouj, "HEVC-compliant tile-based streaming of panoramic video for virtual reality applications," in *proceedings of ACM MM 2016*, Amsterdam, 2016.
- [14] C. Feldmann, C. Bulla and B. Cellarius, "Efficient stream-reassembling for video conferencing applications using tiles in HEVC," in *MMEDIA 2013, The Fifth International Conferences on Advances in Multimedia*, Venice, Italy, 2013.
- [15] J. Vehkaperä and S. Tomperi, "Replacing picture regions in H.264/AVC bitstream by utilizing independent slices," in *IEEE 17th International Conference on Image Processing*, Hong Kong, 2010.
- [16] K. M. Misra, C. A. Segall, M. Horowitz, S. Xu, A. Fuldseth and M. Zhou, "An overview of tiles," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 969-977, Dec. 2013.
- [17] P. Amon, M. Sapre and A. Hutter, "Compressed domain stitching of HEVC streams for video conferencing applications," in *Proc. of 2012 IEEE 19th International Packet Video Workshop*, Munich, 2012.
- [18] "An interactive region-of-interest video streaming system for online lecture viewing," in *2010 18th IEEE International Packet Video Workshop*, Hong Kong, 2010.

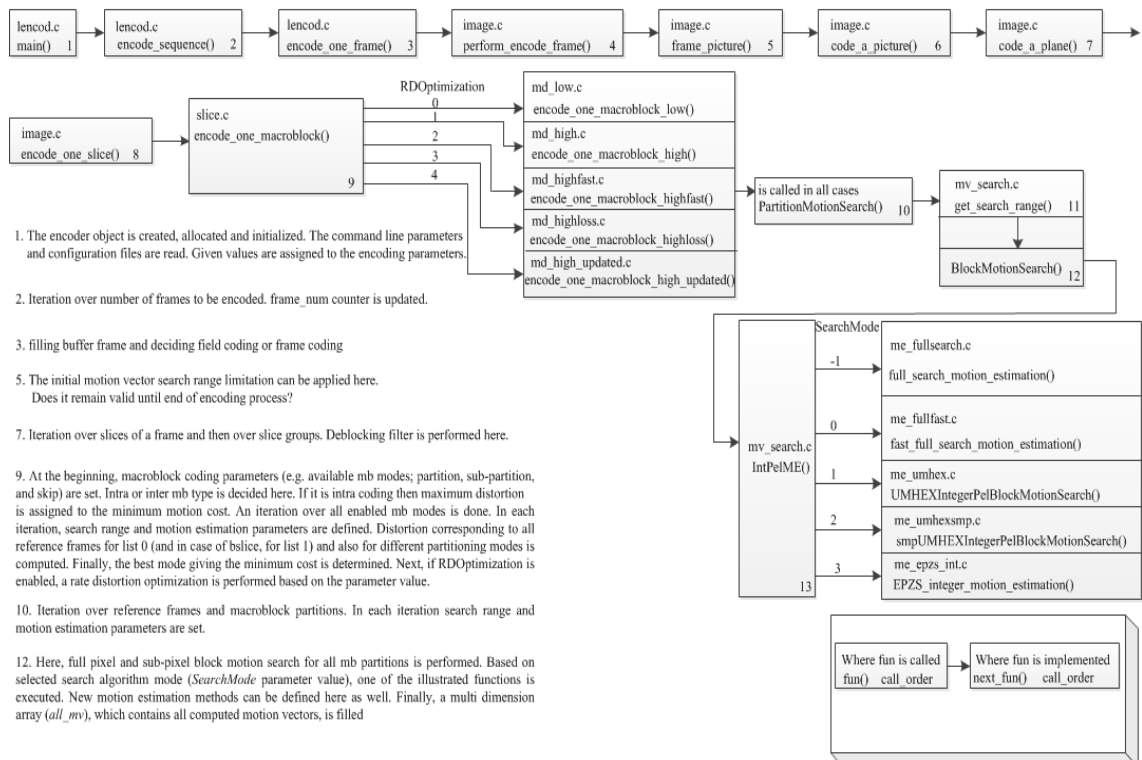
- [19] P. Quax, P. Issaris, W. Vanmontfort and W. Lamotte, "Evaluation of distribution of panoramic video sequences in the eXplorative television project," in *in Proc. 22nd International Workshop on Network and Operating System Support for Digital Audio and Video*, Toronto, 2012.
- [20] "FFmpeg Codecs Documentation," FFmpeg, [Online]. Available: <https://ffmpeg.org/ffmpeg-codecs.html>. [Accessed 2016].
- [21] Y. Sanchez, R. Globisch, T. Schierl and T. Wiegand, "Low Complexity Cloud-video-Mixing Using HEVC," in *IEEE CCNC - Multimedia Networking, Services and Applications*, Las Vegas, NV, USA, 2014.
- [22] I. E. G. Richardson, *H.264 and MPEG-4 video compression, video coding for next-generation multimedia*, Chichester, UK: John Wiley & Sons Ltd, 2003.
- [23] "Committed to connecting the world," ITU-T, [Online]. Available: <http://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jvt.aspx>. [Accessed 26 August 2015].
- [24] M. Wien, *High Efficiency Video Coding-coding tools and specification*, Springer-Verlag Berlin Heidelberg, 2015.
- [25] "ITU-T SG16: Multimedia," ITU-T, [Online]. Available: <http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/default.aspx>. [Accessed 29 9 2015].
- [26] F. H. H. Institute, "High Efficiency Video Coding (HEVC)," [Online]. Available: <https://hevc.hhi.fraunhofer.de/>. [Accessed 29 9 2015].
- [27] H. Schwarz, D. Marpe and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 17, no. 9, pp. 1103-1120, 2007.
- [28] J. a. Y. Y. a. C. J. a. R. A. Boyce, "Overview of SHVC: scalable extensions of the High Efficiency Video Coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1-14, 2015.

- [29] D. Marpe, T. Wiegand and S. Gordon, "H.264/MPEG4-AVC fidelity range extensions: tools, profiles, performance, and application areas," in *IEEE International Conference on Image Processing, ICIP*, 2005.
- [30] G. J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, C. A. Segall and A. Vetro, "Standardized extensions of High Efficiency Video Coding (HEVC)," *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING*, vol. 7, no. 6, pp. 1001-1016, 2013.
- [31] P. Merkle, A. Smolic', K. Müller and T. Wiegand, "Efficient prediction structures for multiview video coding," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 17, no. 11, pp. 1461-1473, 2007.
- [32] Y. Chen, G. Tech, K. Wegner and S. Yea, "Test Model 11 of 3D-HEVC and MV-HEVC," JCT-3V and Video Subgroup, Geneva, Switzerland, February, 2015.
- [33] V. Sze, M. Budagavi and G. J. Sullivan, High Efficiency Video Coding (HEVC)-algorithms and architectures, Springer International Publishing, 2014.
- [34] "Solveig multimedia-zond 265 3.0-HEVC video analyzer," 2003. [Online]. Available: <http://www.solveigmm.com>. [Accessed 2015].
- [35] I.-K. Kim, J. Min, T. Lee, W.-J. Han and J. Park, "Block partitioning structure in the HEVC standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, pp. 1697-1706, 2012.
- [36] G. J. Sullivan, J.-R. Ohm, W.-J. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, pp. 1649-1667, 2012.
- [37] J. Lainema, F. Bossen, W.-J. Han, J. Min and K. Ugur, "Intra coding of the HEVC standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, p. 1792–1801, 2012.
- [38] G. Laroche, J. Jung and B. Pesquet-Popescu, "RD optimized coding for motion vector predictor selection," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 18, no. 9, pp. 1247-1257, 2008.

- [39] P. Helle, S. Oudin, B. Bross, D. Marpe, M. O. Bici, K. Ugur, J. Jung, G. Clare and T. Wiegand, "Block merging for quadtree-based partitioning in HEVC," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, pp. 1720-1731, 2012.
- [40] M. M. Hannuksela, Y. Yan, X. Huang and H. Li, "Overview of the multiview high efficiency video coding (MV-hevc) standard," in *2015 IEEE International Conference on Image Processing (ICIP)*, Quebec City, 2015.
- [41] "Codecian Co. Ltd - CodecVisa analyzer 4.32," 2015. [Online]. Available: <http://www.codecian.com>.
- [42] "H.264/AVC Reference Software," [Online]. Available: <http://iphome.hhi.de/suehring/tml/download/>. [Accessed 2016].
- [43] G. Bjøntegard, "Calculation of average psnr differences between rd-curves," in *document VCEG-M33*, Austin, 2001.
- [44] T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 13, no. 7, pp. 560-576, July 2003.
- [45] M. Wien, *High Efficiency Video Coding-coding tools and specification*, Springer-Verlag Berlin Heidelberg, 2015.

APPENDIX A: H.264 TRACING PATH OVERVIEW

An overview of tracing path H264 Reference Software



APPENDIX B: DOWNLOAD LINKS

Software repository:

The latest version of the reference software for AVC (called JM, **J**oint **M**odel) can be downloaded from the link below:

<http://iphome.hhi.de/suehring/tml/download/>

The reference software for HEVC (called HM, **H**EVC **T**est **M**odel) is available in the following SVN repository:

https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/

The third version of the HEVC reference software (called HTM, **H**EVC **T**est **M**odel) including the MV and 3D extensions can be downloaded from the following SVN repository:

https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSsoftware/

For full understanding of the capabilities and features of the H.264/ AVC and H.265/ HEVC standards, a copy of the ITU/ ISO recommendations can be obtained from ITU web site for free from the links below:

<http://www.itu.int/rec/T-REC-H.264>

<http://www.itu.int/rec/T-REC-H.265>

Software compilation:

The H.264 and H.265 software can be built under linux using make. For Windows, solutions for different versions of Microsoft Visual Studio are provided. The user should select the appropriate solution according to his/her .NET package.

Meeting documents for JCT-VC and JCT-3V can be achieved from ITU web site:

<http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/default.aspx>

Software structure:

The H.264 solution contains the following four projects []:

lencod H.264/AVC: reference encoder

ldecod H.264/AVC: reference decoder

rtpdump: a tool for analyzing contents of RTP packets

rtp_loss: a tool for simulating RTP packet losses

The 3D-HEVC Test Model Software includes several applications and libraries for coding and view synthesis []:

Applications:

TAppEncoder: executable for bit stream generation

TAppDecoder: executable for reconstruction.

TAppRenderer: executable view synthesis

TAppExtractor: executable for bitstream extraction

Libraries:

TAppCommon: library for handling encoder, decoder and renderer options and camera parameters

TLibEncoder: encoding functionalities

TLibDecoder: decoding functionalities

TLibRenderer: renderer functionalities

TLibExtractor: bitstream extraction functionalities

TLibCommon: common functionalities

TLibVideoIO: video input/output functionalities

All the source codes and configuration files related to this work can be downloaded from the link below: