ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ

«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

ΥΛΟΠΟΙΗΣΗ ΕΡΓΑΛΕΙΟΥ ΑΝΑΛΥΣΗΣ HEVC (H.265) ΒΙΝΤΕΟ

Δημόπουλος Γρηγόριος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπουσα
Κοζύρη Μαρία,
Επίκουρος Καθηγήτρια στο
Τμήμα Πληροφορικής
Πανεπιστημίου Θεσσαλίας

Λαμία, 2017

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Implementation of HEVC (H.265) Video Analysis Tool» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ

Ημερομηνία
5/12/2017

Υπογραφή
[redacted]

IMPLEMENTATION OF HEVC (H.265) VIDEO ANALYSIS TOOL

Dimopoulos Grigorios

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

From 1990 to the present, the development of video coding standards is rapidly evolving by providing competitive compression ratios and supporting higher resolutions (e.g. 4K, 8K and 10-bit depth). Nowadays one of the most promising video coding standard is High Efficiency Video Coding (HEVC/H.265) [1], which provides increased compression ratio without sacrificing video quality. However, this comes at the cost of increased coding complexity, which leads to corresponding increment in HEVC video encoding and decoding time. In order to cope with this problem HEVC standard supports three high level parallelization tools [23]: slices, tiles and wavefront parallelism.

Purpose of this master thesis is the development of an application in Microsoft Windows environment, which will not only decode and reproduce an H.265 coded video file, but also visualize the use of tiles and/or slices in the coded sequence. For this reason, the implementation of application named “HEVC Video Analysis Tool” provides the following services:

- The use of the HEVC decoder to decompress a binary video file into a YUV video file.
- Reproduction of the decoded video file via a video display box. The user may handle the reproduction of the video using appropriate buttons for basic actions such as play or stop the video, go to the first or last frame, etc.
- Adjustment of the size of video display box accordingly the user's preferences.
- The partitioning of the video frames into slices and tiles depending on partition parameters that has been defined in encoding procedure. User is able to turn on/off the slice and tile partitioning by clicking the appropriate keys.
- The view of Coding Tree Unit (CTU) grid and the drawing of horizontal and vertical ruler beside video display box for measuring the CTUs columns and rows, respectively.

For the development of the “HEVC Video Analysis Tool” software application, it was considered necessary to contribute other programs and tools. These components are the following:

- HEVC encoder and decoder and the HM reference software version 16.15 [9] [12], which have been developed by the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11.
- Microsoft Visual Studio 2015 [14] for designing and programming the main graphical user interface (GUI) application for Microsoft Windows.
- OpenCV [15] open source computer vision library for reading YUV images inside the code files.
- C [19], C++ [20] and C++/CLI [21] (Visual C++) programming languages for creating the program.

The rest of the master thesis is organized as follows: Chapter 2 contains a brief description of the history of video coding standards through the years. Also, it analyzes the basic concepts of the video coding process. Chapter 3 introduces the basic features of HEVC and highlights the differences with its predecessor, H.264. Moreover an overview of the HM reference software and the configuration file of encoder are also presented. Chapter 4 provides useful information about slices and tiles, and instructions for using slice and tile parameters in configuration file. Chapter 5 has a detailed description of the development the tool and demonstrates the application. Finally, Chapter 6 summarizes and concludes this thesis.

CHAPTER 2: VIDEO CODING

From the very first moment that video came into the lives of people, they presented some issues that were needed to be resolved. In detail, an uncompressed video consumes a large space in storage, bandwidth and transmission time. After this, they found a way to store and transmit a huge amount of video data by compressing them. The result that come from this process is to reduce the resource utilization. Video Encoding or Video Compression is an application of data compression and its objective is to remove redundant information from a video and to omit those parts of the video that will not be noticed by Human Visual System (HVS). In order to reconstruct the compressed video to the original video follows the decompression process. A Codec consist of the Compressor (Encoder) and Decompressor (Decoder).

History

Let's see now the evolution of video coding over the past few years. Some examples of this evolution are:

- Analog television converted to Digital television.
- VHS video tapes improved to DVDs.
- Cell phones that used only for making calls and sending text messages transformed into smart phones with plenty of functions, such as camera, social network, web browser etc. and barely used to make calls.

The continuous evolution of digital video industry is driven by commercial factors and technological advances. The commercial drive comes from the huge revenue potential of persuading consumers and businesses. In the technology field, the factors include better communications infrastructure, cheap broadband networks, 4G mobile networks and the development of easy-to-use applications for recording, editing, sharing and viewing videos.

Video Coding Standards

In video signal, there is a relation between neighboring frames in the video sequence and between neighboring blocks of data in each frame. The methods intra-frame coding and inter-frame coding can compress the required data without affecting the video quality. So, it has been developed video coding standards [3] [4] such as H.261 [16] [16][18], MPEG-1 [16], H.262 [17], MPEG-2 [17], H.263 [18], H.264 and MPEG-4 [2] [7], H.265/HEVC) [1] [6] [8] [13] based on the following elements:

- The redundant information of each frame. Disintegration of each frame is usually done with transform.
- The redundant information between consecutive frames. The exploitation can be done by encoding their differences.
- The repetition of produced symbols. Entropy encoding can reduce the number of bits needed for encoding a symbol.

Figure 1 shows the video coding standards that were developed by two main Standardization Organizations, ITU-T and ISO over the last few years. The International Organization for Standardization (ISO) is an international standard-setting body composed of representatives from various national standards organizations and its purpose is to promote worldwide proprietary, industrial and commercial standards. Regarding video coding such standards were MPEG1, MPEG2 and MPEG4, which were developed by the Moving Picture Experts Group (MPEG). On the other hand, the ITU Telecommunication Standardization Sector (ITU-T) is one of the three sectors of the International Telecommunication Union (ITU). It aims at standards for telecommunications. The .26x standards used for video telephony were developed by ITU-T.

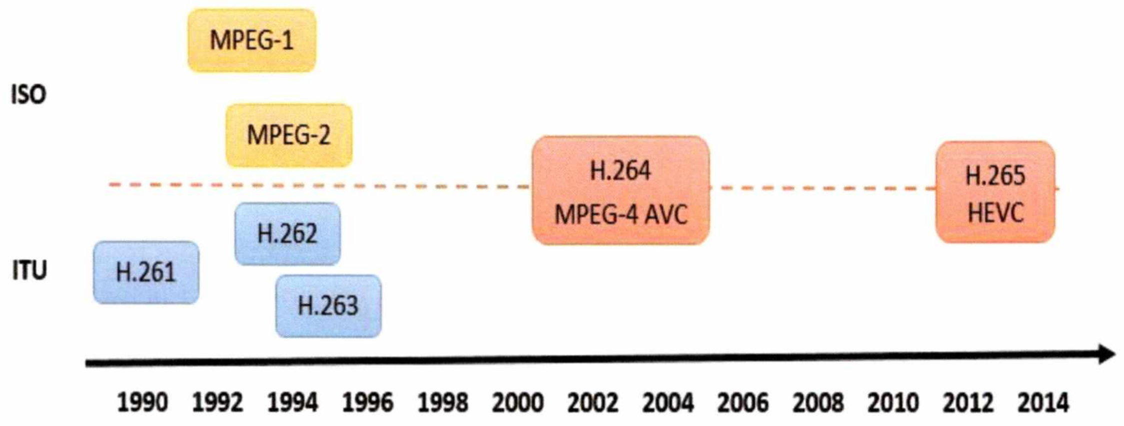


Figure 1. Chronology of video coding standards

The majority of video coding standards are based on block based hybrid video coding technique. Each block is either intra-coded or inter-coded. Furthermore, video compression standards have similar basic structure, but the coding efficiency has a considerable improvement from one generation to the next. Let's analyze some of the video standards that were important in video coding:

1) **H.261** [16], [16][18] was the first accepted compression standard that was defined by ITU in 1990. It was used for video conferencing and video telephony over ISDN. It also supports two picture formats: CIF (Common Intermediate Format) and QCIF (Quarter of Common Intermediate Format). The sequence of coding process is prediction, block transformation, quantization and entropy encoding. It also runs a

motion compensation algorithm in order to encode the differences between neighbor frames and exploit redundancy.

2) **MPEG-1** [16] is the first video compression standard by ISO, released in 1993. MPEG-1 has some improvements in comparison to H.261 standard and it also reaches coding at a rate of about 1.5 Mbps. The purpose of this standard is storage and retrieval of audio and video on a digital storage media. MPEG-1 has the advantage of achieving better quality (i.e. reducing the noise) by using a bi-directional prediction. On the other hand, MPEG-1's complexity is higher than that of H.261.
3) **H.262** and **MPEG-2** [17] were developed by ISO and ITU together and released in 1994. MPEG-2 has some benefits in comparison to MPEG-1, such as wider motion compensation and support for interlaced videos. It also supports both interlaced and progressive videos and has a method that uses wider search range in high resolution videos, for instance, digital TV and DVD.
4) **H.263** [18] is developed by ITU in 1995 and has extra features from H.261 such as Unrestricted Motion Vector Mode, Half-Pixel Motion Estimation, Advanced Prediction Mode and 3-D Variable Length Coding of DCT coefficients. H.263 supports the following picture formats: QCIF, Sub-QCIF, CIF, 4-CIF and 16-CIF. H.263 standard is used on video telephony over PSTN. Their methods, Conversational High Compression Profile (CHC) and High Latency Profile (HLP), were significantly improved in relation to H.261 standard and have higher coding efficiency.
5) **H.264** and **MPEG-4 AVC** [2], [7] jointly developed by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) in 2003. The remarkable improvements of H.264/MPEG-4 standard are: i) intra coding and inter coding efficiency, ii) enhanced error robustness and increased flexibility, iii) efficient motion compensation and reduced bit-rate. H.264/MPEG-4 AVC also takes a 16x16 pixels Macroblock as a unit during the process. It uses Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC) for entropy encoding. H.264/MPEG-4 AVC was adopted in numerous applications such as HDTV broadcasting, internet video and video conference. Figures 2 and 3 show the block diagram of an H.264 encoder and an H.264 decoder, respectively.
6) **H.265/HEVC** (High Efficiency Video Coding) [1], [6], [8], [13] jointly developed by Joint Collaborative Team on Video Coding (JCT-VC) of ITU and ISO/IEC in 2013. It's the best standard nowadays, because it provides many new features in comparison of previous standards, such as larger coding block structures (64x64 pixels), quadtree syntax of block structures, advanced motion vector prediction, more intra prediction directional modes. Additional details about HEVC standard will follow in the next chapter.

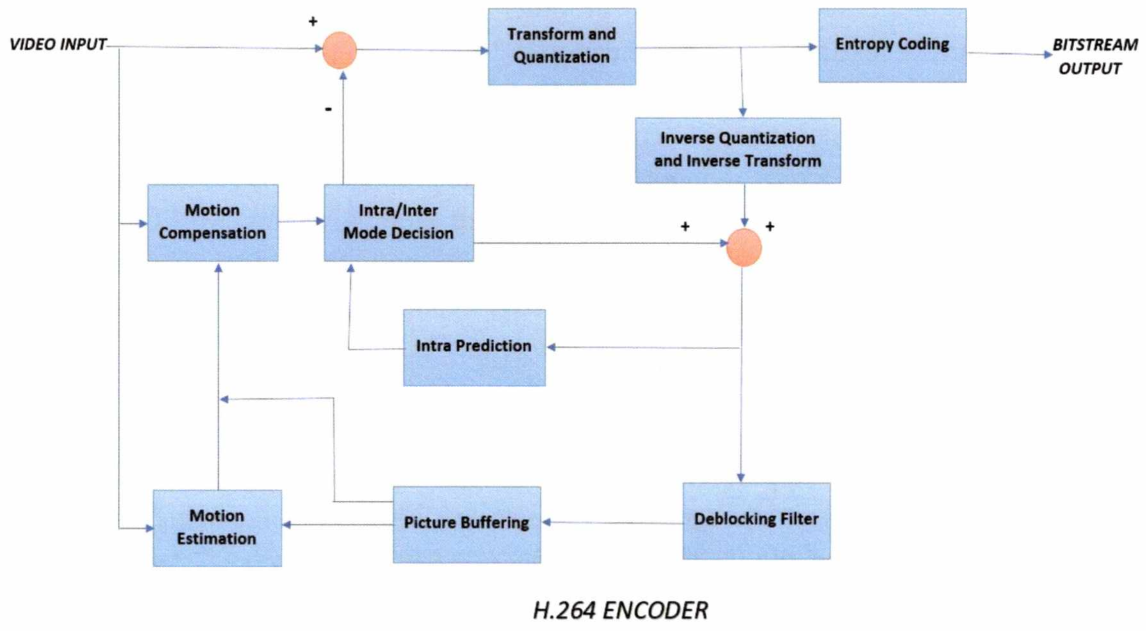


Figure 2. H.264 Encoder Block-Based Diagram

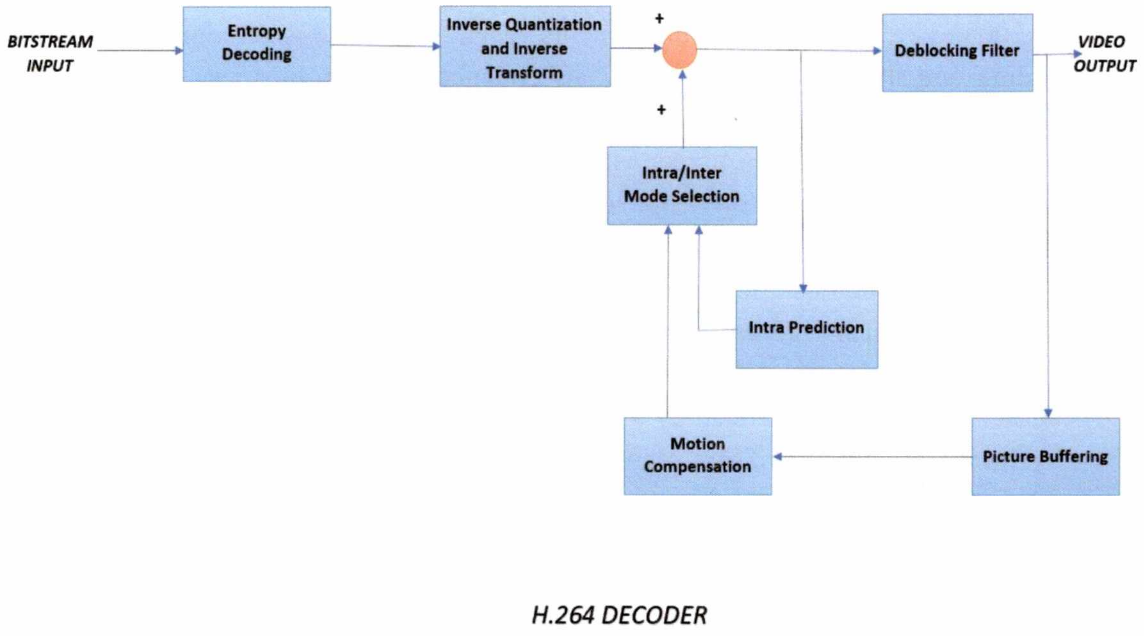


Figure 3. H.264 Decoder Block-Based Diagram

Video Compression Basics

This subchapter will highlight the basic concepts of video coding process [5] [8]. In order to compress a video, it is necessary first to remove the redundant information from a video sequence. The basic types of redundancies in video sequence are:

1) Spatial Redundancy

- One way to reduce spatial redundancy in the video, is by using Intra Prediction Method. In Intra Prediction the prediction for the current block is calculated from the neighboring blocks of pixels, within the current picture, that have already been encoded. Each video standard supports different intra prediction modes. These modes choose different directions to create the prediction block, such as vertical prediction, horizontal prediction, DC prediction, Plane Prediction and angular prediction. Also, prediction mode can partitioned a block of picture into smaller number of prediction blocks with various sizes.
- Another way to reduce spatial redundancy in a video is the block transform. The operation of transform in the encoder is to convert frame or residual signal, resulting from inter or intra picture prediction, into the transform domain. The picture and the residual signal of a picture is divided into square blocks with same dimensions NxN (width and height). The data of transform domain should be separated into minimum dependency components and into high energy components. These high energy components are gathered in adjacent places within the block. Also, transform can be reversed by following a set of calculations, which form the Inverse Transform. There are two types of transformations: the block-based and image-based transform. The most popular method in block-based transform is Discrete Cosine Transform (DCT) and in image-based transform is Discrete Wavelet Transform (DWT). A negative of the image-based transform is that requires high memory, because it takes the whole image or tile to make transform calculations.

2) Perceptual Redundancy

- Perpetual redundancy is a method that is applied in the significant video data. The process of Perceptual redundancy makes a use of Human Visual System (HVS), a system where humans perceive and interpret visual images. Human Visual System is more sensitive to low sensitive information than high sensitive information. Also, in YUV format is more sensitive to luminance (Y) than to chrominance (U or C_b and V or C_r) components.
- The main tool to remove perceptual redundancy is Quantization. After the transform stage, the encoder continues and sends the resulting transform coefficients to the quantizer. The quantizer receives the transform coefficients, which are divided by quantization step (Qstep) and then are rounded, and converts them to quantized transform coefficients. As the range of values is smaller, the resulted quantized signal has fewer bits than the original. On the other hand, the decoder follows the

inverse quantization, where the quantized transformed coefficients are multiplied by Qstep and are converted into original transform coefficients. After that, the inverse transform is applied to de-quantized transform coefficients in order to produce a residual block that needs intra or inter prediction samples to obtain reconstructed block. There are two types of quantizations: the scalar quantization and the vector quantization. In scalar quantization, it takes one input sample and convert it to a quantized output value. In vector quantization, it takes a group of input samples and convert it to a group of quantized output values.

3) Statistical Redundancy

- Entropy coding is the last method that is performed in video encoder (and the first method in video decoder) and its work is to reduce the statistical redundancy. The entropy encoder is a lossless data compression scheme that uses all statistical properties from previous executed methods in order to compress data. At first, it assigns a unique prefix-free code to each unique symbol and send it to the input. Then entropy encoder compresses data by replacing these unique input coded symbols with the corresponding variable-length prefix-free output codeword. The bits of each codeword are proportional to the negative logarithm of the probability. For example, the most common codewords are represented by a small number of bits, in comparison to the most uncommon codewords, which are represented by many bits. Shannon's coding theory tells that the optimal average code length for a symbol with probability p is $-log_b p$, where b is the number of symbols used to represent output codes, in this case compressed data is created by bits 0 and 1, and p is the probability of the input symbol. Some of the known entropy modes are the Variable-Length Codes (VLCs), the Context-Adaptive Binary Arithmetic Coding (CABAC) and the Context-Adaptive Variable Length Coding (CAVLC).

4) Temporal Redundancy

- Inter prediction is the method that a video encoder uses in order to remove the temporal redundancy. Specifically, inter prediction uses reference pictures, which have been previously coded (encoded and decoded) and stored in a decoded picture buffer. Motion estimation and motion compensation are used to find the best candidate block in the reference frame and create the difference block. Motion vectors are calculated by finding the relative displacement between the position of the current picture block and the position of reference picture block. The steps of inter prediction are the following: i) divide the picture into blocks and apply motion estimation and compensation to each block, ii) for each block search the relative motion between current picture block and reference picture block, iii) create and transmit the motion vector of each block.
- There is an extra method that reduces temporal redundancy and comes of the frame difference coding. The processes, which are involved in the computation of difference between two adjacent frames, are the Discreet Cosine Transform, the Quantization and the Entropy Coding.

CHAPTER 3: HEVC AND REFERENCE SOFTWARE

In 2003, two expert groups of video coding, ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG), joined together and cooperated for the purpose of developing the H.264 codec [2]. In 2010, MPEG and VCEG expert groups had the same challenge and decided to establish a team called Joint Collaborative Team on Video Coding JCT-VC. The JCT-VC team aimed at developing H.265/HEVC (High Efficiency Video Coding) [1]. The first edition of HEVC was finalized in January 2013 and it was officially defined in April 2013. Furthermore, JCT-VC is extended the first edition in order to support many additional application cases, such as an extended-range uses with enhanced precision and color format support, scalable video coding [27], screen content coding [28] and 3D, stereo, multiview video coding [29]. So, in ISO/IEC, the HEVC standard will become MPEG-H Part 2 (ISO/IEC 23008-2) and in ITU-T it will possibly become ITU-T Recommendation H.265.

The establishment of HEVC has a strong impact in global society and launched a wide variety of products that are increasingly prevalent in our daily lives. However, development teams continues the effort to reach higher compression capability and make efficient some characteristics such as data loss robustness, while considering the available computational resources. Moreover, the increased demand for higher quality and resolutions in mobile applications cause this thing necessary. HEVC has been designed to support sufficiently all existing applications of H.264/MPEG-4 AVC and to focus mainly on two key issues: increased video resolution and increased use of parallel processing architectures.

To assist the industry community, the standard is necessary to have not only a text of specification document, but also the reference software source code which serves as an example of the way that a HEVC video can be encoded and decoded. The reference software can be used as a research tool during the design of the standard but also as a general research tool in the field of video coding.

The HEVC standard was developed so as to achieve multiple goals. Those goals include higher coding efficiency, easier transport system integration and data loss resilience, and use of parallel processing tools for more efficient implementation. The following subchapter briefly describes how these goals can be succeeded and the benefits of using H.265 standard instead of H.264.

Features of HEVC Standard

HEVC has so many new features compared to H.264/MPEG-4 AVC [3], [5], [6], [8]. A block-based hybrid coding is used in HEVC and it offers the following features:

I. HEVC uses Coding Tree Units (CTUs) and Coding Tree Blocks (CTBs) structures. The Coding Tree Unit (CTU), which has a size selected by the encoder and can be larger than a traditional macroblock (16x16). CTUs can have size 64x64, 32x32, or 16x16 pixels in a picture (Figure 4). A CTU consists of a luminance component (Luma CTB - Coding Tree Block), the corresponding chroma components (C_b and C_r CTB) and syntax elements (Figure 5). Furthermore, HEVC provides a tree structure and quadtree-like method that partitioning CTBs into smaller blocks.

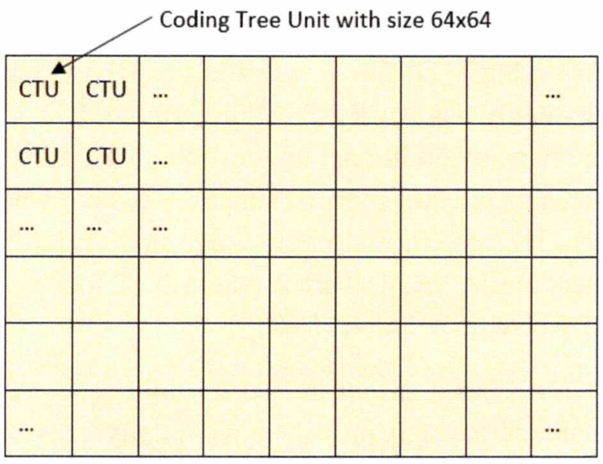


Figure 4. CTUs in a Frame

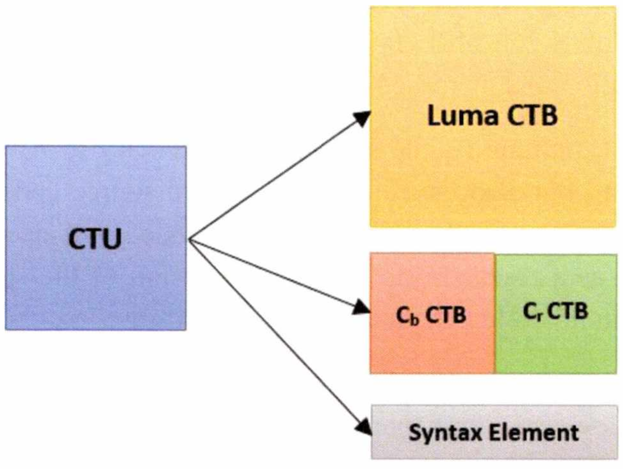


Figure 5. CTU Partitioning

II. HEVC supports a quadtree syntax of CTU that uses Coding Units (CUs) and Coding Blocks (CBs) structures. Firstly, CTU specifies the size and positions of its luma and chroma CBs. It starts with the root of the quadtree, which is the CTU. Then the root CTU splits into luma and chroma CBs at the same time. A Coding Unit consists of one luma CB and ordinarily two chroma CBs, together with syntax element (Figure 6). A CTB could be split into one CU or multiple CUs, and each CU has an associated partitioning into Prediction Units (PUs) and a tree of Transform Units (TUs).

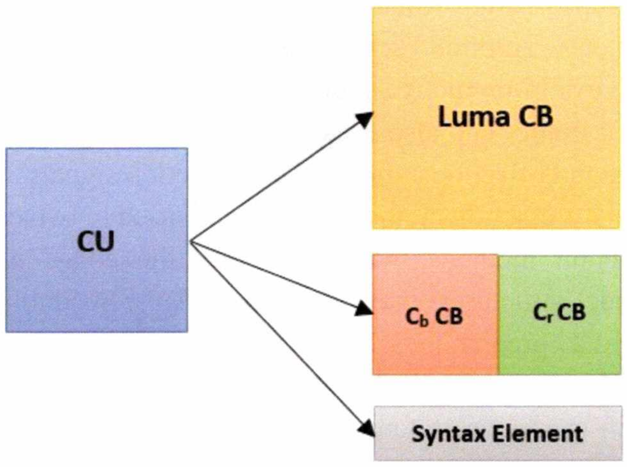


Figure 6. CU Partitioning

III. Prediction Units (PUs) and Prediction Blocks (PBs) are parts of CU that provide the information whether to code a frame area using inter or intra prediction. The root of a PU structure is the CU. The luma and chroma CBs can be used in inter prediction and split into the luma and chroma PBs. A prediction block (PB) is a block of samples of luma or chroma component that uses the same motion parameters for motion-compensated prediction. PB can take sizes from 64x64 to 4X4 pixels. Intra-coded CUs can only be divided into square PUs with same width and height. Inter-coded CUs can be divided into square or non-square PUs as long as one side has at least 4 pixels. It also provides 2 modes of partitioning intra-coded CUs and 8 methods of partitioning inter-coded CUs (Figure 7).

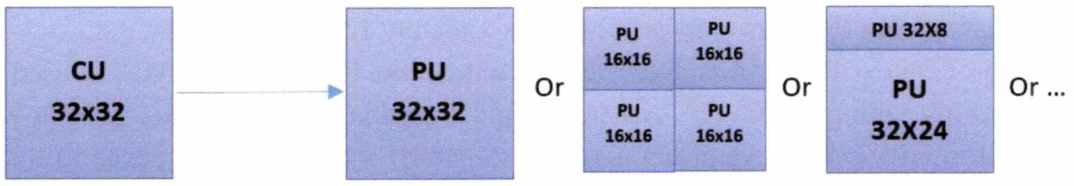


Figure 7. Partitioning modes for splitting a 32x32 inter-coded CU into PUs

IV. Transform Units (TUs) and Transform Blocks (TBs) provide the information about the transform applied to residual signal. The root of a TU structure is also the CU. The luma and chroma CBs residual can be split into a luma and chroma TBs or smaller luma and chroma TBs, respectively. Either integer basis functions or Discrete Cosine Transform (DCT) can be defined for the square TB sizes 4×4, 8×8, 16×16, and 32×32.

V. HEVC uses Advanced Motion Vector Prediction (AMVP) that is based in information about adjacent PBs and the reference picture. There is also a Merge

Mode (MV) that gives the ability to inherit temporally or spatially neighboring PBs to Motion Vectors (MVs).

VI. In Motion compensation, quarter-sample precision is used for the Motion Vectors (MVs) and 7-tap or 8-tap filters are used for interpolation of fractional-sample positions. One or two motion vectors can be transmitted from each PB and resulting in uni-predictive or bi-predictive coding. Like in H.264/MPEG-4 AVC, HEVC uses many reference frames and also uses a way called weighted prediction in order to apply a scaling and offset operation to the prediction signals.

VII. The HEVC intra prediction methods can be classified in two categories. Firstly, the angular prediction methods, which provides the codec with a possibility to accurately model structures with directional edges. Secondly, the namely planar prediction and DC prediction, which provide predictors estimating smooth image content. HEVC supports totally 35 intra prediction directional modes (Table I). All intra prediction modes use reference samples from the neighboring reconstructed blocks and also the intra prediction is applied in transform blocks with size ranging from 4x4 to 32x32 samples.

Table I. Intra Prediction modes and Associated Methods in HEVC

INTRA PREDICTION MODE NUMBER	ASSOCIATED PREDICTION METHODS
0	Intra Planar Prediction
1	Intra DC Prediction
2 - 34	Intra Angular Prediction[i], i=2 . . . 34

VIII. Uniform reconstruction quantization (URQ) is used in HEVC. The scales of quantization matrices support various transform block sizes. The quantization parameter (QP) is used to calculate exact Qstep value and can take 52 values from 0 to 51 for 8-bit video sequences. Every time the QP increased by one, then the quantization step size increased by approximately 12%.

IX. The Context-Based Adaptive Binary Arithmetic Coding (CABAC) is used for entropy coding in HEVC. CABAC scheme is used in H.264/MPEG-4 AVC as well, but in HEVC has several improvements. The HEVC CABAC achieved to have a better throughput speed and compression performance, and to reduce its context memory requirements. Entropy coding is a lossless compression method of syntax elements, which uses the statistical properties to compress data such that the number of bits used to represent the data is logarithmically proportional to the probability of the data. For example, during the compression of data symbols, the frequent symbols represented by a few bits, while the infrequent symbols represented by many bits.

X. The HEVC specifies two in-loop filters, a deblocking filter and a Sample Adaptive Offset (SAO). The in-loop filters run in the encoding and decoding loops, after the inverse quantization and before saving the picture in the decoded picture buffer. The deblocking filter is executed first and decreases discontinuities at the prediction and transform block boundaries. The Sample Adaptive Offset (SAO) in-loop filter is applied to the output of the deblocking filter and improves the quality of the decoded picture. This happens by diluting sound components and by modifying some intense areas of a picture. Using in-loop filters achieves improved quality of reconstructed pictures and during the decoding has the advantage to increase the quality of the reference pictures, as well as the compression efficiency.

Figures 8 and 9 illustrate the basic operations of HEVC encoder and decoder, respectively, and show all the steps followed during the encoding and decoding procedure.

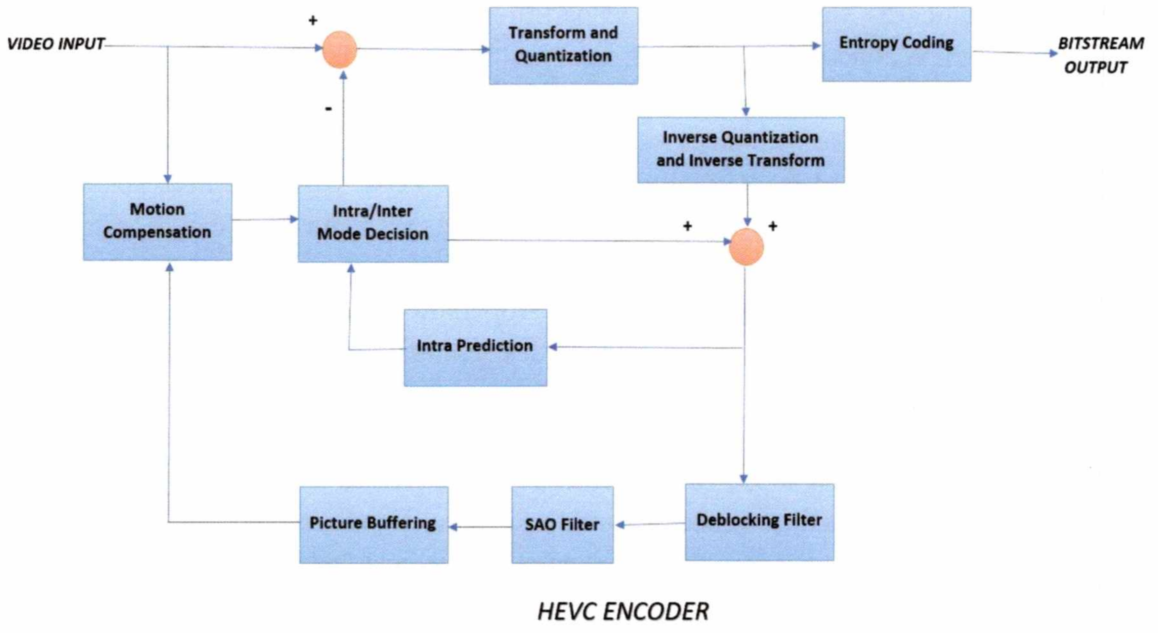


Figure 8. HEVC Encoder Block-Based Diagram

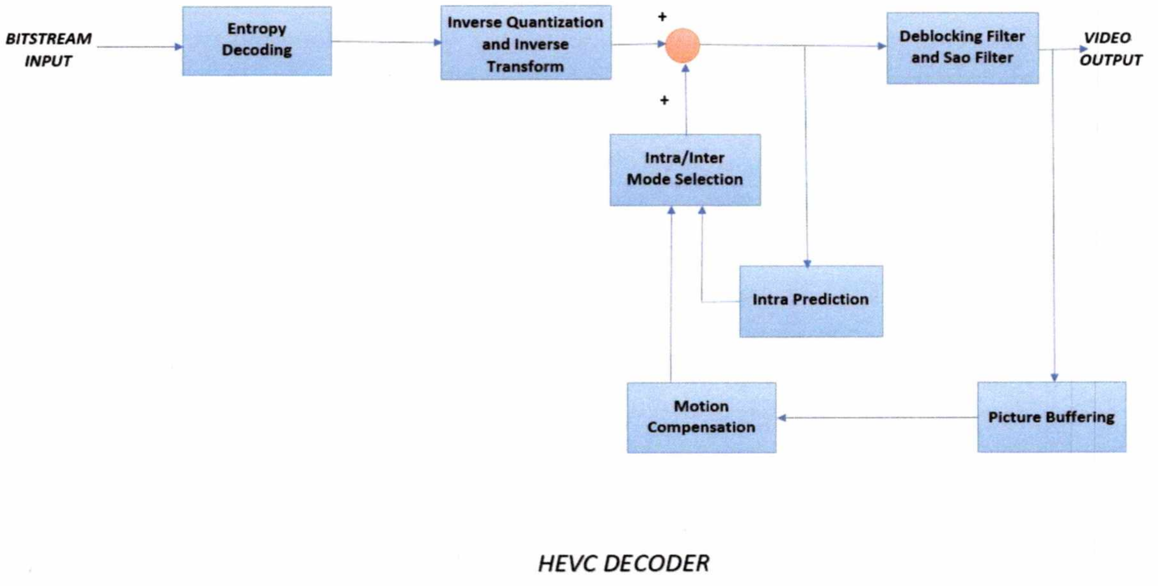


Figure 9. HEVC Decoder Block-Based Diagram

Comparison between H.264/AVC and HEVC

HEVC was designed for the purpose to support all existing applications of H.264/AVC and to focus on the solution of two issues: to support higher resolutions and to improve parallel processing architectures. Developers of JCT-VC team achieved to improve the HEVC standard in many spots and transform it into an innovative and more flexible product in comparison with the previous standard H.264/AVC. The main differences that are located between HEVC and H.264/AVC [3] [6] [7] [8] are the following:

- ❖ Both standards support block size partitioning, but there are many differences among them concerning partitions sizes and flexibility. In H.264/AVC, each frame is divided into Macro Blocks (MBs) with size 16x16 pixels. Each MB can be further splitted into blocks with sizes varying down to 4x4. On the other hand, in HEVC each frame is divided into Coding Tree Units (CTUs) with maximum size 64x64 pixels. Different structures within a CTU, such as PUs and TUs, can be splitted to blocks with sizes varying down to 4x4.
- ❖ HEVC supports 33 directional modes for Intra Prediction, plus Intra Planar Prediction and Intra DC Prediction modes, compared to 9 directional modes in H.264/AVC.
- ❖ HEVC uses a similar deblocking filter as in H.264/AVC, which is operated within the inter prediction loop. However, HEVC design has the advantages to make decisions easier, to filter processes and to run parallel processing faster. Deblocking filter is performed on every 4x4 block edge of MB in H.264 and on every 8x8 block edge of CTU in HEVC.
- ❖ In order to locate the compression gains of HEVC standard, developers made some tests using human viewers and objective metrics such as Peak Signal to Noise Ratio (PSNR). Those computations shown up about 40%-50% bit-rate reduction in HEVC compared to H.264/AVC standard, for similar video quality. A similar test shown up that H.264/AVC has 50% bit-rate reduction compared to H.263.
- ❖ Context Adaptive Binary Arithmetic Coding (CABAC) and Context-Adaptive Variable-Length Coding (CAVLC) are used for entropy coding in H.264, while only Context Adaptive Binary Arithmetic Coding (CABAC) is used for entropy coding in HEVC. CABAC and Low-Complexity Entropy Coding (LCEC) as a follow-up of CAVLC were parts of HEVC. Later, during the standardization processes, the complexity of LCEC was found higher than CABAC and the compression efficiency of LCEC had to be increased. So, CABAC became the single coding method of HEVC, which has improved the throughput speed and the compression efficiency.
- ❖ HEVC standard introduced a predictive coding for motion vectors named Advanced Motion Vector Prediction (AMVP). AMVP uses the decoder to produce the best prediction in each motion block in relation to Motion Vector Prediction (MVP) in H.264/AVC. HEVC had been designed with a new technique called inter prediction block merging, where derives all motion data of a block from the adjacent blocks,

by replacing direct and skip modes of Motion Vector Prediction (MVP) in H.264/AVC.

- For Motion Compensation the H.264/AVC standard uses 6-tap filtering of half-sample position followed by linear interpolation for quarter sample position, in contrast to HEVC standard that uses 7-tap or 8-tap filters for interpolation of fractional sample position.
- H.264/AVC standard was not designed for Ultra High Definitions (UHD) and supports up to 4K (4,096×2,304) resolution and 59.94 fps only, in contrast to HEVC standard, which can support UHD videos with up to 8K (8192×4320) resolution and 300 fps.

Reference Software of HEVC Encoder and Decoder

In this chapter, it is presented an overview of the reference software of HEVC/H.265 encoder and decoder that has been developed by the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 [12]. HM Reference Software Manual is composed of the implementation of an HEVC encoder and an HEVC decoder, along with supplementary tools, documentation and configuration files [9]. The main purpose of the reference software is to implement all aspects specified in the standard and provide a complete environment for research implementations and experiments. Therefore, the application wasn't developed to be particularly efficient in terms of execution time and memory usage.

The software provides the following list of available project files that have been developed in different environments:

- MS Visual Studio 2008 (VC9)
- MS Visual Studio 2010 (VC10)
- MS Visual Studio 2012 (VC11)
- MS Visual Studio 2013 (VC12)
- Xcode
- Eclipse
- make/gcc (e.g. Ubuntu, Linux)

The user may run the encoder by opening the command prompt [24] in Microsoft Windows operating system and then typing the following command at the proper root directory of the encoder:

```
TAppEncoder [--help] [-c config.cfg] [--parameter=value]
```

The option `--help` shows the parameter usage of the encoder. The option `-c` defines configuration file that the user want to import. It also gives the opportunity to put multiple configuration files by repeating `-c` option. The option `--parameter=value` determines the parameters that is going to be applied during the encoding. However, parameters values that defined on the command line have higher priority from parameters that have been set in the configuration file. So, if both command line and configuration file are setting the same parameter, the command line parameter will be used.

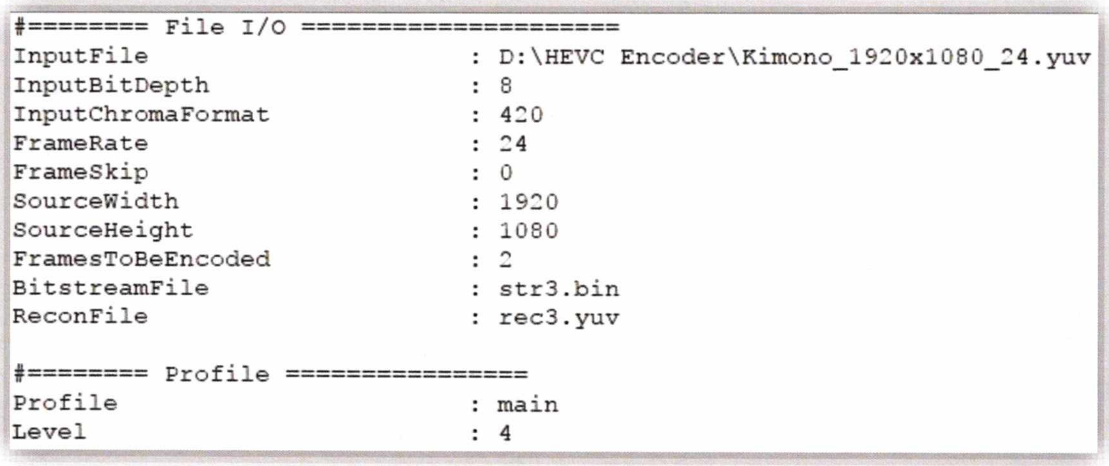
The user may execute the decoder, after getting the compressed binary file from the encoder, by following the same steps and typing the command:

```
TAppDecoder -b str.bin -o dec.yuv [options]
```

The option `-b` following by `str.bin`, which is the proper binary encoded file name that user have to set. The option `-o` following by `dec.yuv`, which is the reconstructed video file name that user have to define and export it to the root directory (otherwise it have to type the full path of the proper directory). The part `[options]` determines the parameters values of the decoder. Like the encoder parameters, the command line parameters settings, which are the same with parameters settings in the configuration file, will be used only.

The `cfg/` folder included in HM Reference Software, includes samples of configuration files.

The configuration file is developed to facilitate the user to define video information and parameters that will be used by the software during encoding [9]. Figure 10 includes information about input/output of the file and the profile. User can set basic features of the video, such as to define the full path of input file that will be imported to the encoder and to define the full path of bitstream and reconstructed file that will be exported by the encoder. Other things that can be determined are the input video width, height and frame rate per second, the number of frames that will be encoded or will be skipped, the input bit depth and the input ratio of luminance to chrominance samples that can take value 420 for a 4:2:0 format.

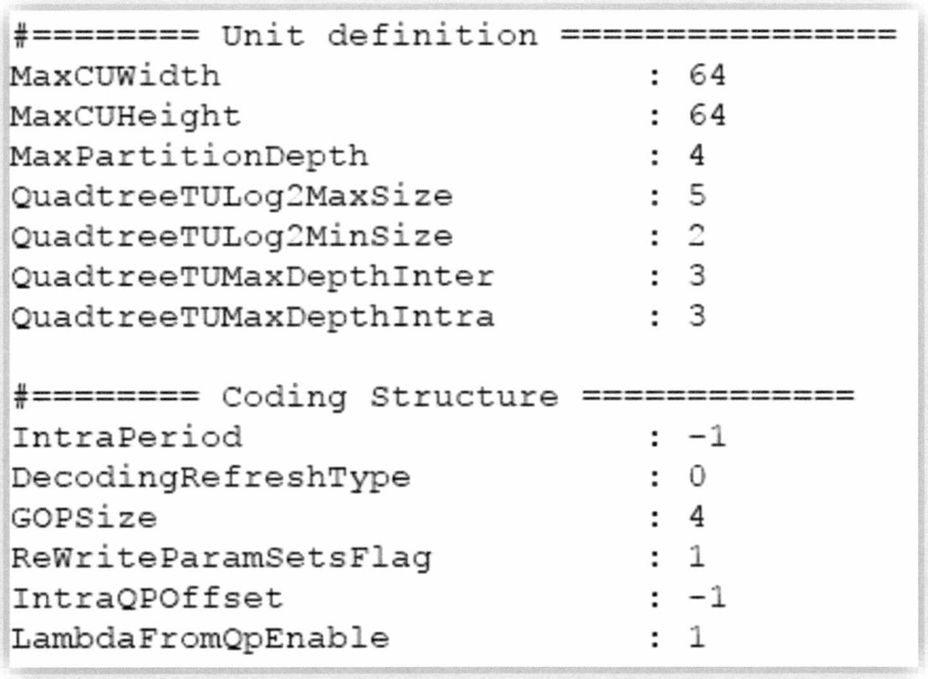


```
#======== File I/O =====================
InputFile                     : D:\HEVC Encoder\Kimono_1920x1080_24.yuv
InputBitDepth                 : 8
InputChromaFormat             : 420
FrameRate                     : 24
FrameSkip                     : 0
SourceWidth                   : 1920
SourceHeight                  : 1080
FramesToBeEncoded             : 2
BitstreamFile                 : str3.bin
ReconFile                     : rec3.yuv

#======== Profile ================
Profile                       : main
Level                         : 4
```

Figure 10. Input/Output file and Profile properties

In unit definition property, the user assigns the maximum size and depth of CU in pixels and also the maximum/minimum size and inter/intra depth of each quadtree-based TU. The group of pictures size (number of B slice), the period of intra frame (-1 sets only first), the quantization parameter of intra mode, the decoding refresh parameter (0 for none, 1 for Clean Random Access - CRA, 2 for Instantaneous Decoding Refresh - IDR, 3 for Recovery Point Supplemental Enhancement Information - SEI) etc., can be adjusted from Coding Structure section (Figure 11).

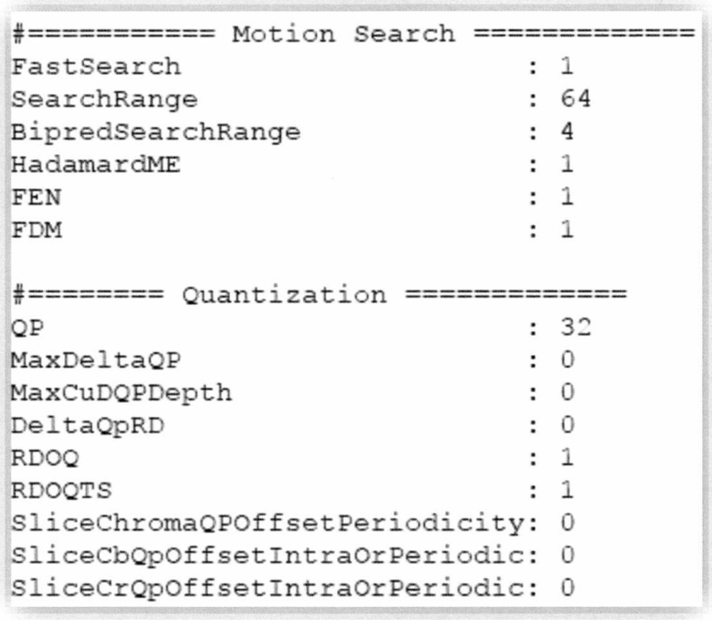


```
#======== Unit definition ================
MaxCUWidth                       : 64
MaxCUHeight                      : 64
MaxPartitionDepth                : 4
QuadtreeTULog2MaxSize            : 5
QuadtreeTULog2MinSize            : 2
QuadtreeTUMaxDepthInter          : 3
QuadtreeTUMaxDepthIntra          : 3

#======== Coding Structure =============
IntraPeriod                      : -1
DecodingRefreshType              : 0
GOPSize                          : 4
ReWriteParamSetsFlag             : 1
IntraQPOffset                    : -1
LambdaFromQpEnable               : 1
```

Figure 11. Unit definition and Coding Structure settings

For the motion search adjustment, FastSearch can take value 0 for full search and 1 for TZ search motion estimation. The range of each search can be decided in SearchRange parameter (0 value sets full frame for search range). There are also other options that can set the search range of bi-prediction refinement, the use of hadamard measure (1 means true and 0 means false), Fast Encoder Decision (FED) and Fast Decision for Merge (FDM). The main parameters of the Quantization part are: i) Quantization parameter that takes values from 0 to 51, ii) MaxDeltaQP for CU-based multi-QP optimization, iii) MaxCuDQPDepth for defining maximum depth of a minimum CuDQP for sub-LCU-level (Largest Coding Unit) delta QP, iv) DeltaQpRD for sliced-based multi-QP optimization, v) RDOQ for Rate-Distortion Optimized Quantization, and vi) RDOQTS for Rate-Distortion Optimized Quantization Transform Skip. Operations of motion estimation and quantization are illustrated in Figure 12.

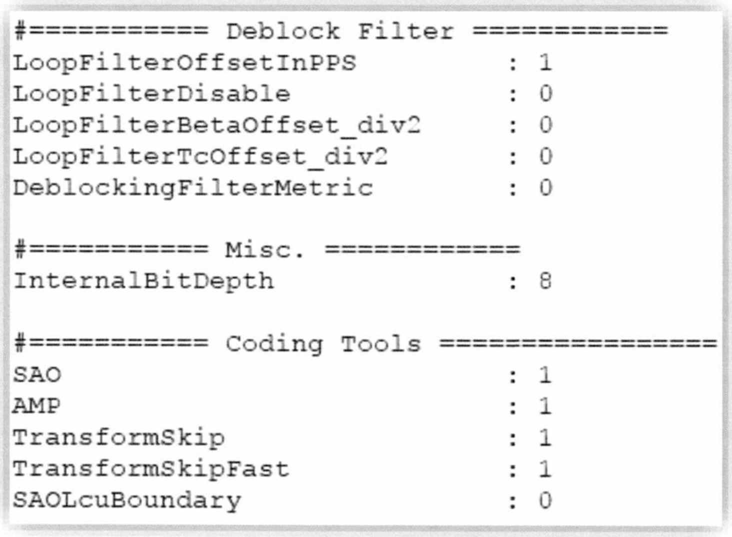


```
#=========== Motion Search =============
FastSearch                        : 1
SearchRange                       : 64
BipredSearchRange                 : 4
HadamardME                        : 1
FEN                               : 1
FDM                               : 1

#======== Quantization =============
QP                                : 32
MaxDeltaQP                        : 0
MaxCuDQPDepth                     : 0
DeltaQpRD                         : 0
RDOQ                              : 1
RDOQTS                            : 1
SliceChromaQPOffsetPeriodicity: 0
SliceCbQpOffsetIntraOrPeriodic: 0
SliceCrQpOffsetIntraOrPeriodic: 0
```

Figure 12. Motion Search and Quantization features

The next setting concerns Deblocking Filter, where the user may enable or disable deblocking filter (0 value enables filter, 1 value disables filter), may set a Loop Filter Offset in picture parameter set (PPS) by putting 0 value for varying parameters in SliceHeader and 1 value for constant parameters in picture parameters set and may use a deblocking filter metric, which automatically configures deblocking parameters in bitstream (LoopFilterOffsetInPPS and LoopFilterDisable must be 0). Miscellaneous has only one option to set the codec operating bit-depth. Coding tools provides options like sample adaptive offset, asymmetric motion partitions, transform skipping, fast transform skipping and SAOLcuBoundary using non-deblocked pixels, which can be defined enabled by using value 1 and disabled by using value 0 (Figure 13).



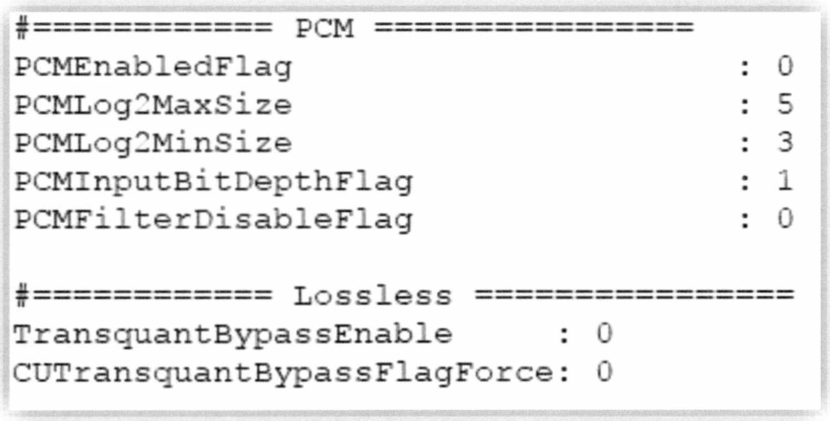
```
#=========== Deblock Filter ============
LoopFilterOffsetInPPS           : 1
LoopFilterDisable               : 0
LoopFilterBetaOffset_div2       : 0
LoopFilterTcOffset_div2         : 0
DeblockingFilterMetric          : 0

#=========== Misc. ============
InternalBitDepth                : 8

#=========== Coding Tools =================
SAO                             : 1
AMP                             : 1
TransformSkip                   : 1
TransformSkipFast               : 1
SAOLcuBoundary                  : 0
```

Figure 13. Deblock Filter, Miscellaneous and Coding Tools settings

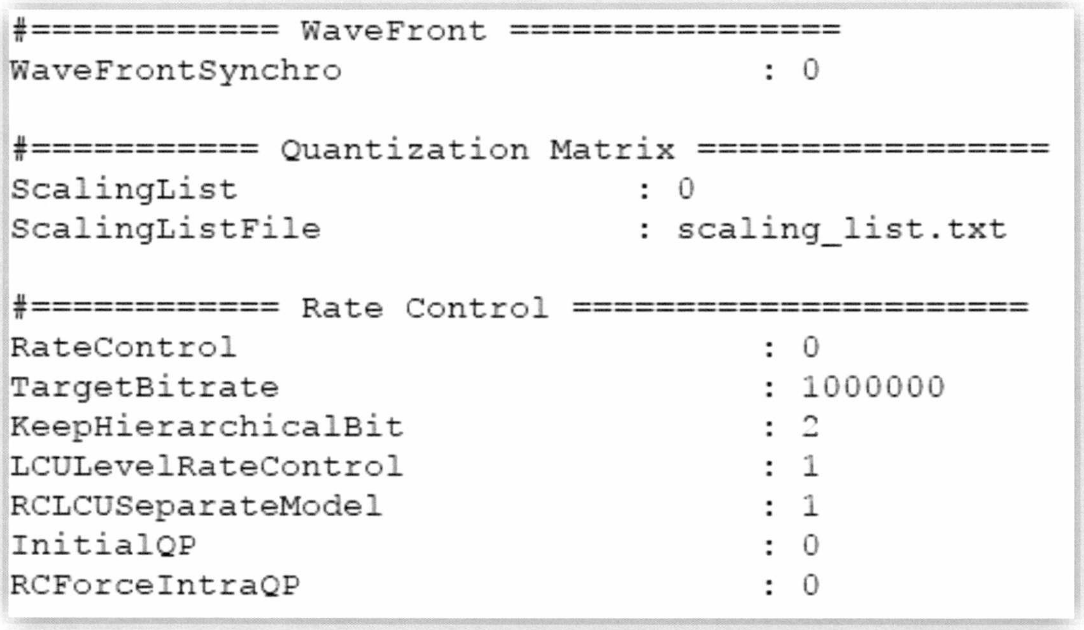
The application provides two special coding modes, the Pulse Coding Modulation (PCM) mode and the Lossless mode, which modify the transform and quantization process by either skipping the transform or by skipping both transform and quantization. In PCM part, the user may choose a PCM mode by assigning the PCMEnabledFlag parameter (0 for no PCM mode), may determine the maximum and minimum PCM block size, may set value 0 for internal bit-depth and value 1 for input bit depth at PCMInputBitDepthFlag parameter and may set 0 value for enabling and 1 value for disabling loop filtering on I_PCM samples. In Lossless part, the user may choose a PPS flag by assigning TransquantBypassEnable parameter and may force transquant bypass mode when transquant_bypass_enable_flag is enabled (Figure 14).


```
#============ PCM ================
PCMEnabledFlag                        : 0
PCMLog2MaxSize                        : 5
PCMLog2MinSize                        : 3
PCMInputBitDepthFlag                  : 1
PCMFilterDisableFlag                  : 0

#============ Lossless ================
TransquantBypassEnable      : 0
CUTransquantBypassFlagForce: 0
```

Figure 14. PCM and Lossless modes

Wavefront is a tool that is used for high-level parallel processing in HEVC standard. This method is used, when the frame is partitioned into CTU rows and each row waiting to take information about prediction and entropy coding of the previous row, except the first row. In configuration file, wavefront has only the parameter WaveFrontSynchro, where zero value gives no Wavefront synchronization and higher than zero values gives Wavefront synchronization with the Largest Coding Unit (LCU) above and to the right by this many LCUs. In the Quantization Matrix mode, the user can set ScalingList by typing 0 value for turning it off, 1 value for reading a default matrix and 2 value for reading user's chosen file. In order to read a specific file, the user have to type scaling list file name at ScalingListFile parameter. Furthermore, in Rate Control part can be adjusted the following parameters: i) enable rate control (0 turning it off and 1 turning it on), ii) set target bitrate in bps, iii) put 0 for equal bit allocation, 1 for fixed ratio bit allocation and 2 for adaptive ratio bit allocation in KeepHierarchicalBit parameter , iv) set 0 for picture level RC and 1 for LCU level in LCULevelRateControl, v) use LCU level separate R-lambda model in RCLCUSeparateModel, vi) set initial QP, and vii) force intra QP to be equal to initial QP (Figure 15).



```
#============ WaveFront ================
WaveFrontSynchro                          : 0

#=========== Quantization Matrix =================
ScalingList                         : 0
ScalingListFile                     : scaling_list.txt

#============ Rate Control ======================
RateControl                               : 0
TargetBitrate                             : 1000000
KeepHierarchicalBit                       : 2
LCULevelRateControl                       : 1
RCLCUSeparateModel                        : 1
InitialQP                                 : 0
RCForceIntraQP                            : 0
```

Figure 15. WaveFront, Quantization Matrix and Rate Control modes

Last but not least, the slices and tiles parameters are the most important parts of the configuration file for developing the software application and will be analyzed in the following chapter.

CHAPTER 4: SLICES & TILES

The necessity of executing many processes and calculations at the same time had the result of inventing parallel techniques. Parallel processing can take advantage of multiple computing resources in order to complete different tasks. MPEG-2 was the first standard in video coding, which applied parallel techniques in 1994. As time went on, the release of HEVC (H.265) standard made parallel methods evolve and become more flexible and speedy for satisfying present demands. There have been done plenty tests in HEVC, on how efficient is parallel implementations by using performance analysis for encoding and decoding [10] [11]. So, HEVC standard has defined three basic methods for high level parallelization, which are able to partition a frame into smaller parts and then encode and decode these parts independently [23]. These three methods are named slices, tiles and wavefront.

Wavefront parallel processing is a technique that is used in both encoder's and decoder's side. As known each frame is divided into CTUs. The process starts for each frame from the first row of CTUs, which is normally decoded, and then takes information about prediction and entropy coding from previous decoded rows in order to decode the rest rows of CTUs. The opportunity of searching and taking previous lines for crucial data can achieve better compression than slices and tiles method.

Parallelization exploits two important points of computer hardware, Graphics Processing Unit (GPU) and Central Processing Unit (CPU). On the one hand, GPUs manage the parallelization of motion estimation algorithm, which is used for the inter-picture prediction. On the other hand, CPUs offer a high level parallelization by using wavefront algorithms and group of pictures (GOP) based algorithms via multicore processing. Group of pictures structure is a group of consecutive frames, which have been already encoded, in a video stream and determines the order of intra and inter predicted frames in the sequence. Every time there is a new GOP, decoder has access to previous decoded frames that will need and thus is succeeded quicker searching through the video.

In next subchapter will be analyzed with a particular interest the main operations of slice and tile partitioning. Knowledge about these two meanings, is important for understanding the software application of master thesis.

Slices

HEVC has a tool that can partition coded frames into slices [25] in a similar way as MPEG-2 video standard. Briefly, slice is an independent piece of frame, which includes a sequence of CTUs and can be managed in the order of a raster scan. Frames can be separated into one or several slices depending on what encoding options and parallel process have been decided. Figure 16 shows the 6th frame from decoded video “Kimono” that has been partitioned into 6 same-sized slices with 85 CTUs each one (total frame CTUs is 510). Also, slices have the ability to be self-contained and independent from the other slices in the frame. This means that every slice can gather syntax elements data from bitstream encoded file and any other value that is contained in the slice area only, except for the data that arises from in-loop filter operations near the slices. So, all this collected information can normally decoded without using any data from other slices in the same frame. Another thing that cannot surpass slice boundaries and make calculations from neighboring slices is prediction procedure, such as intra-picture prediction and motion vector estimation. However, the in-loop filtering method is capable of having access into information beyond slice boundaries.

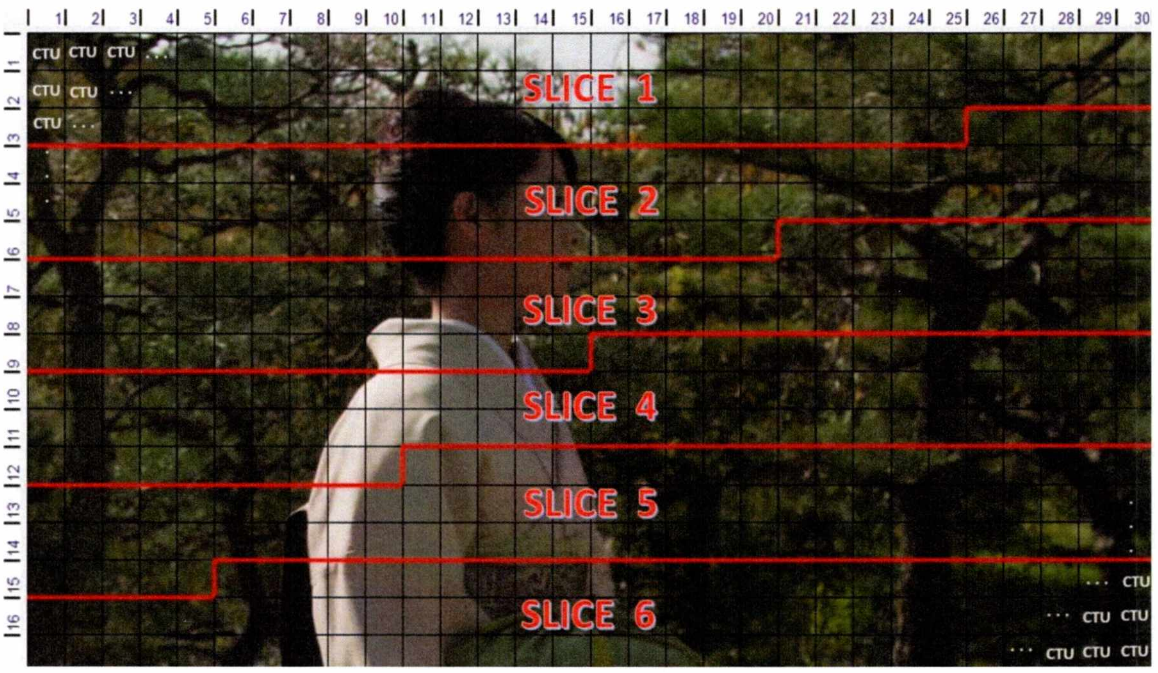


Figure 16. Frame with slice partitioning from decoded video “Kimono”

A slice can be partitioned into one or multiple slice segments, where the first slice doesn't have any relation and any connection with the rest slice segments and for that reason was called independent slice segment. However, the other slice segments in the sequence were called dependent slice segments, because of the connection that have with the previous slice segment (Figure 17). Each coded slice segment includes a header part and a data part. To be more specific, header slice segment has information about slice segment and data slice segment has information about coded samples. Independent slice segment header is the only one segment that carries information about all slice segments of the slice. Furthermore, each slice segment header consists of data for slice segments, such the slice segment address and the index of independent slice segment in the picture. The difference between first slice segment and the following slice segments is that only the first slice segment contains same useful parameters and additional information about slice type, output flag of the picture and reference parameter set (RPS). Slice type indicates the type that a slice has been coded [6] and are the following:

- I slice. It's a slice, which uses only intra-picture prediction in order to code all CUs of the slice segment.
- P slice. It's a slice, which has the same coding attributes of an I slice and also can be applied inter-picture prediction coding with maximum one motion compensation vector (uniprediction) per Prediction Block (PB) into several CUs of the slice segment. Generally, uniprediction process uses one picture from either reference picture list 0 or 1, one from each list. However, P slices are able to have access into reference picture list 0, so uniprediction can select a reference picture only from list 0.
- B slice. It's a slice, which has the same coding attributes of an I and a P slice, and also can be applied inter-picture prediction coding, as in P slice, but this time with maximum two motion compensation vectors (biprediction) per Prediction Block (PB) into some CUs of the slice segment. Generally, biprediction process uses two pictures from reference picture list 0 and 1, one from each list. Additionally, B slices are able to have access into both reference picture list 0 and 1.

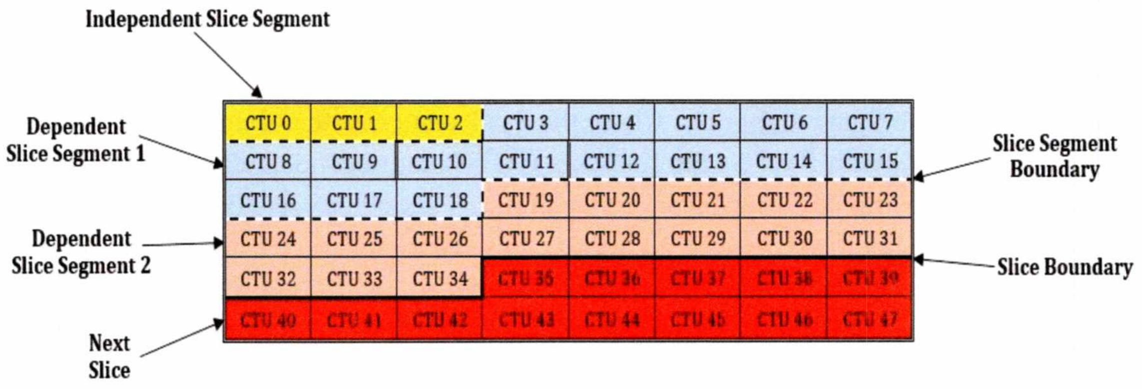


Figure 17. Slice Segment Partitioning

The partitioning of picture into slices offers the capability of decoding each slice independently, where the decoder uses entropy, residual and predictive decoding. A picture can be divided into one or several slices. In HEVC, the minimum number of CTUs that can be included in a slice is one.

The three main purposes of slice partitioning are the following:

❖ Error Robustness. Due to data losses, it was necessary to ensure the error robustness by partitioning the picture into smaller independent pieces. That gave the opportunity to re-synchronize both the decoding and parsing operation and helped dramatically to eliminate data losses. So, slice can be transmitted as a single packet, for example a loss of packet during the transmission equals to a loss of slice.

- Max Transmission Unit (MTU) Size Matching. IP networks has got some constraints in their communication channels, such as the Maximum Transmission Unit (MTU). MTU size matching, which follows a packetization scheme, forbids every slice to surpass the maximum number of bits regardless of the size of the coded frame. In order to comply with this rule each slice simultaneously minimize the packetization overhead, so the encoder can determine slices with varying sizes within a frame depending on the activity in the video scene.
- Parallel Processing. Slices of a partitioned picture have the capability of being processed in parallel. This is happening because all slice-based encoding and decoding methods except for loop filtering, can be independently executed in parallel [30], [31], [32].

Tiles

Another tool for partitioning a picture into a group of blocks in HEVC is tiles [26], [33], [34]. Similar to slices, tiles are rectangular-shaped areas of the picture, which are considered self-contained and can be decoded independently. Also, the rectangular-shaped area of each tile consists of CTUs and every tile is able to have different number of CTUs from others. Figure 18 illustrates the 3rd frame from decoded video "Kimono" that has been partitioned into 9 unevenly tiles, where tile has 3 same-width columns with 10 CTUs each one and 3 rows with 5 CTUs the first one, 6 CTUs the second and the third one. The activation of tile partitioning happens when the Picture Parameter Set (PPS) syntax element has the tile flag enabled. As it referred above, tiles were constructed for the purpose of encoding and decoding in parallel. By using slice and tile partitioning based on parallel processing architectures, they achieved to create more efficient encoders and decoders. One slice can contain multiple tiles, which will have the same header information. On the contrary, a single tile can contain multiple slices. However, tiles have the possibility of losing data during the transmission, because each tile of a picture is transported in a different packet.

Picture parameter set (PPS) provides a list of tile partitioning parameters, which can define the number of tiles and the size of each tile in a picture sequence or in a specific picture. The signaling parameters for slices and tiles will be explained at the next subchapter in detail. The use of PPS and the partitioning of picture into tiles have a major benefit that the encoder is capable of dividing each picture into different number of tiles each time considering the load balance between CPU cores. For instance, a region in a picture may need more processing resources from another region with more tiles, which requires less resources. All these resource allocations have to be determined before encoder starts processing the video.

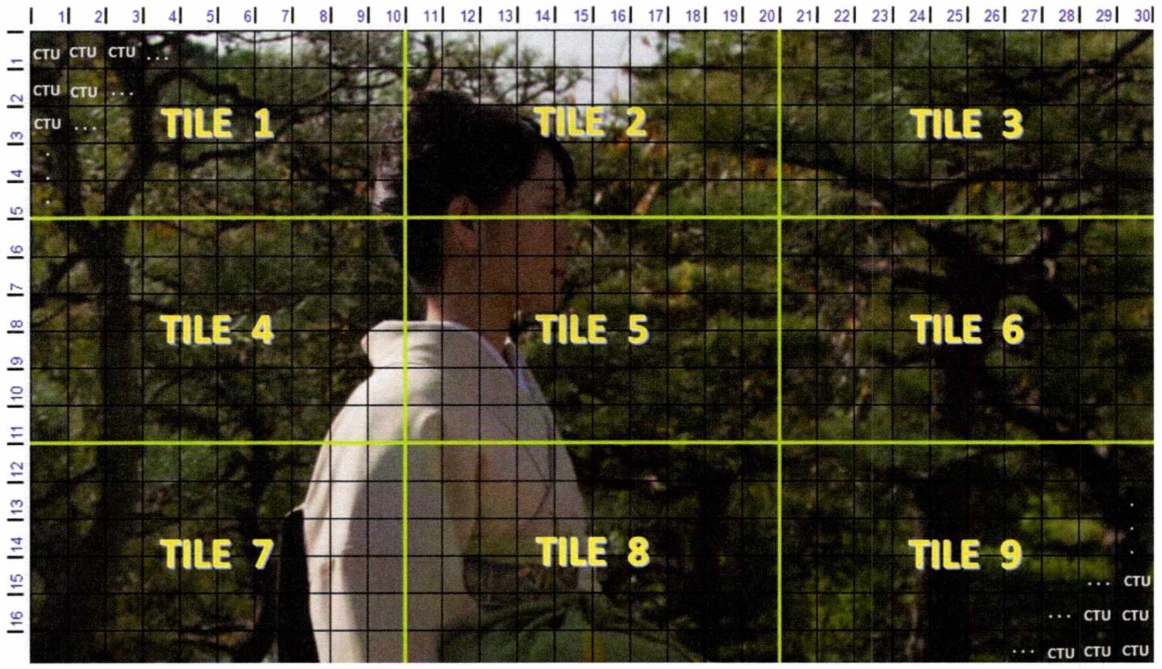


Figure 18. Frame with tile partitioning from decoded video "Kimono"

A difference between slice and tile mode, is that tile mode has a particular raster scan order. In tile partitioning, the raster scan modified from picture-based order to tile-based order. In picture-based, CTUs are processed in each row of a picture from left to right. In tile-based, CTUs are processed in each row of a tile from left to right, starting from the top left tile and moving to right tile after previous tile process completion. A negative of the decoder is that still using picture-based raster scan as a result of not exploiting parallel process.

It's possible, both tile and slice segments coexist in the same picture, so it have been taken some serious constraints about them [6]. The conditions that have to follow are:

1. Each slice and tile in a picture must at least follow one of the conditions: All CTUs in a slice belong to the same tile, or all CTUs in a tile belong to the same slice.

2. Each slice segment and tile in a picture must at least follow one of the conditions: All CTUs in a slice segment belong to the same tile, or all CTUs in a tile belong to the same slice segment.

3. A slice or a slice segment, which does not start at the same point of a tile, cannot expand into multiple tiles. Figure 19 illustrates an example of this condition, where it is observed that slice segments of the second slice don't expand into the second tile area.

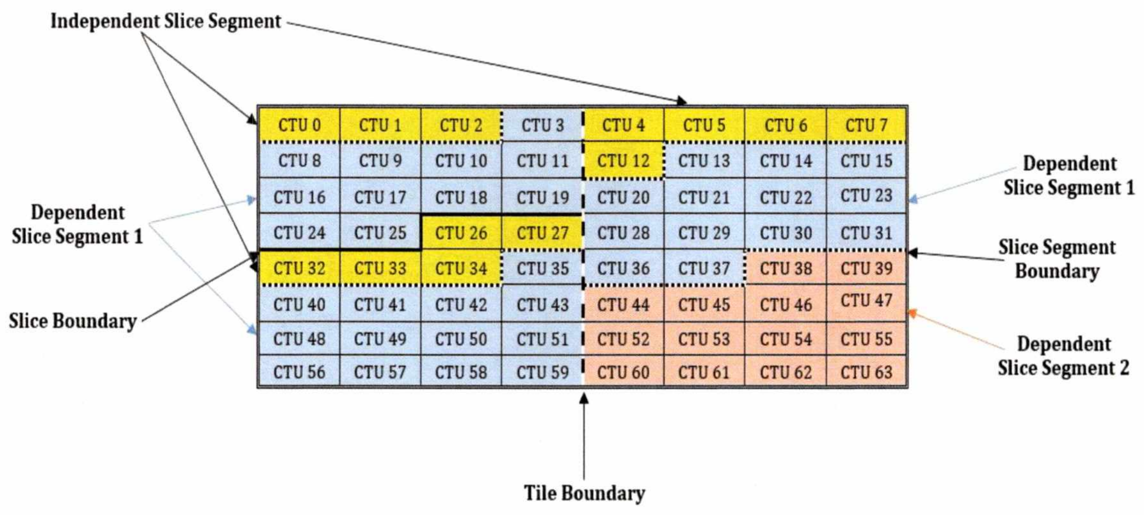


Figure 19. An example of condition 3 in tile and slice segment partitioning of a frame

Tiles have the capability of not having a contact with entropy decoding and reconstruction operations, but only communicate with in-loop filter in order to execute the cross tile-border filtering mode. Also, cross tile-border filtering mode has a possibility to produce visual artifacts at tile boundaries during the data exchange, so it is better this mode to be switched off.

The conclusion about slices and tiles is that tiles achieve better coding efficiency because they have the ability to reduce the spatial distance in tiles by exploiting spatial correlations between samples within a tile. Another benefit of tiles is that can reduce the slice header in case of the correspondence is slice per tile. As in slices, the increment of number of tiles in a frame has the consequence of getting coding efficiency loss, due to the breaking of dependencies along tile boundaries and the re-initialization of all CABAC context variables at the beginning of each tile.

Slices & Tiles in Reference Software

As it referred in the previous chapter, the configuration file of HEVC encoder [9] has two basics modes that will be used to verify the correctness of this project. The two modes are based in slice and tile partitioning of video frame. It offers the flexibility and the convenience to the user defining any combination of slices and tiles in the video.

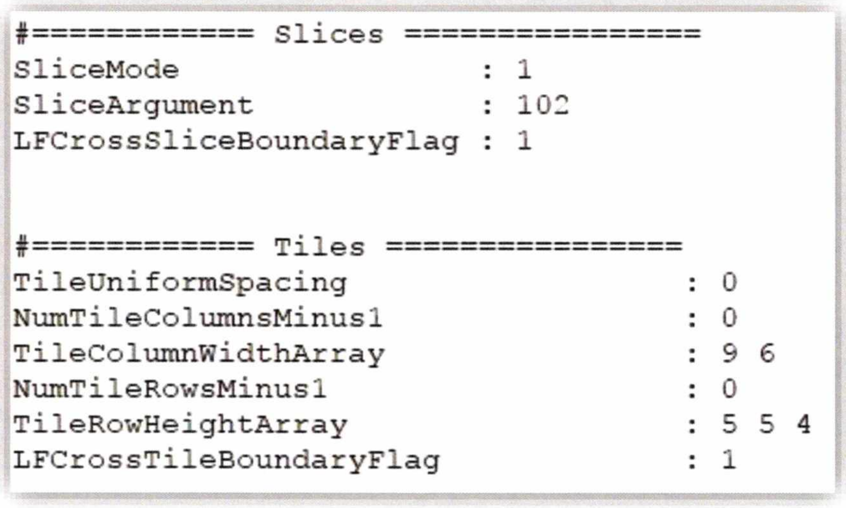
The parameters, which can be modified in slice mode, are described below (Figure 20):

- **SliceMode** defines whether the input video will be partitioned into slices or not. The values that can take are: 0 for turning all slice options off, 1 for setting a maximum number of Largest Coding Units (LCUs) in a slice, 2 for setting a maximum number of bytes in a slice, 3 for setting a maximum number of tiles in a slice. It is necessary to assign tile partitioning parameters, in order to take into account mode 3 of SliceMode and allocate the tiles to each slice.

- **SliceArgument** is an argument that is relative with SliceMode value. If SliceMode value is 0, nothing happens. If SliceMode value is 1, the user has to insert the maximum number of blocks that will include each slice. If SliceMode value is 2, the user has to insert the maximum number of bytes that will include each slice. If SliceMode value is 3, the user has to insert the maximum number of tiles that will include each slice.
- **LFCrossSliceBoundaryFlag** sets whether in-loop filter, like Adaptive Loop Filter (ALF) and Deblocking Filter, is across or not across the slice boundary. It takes value 0 for not across and value 1 for across.

The settings, which can be defined in tile mode, are described below (Figure 23):

- **TileUniformSpacing** can take two values: 0 and 1. When value is 0, then the column boundaries are assigned by TileColumnWidthArray and the row boundaries are assigned by TileRowHeightArray. When value is 1, then the column and row boundaries are assigned uniformly.
- **NumTileColumnsMinus1** can determine the number of tile columns in a frame minus 1. For instance, if a frame needed to be partitioned into 5 tile columns, NumTileColumnsMinus1 must have the value 4.
- **TileColumnWidthArray** defines an array that includes tile column width values in units of CTU starting from left to right in the frame. For example, if each frame has to be partitioned into 9 CTUs width the first column and 6 CTUs width the second column, TileColumnWidthArray must have the values 9 and 6 separated with a space. If there is a rest of CTUs in each frame width that hasn't set as tile column by TileColumnWidthArray parameter, encoder enforces these CTUs as the last tile column.
- **NumTileRowsMinus1** can set the number of tile rows in a frame minus 1. For instance, if a frame needed to be partitioned into 4 tile rows, NumTileRowsMinus1 must have the value 3.
- **TileRowHeightArray** set an array that includes tile row height values in units of CTU starting from top to bottom in frame. For example, if each frame has to be partitioned into 5 CTUs height the first row, 5 CTUs height the second row and 4 CTUs height the third row, TileRowHeightArray must have the values 5, 5 and 4 separated with a space. If there is a rest of CTUs in each frame height that hasn't defined as tile row by TileRowHeightArray parameter, encoder enforces these CTUs as the last tile row.
- **LFCrossTileBoundaryFlag** sets whether in-loop filter is across or not across the tile boundary. It takes value 0 for not across and value 1 for across.



```
#============ Slices ================
SliceMode                  : 1
SliceArgument              : 102
LFCrossSliceBoundaryFlag : 1


#============ Tiles ================
TileUniformSpacing                   : 0
NumTileColumnsMinus1                 : 0
TileColumnWidthArray                 : 9 6
NumTileRowsMinus1                    : 0
TileRowHeightArray                   : 5 5 4
LFCrossTileBoundaryFlag              : 1
```

Figure 20. Slices and Tiles modes in configuration file

The following figures show some examples on how to partition video frames into tiles and slices by using the configuration file parameters that were referred above. The YUV video that has been chosen for this test is “Kimono_1920x1080_24.yuv”. It has 1920 pixels frame width, 1080 pixels frame height and frame rate 24 fps. Firstly, each video has been coded following up different slice and tile partitioning parameters by HM Encoder and then the encoded bitstream outputs were used in “HEVC Video Analysis Tool” software application of master thesis in order to take the appropriate results for this test.

Figure 21 illustrates a frame of decoded video, where hasn't been applied any configuration of tile and slice partitioning. That happened because parameter SliceMode has taken the value 0, which disables all slice and tile options.

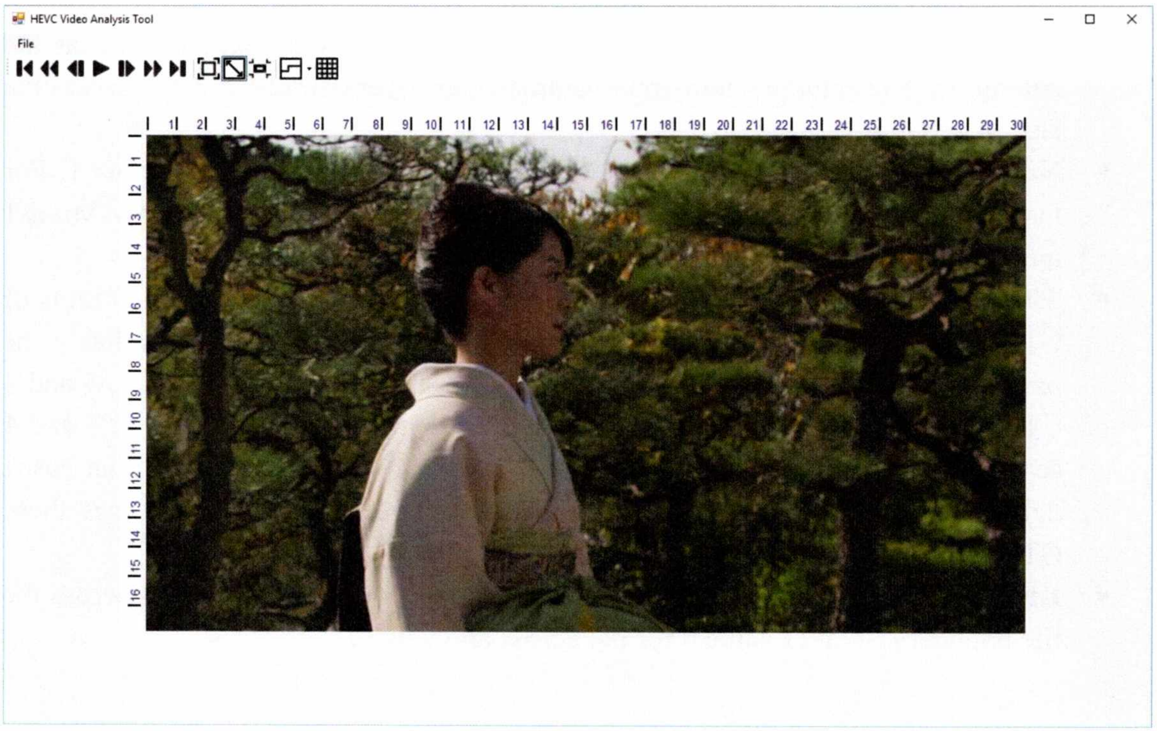


Figure 21. Decoded video “Kimono” without slice and tile partitioning (SliceMode=0)

Figure 22 shows a frame of decoded video, where has been divided into 6 uniform slices with 85 CTUs each one. In this example, SliceMode parameter was taken the value 1 and SliceArgument parameter was taken the value 85 in the configuration file. When SliceMode is 1, frames are cut into slices with maximum number of CTUs the SliceArgument value (85 CTUs current value).

Figure 23 illustrates a frame of decoded video, where has been divided into 4 uneven slices with maximum 10,000 bytes each one. In this instance, SliceMode parameter was taken the value 2 and SliceArgument parameter was taken the value 10,000 in the configuration file. When SliceMode is 2, video frames are cut into slices with maximum number of bytes the SliceArgument value.

Figure 24 illustrates a frame of decoded video, where has been partitioned into 16 tiles with different number of CTUs each one. In slice mode, SliceMode parameter is equal to 3 and SliceArgument parameter is equal to 16. It means that encoder of video has applied tile partitioning, where each slice size depends on tile assignments, and has distributed maximum 16 tiles in each slice per frame. Current frame hasn't divided into slices, because SliceArgument parameter is equal to the number of tiles in the frame. In tile mode, number of tile columns and number of tile rows are 4 (value 3 in corresponding parameters). Tile 1st column has 8 CTUs width, 2st column has 5 CTUs width, 3rd column has 9 CTUs width and 4th column takes the rest of CTUs for width (8 CTUs in this example). Tile 1st row has 6 CTUs height, 2st row has 3 CTUs height, 3rd row has 5 CTUs height and 4th row takes the rest of CTUs for height (3 CTUs in this instance). TileUniformSpacing parameter took value 0 for the purpose of adjusting unevenly column width and row height of each tile via array parameters.

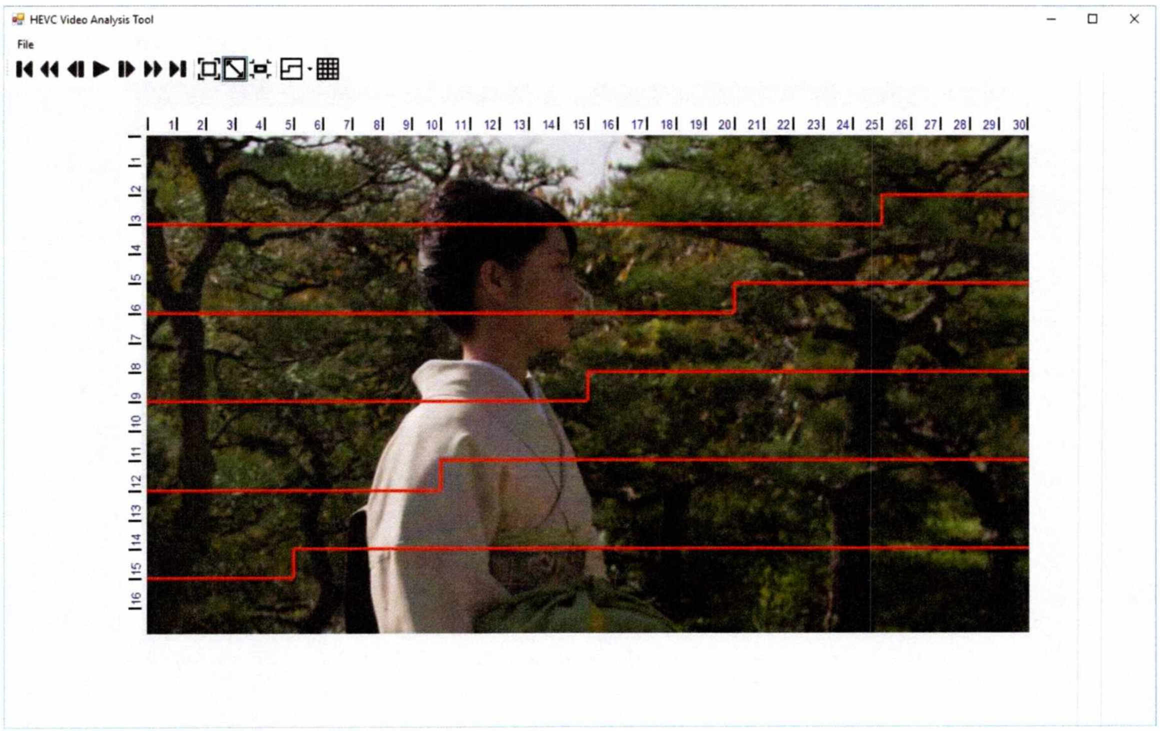


Figure 22. Decoded video "Kimono" with slice partitioning (SliceMode=1)

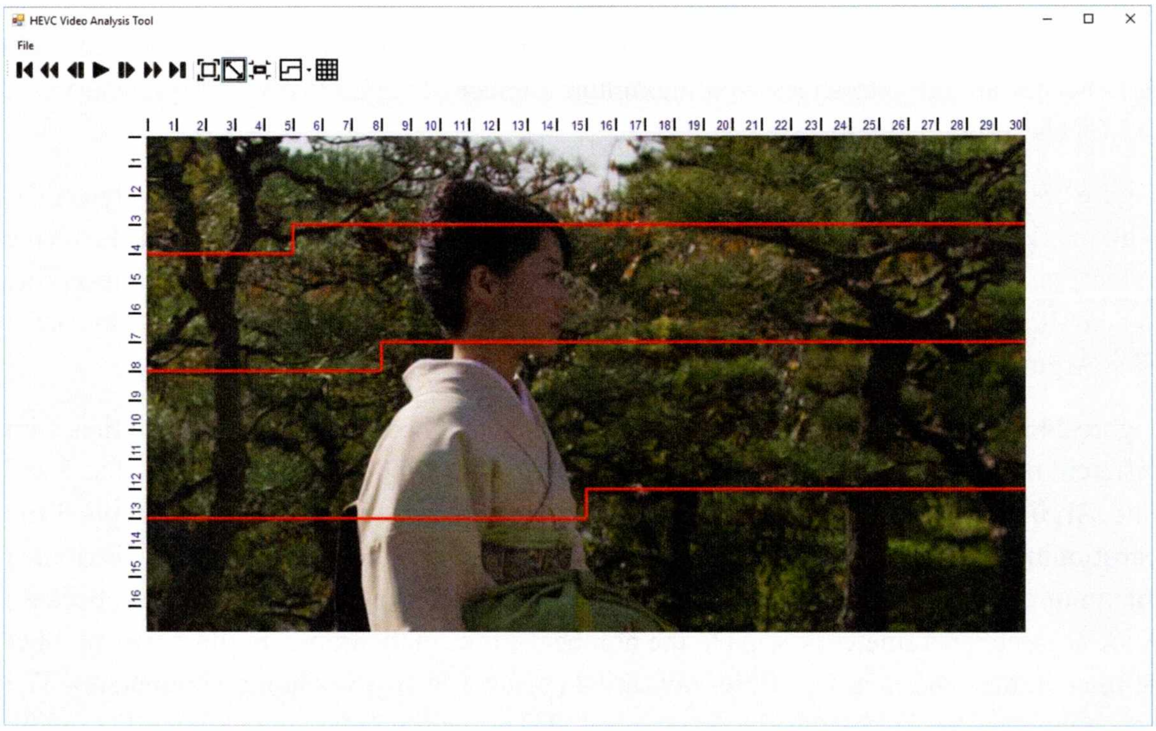


Figure 23. Decoded video "Kimono" with slice partitioning (SliceMode=2)

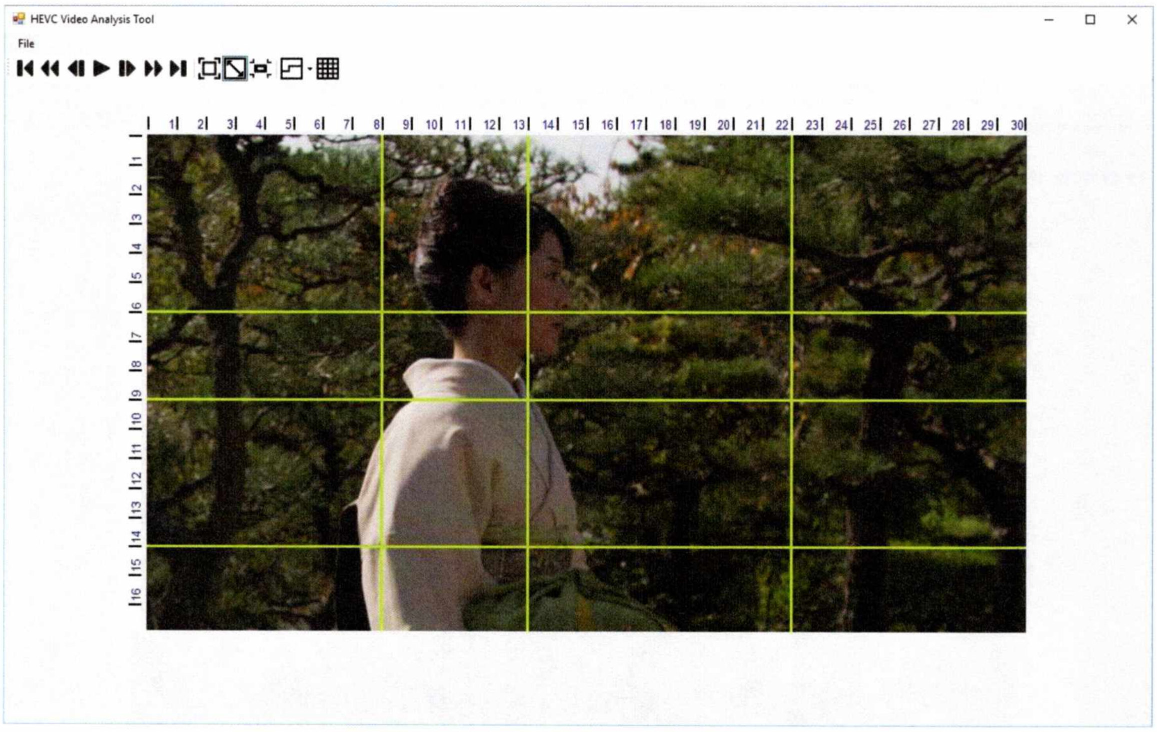


Figure 24. Decoded video "Kimono" with tile partitioning (SliceMode=3 & TileUniformSpacing=0)

Figure 25 and Figure 26 have been produced from the same configuration file settings and essentially show a frame that has been partitioned in both slices and tiles. In Figure 25, frame has 16 tiles with different number of CTUs each one and in Figure 26, the same frame has 6 slices with different number of CTUs each one. In slice settings, SliceMode parameter has value 3 and SliceArgument parameter has value 3. It means that encoder of video has applied tile mode, where each slice size depends on tile assignments, and has distributed maximum 3 tiles in each slice per frame. In tile mode, number of tile columns and number of tile rows are 4 (value 3 in corresponding parameters). Tile 1st column has 6 CTUs width, 2st column has 8 CTUs width, 3rd column has 6 CTUs width and 4th column takes the rest of CTUs for width (10 CTUs in this example). Tile 1st row has 4 CTUs height, 2st row has 2 CTUs height, 3rd row has 5 CTUs height and 4th row takes the rest of CTUs for height (6 CTUs in this instance). TileUniformSpacing took value 0 for the purpose of adjusting unevenly column width and row height of each tile via array parameters.

Figure 27 and Figure 28 have been also resulted from the same configuration file settings and illustrate one frame that have been partitioned in both slices and tiles. In Figure 27, frame has 25 tiles with various number of CTUs each one (depending on frame division into tiles), and in Figure 28, the same frame has 7 slices with different number of CTUs each one. It's observed that the adjacent horizontal tiles in each row have equal number of CTUs between them in Figure 27. SliceMode option has value 3 and SliceArgument option has value 4. Same as the previous instance, the encoder of video has defined tile mode, where each slice size depends on tile assignments, and has grouped maximum 4 tiles in each slice per frame. This time, current test has been run with TileUniformSpacing parameter equal to 1. It means that the encoder had to allocate tile column and tile row boundaries uniformly. In order to achieve that, the division of frame width in CTUs with NumTileColumnsMinus1 value and the division of frame height in CTUs with NumTileRowsMinus1 have to give zero remaining. Otherwise, the encoder tries to determine tile column and/or tile row boundaries in a different way in order the number of tile columns and number of tile rows remain the same. So, number of tile columns and number of tile rows are 5 (value 4 in corresponding parameters). Tile columns have been equally separated with 6 CTUs width each one (30 CTUs frame width % 5 tile columns = 0 -> gives uniform tile column widths). However, tile rows have been partitioned into two sizes, 1st, 2nd and 4th tile rows have 3 CTUs height each one and 3rd and 5th tile rows have 4 CTUs each one, because the remaining of division isn't zero (17 CTUs frame height % 5 tile rows = 4 -> gives uneven tile row heights). Furthermore, TileColumnWidthArray and TileRowHeightArray options didn't take into account, because TileUniformSpacing was 1 and didn't use array parameters for tile and slice partitioning.

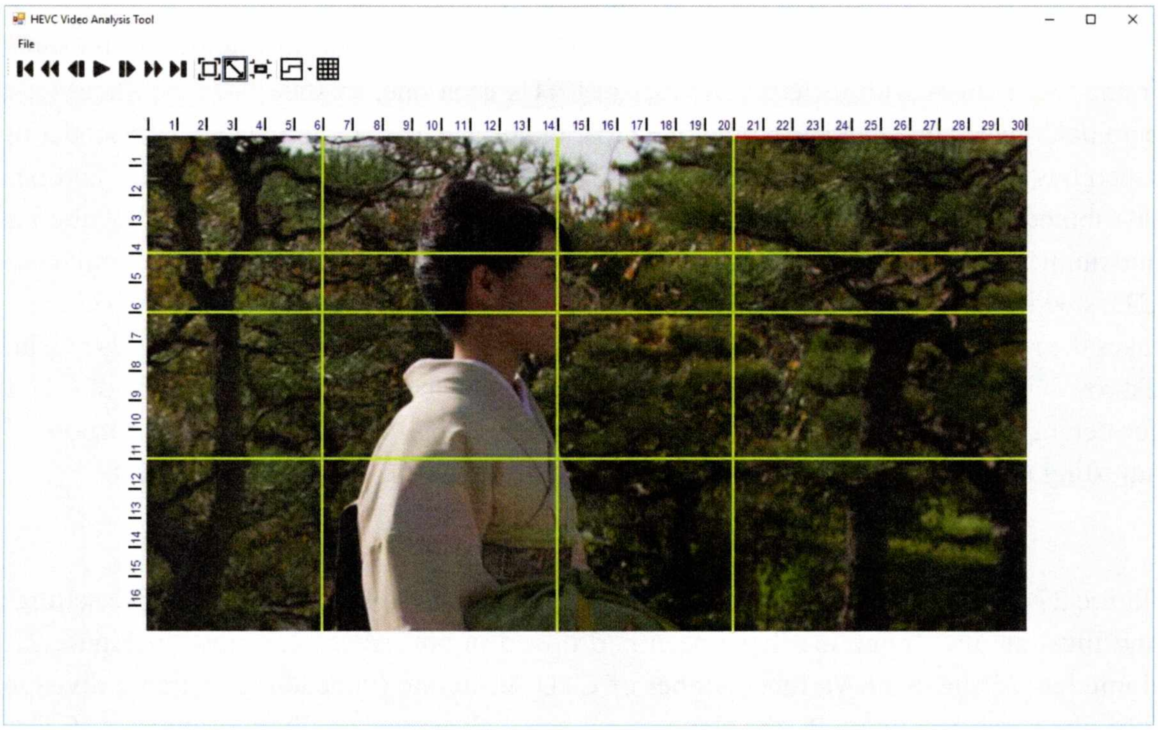


Figure 25. Decoded video "Kimono" with slice and tile partitioning (shows tile boundaries only and has SliceMode=3 & TileUniformSpacing=0)

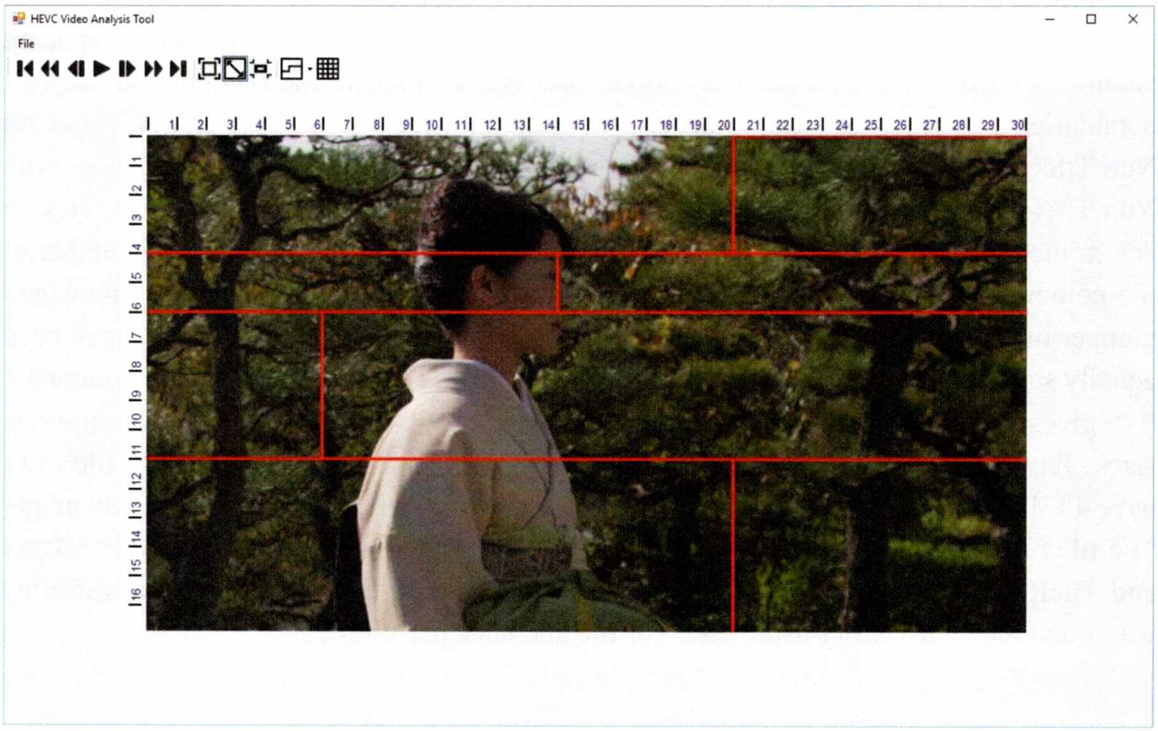


Figure 26. Decoded video "Kimono" with slice and tile partitioning (shows slice boundaries only and has SliceMode=3 & TileUniformSpacing=0)

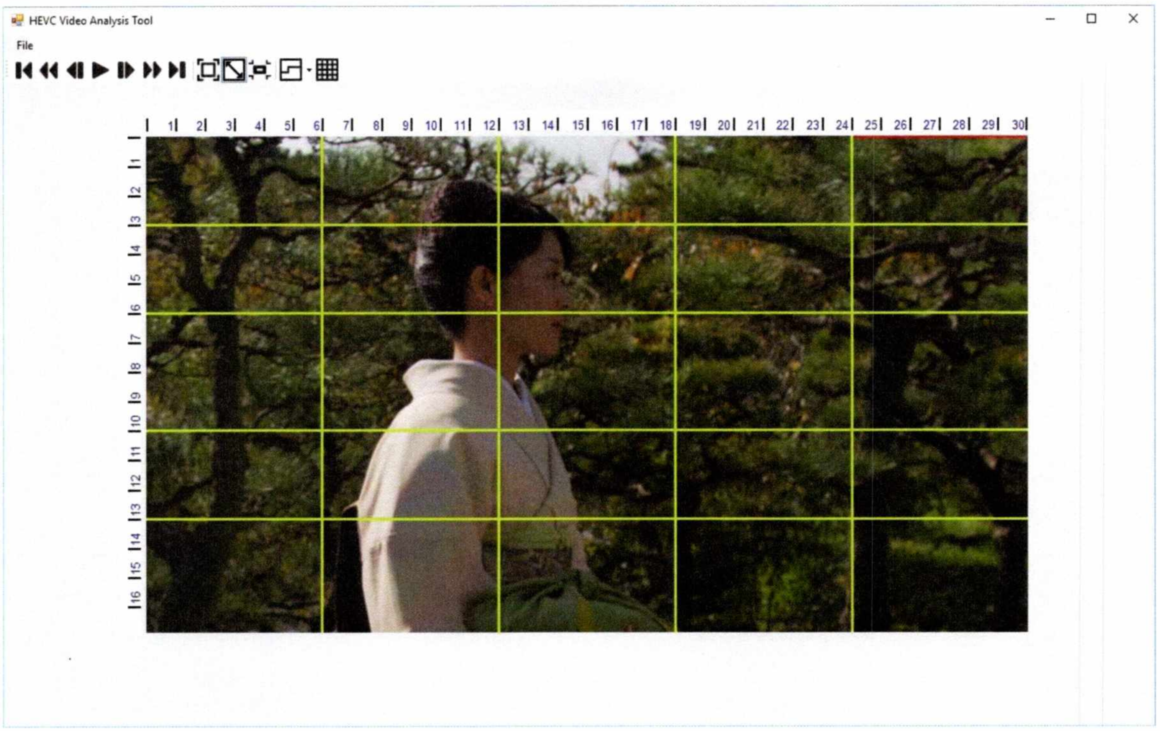


Figure 27. Decoded video "Kimono" with slice and tile partitioning (shows tile boundaries only and has SliceMode=3 & TileUniformSpacing=1)

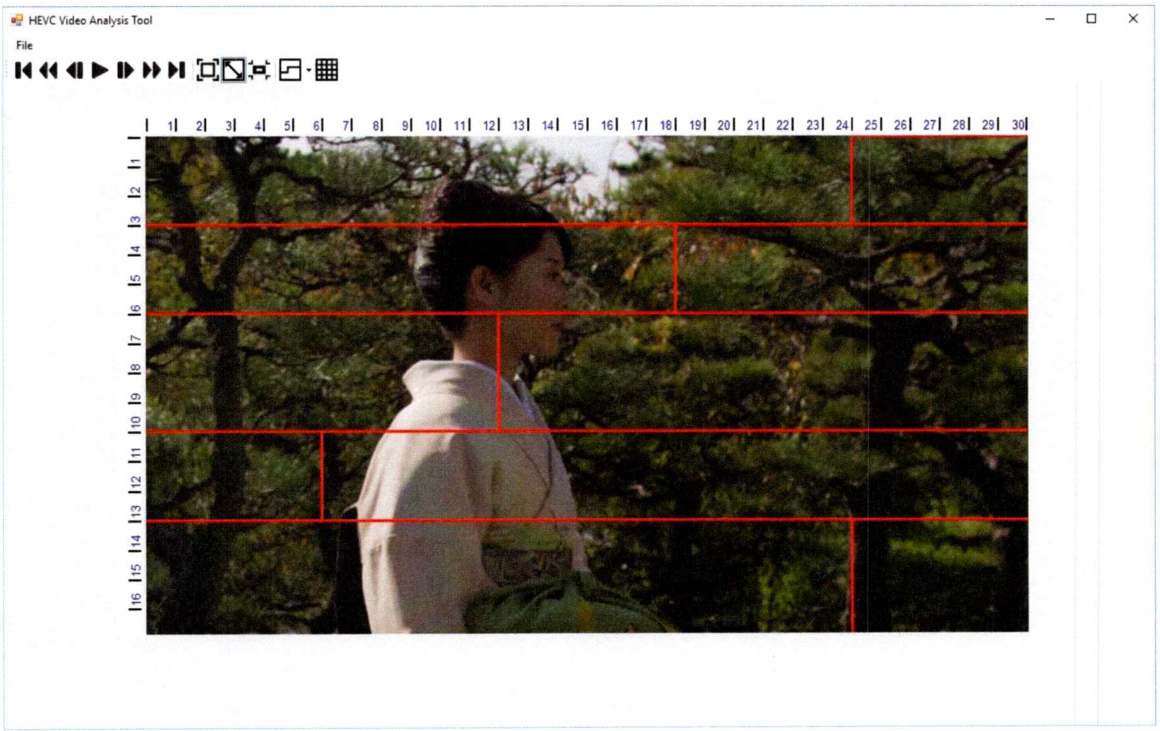


Figure 28. Decoded video "Kimono" with slice and tile partitioning (shows slice boundaries only and has SliceMode=3 & TileUniformSpacing=1)

CHAPTER 5: HEVC VIDEO ANALYSIS TOOL

In this chapter, it will be extensively presented the main purpose of the redaction of this master thesis. It will provide all necessary information about understanding the construction, handling and functionality of implemented software. Furthermore, there will be many tests for each stage of the program, where will offer a helpful guidance for those who intend to use this software and for those who just wants to know how the code works. Additionally, will be analyzed the significant parts of code such as the classes, the functions, the structures, the variables and some of the libraries, and also the designing of the window application with its controls.

The application of master thesis is called "HEVC Video Analysis Tool" and has been developed for the purpose of decoding simply and automatically a bitstream file that encoder has been produced and showing the frame sequence of decoded video into a suitable designed window. In this window, it has been added and programmed several tools and buttons, in which have been assigned particular operations and actions for each one. These operations, such as decoding a compressed file, managing the decoded video sequence, showing slices and tiles onto each frame, etc. will be described in detail below.

The software of HEVC encoder and decoder, which is used in master thesis project, has been jointly developed by the ITU-T Video Coding Experts Group (VCEG, Question 6 of ITU-T Study Group 16) and the ISO/IEC Moving Picture Experts Group (MPEG, Working Group 11 of Subcommittee 29 of ISO/IEC Joint Technical Committee 1) [12]. The packet of this software includes all source and header files for encoder and decoder, which were written in C++ programming language, and the executable files of encoder and decoder in debug and release mode as well. Also, it provides the Reference software Manual that is useful in assisting users of a video coding standard to establish and test conformance and interoperability, and to educate users and demonstrate the capabilities of HEVC standard. Last but not least, it also provides configuration files (based on default test conditions as they were specified in [9]) that help the user to define all encoding parameters as desired before compress the video (the content of configuration file is referred in Chapter 3 and 4). Within the framework of this thesis we used version 16.15 of HM Reference Software [12].

The main software "HEVC Video Analysis Tool" was developed by using the program Microsoft Visual Studio 2015 [14]. Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft, that it is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps. Visual Studio includes the code editor for typing program code and the integrated debugger, which works both as a source-level debugger and as a machine-level debugger. There are also many built-in tools such as code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer. Furthermore, it can use plug-ins that may improve the functionality of any level of the program, for example by adding new toolsets like editors and visual designer. Visual Studio has been designed to support

36 different programming languages by allowing code editor and debugger to recognize them. The build-in languages that program supports are C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), F# and TypeScript. The other languages that program may support, as long as the appropriate language services are installed, are Python, Ruby, Node.js, M, etc. Additionally, it supports web designing languages such as XML/XSLT, HTML/XHTML, JavaScript and CSS. The first version is Visual Studio 97 that is developed in February 1997 and the last and the newest version is Visual Studio 2017 that is released on 7 March 2017.

"HEVC Video Analysis Tool" is a Graphical user interface (GUI) application developed in Visual Studio 2015, as referred at previous paragraph, and composed in C [19], C++ [20] and C++/CLI [21] (via Visual C++) programming languages. Also, the project was run and tested by using the Common Language Runtime (CLR) [22], which is the virtual machine component of Microsoft's .NET framework. CLR is a process that its job is to convert compiled code into machine instructions and then executed by the computer's CPU. All .NET framework versions, regardless of programming language, are supported by CLR. Furthermore, CLR implements the Virtual Execution System (VES) as defined in the Common Language Infrastructure (CLI) standard, which is developed by Microsoft. Essentially, CLI allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures. CLR supports Visual C++, as well as C and C++ programming languages, where a combination of all three were needed for developing the project.

Visual Studio projects are partitioned into five main parts, where each one has a different role and provides particular information that will used for the execution of a project. These parts are the following:

- References. Before writing code against an external component or connected service, your project must first contain a reference to it. A reference is essentially an entry in a project file that contains the information that Visual Studio needs to locate the component or the service. In the current project "HEVC Video Analysis Tool" hasn't been added any kind of reference.
- The external dependencies contain all required libraries, which have been installed into user's computer, in order to be used in main classes of the project. For example, math.h library is an external dependency that Visual Studio has already included it and defines various mathematical functions and one macro. The user can also specify the location of a dependent library that will be used in the project. In "HEVC Video Analysis Tool" there isn't assigned any external dependency by the user.
- Header files. A header file in C/C++ may contain function, structure and variable declarations, macro definitions, as well as namespaces with nested classes and functions, where are shared between several source files of a project. Their extension is .h and are included in .c file by adding the phrase #include, one space and the name of header file with double quotation marks in the top lines of needed source file (for example #include "MyForm.h"). Also, there are two types of header files: the files that the programmer writes and the files that comes with your

compiler (it is needed to put <...> instead of "..."). "HEVC Video Analysis Tool" project has two header files: "MyForm.h" and "yuv.h".

- Resource files. A resource file is a text file that can be a resource script (.rc) file of a program, a resource template (.rct) file, an individual resource existing as a stand-alone file, such as a bitmap, icon, or cursor file that is referred to from an .rc file or a header file generated by the development environment, for example Resource.h, that is referred to from an .rc file. Resources can also be found in other file types such as .exe, .dll, and .res files. There is a capability of using resources and resource files from within a project and with those that are not part of your current project, as well as working with resource files that were not created in the development environment of Visual Studio. "HEVC Video Analysis Tool" project hasn't any resource file.
- Source files. A source file contains the source code of a project and specifically the main or common classes and functions with lines of code which will be used for compilation and execution of a program. The written code of source files may be declaration and definition of variables, functions, structs and classes, as well as probably calculations for the project solution. Source file extension in C and C++ programming languages is .c and .cpp, respectively. "HEVC Video Analysis Tool" project has got two source files: MyForm.cpp and yuv.cpp.

Code files and Functions

As it already referred above, code files of the project "HEVC Video Analysis Tool" are: "MyForm.cpp", "MyForm.h", "yuv.cpp" and "yuv.h". The main reason, why these code files were composed, is to achieve master thesis goals. Actually, execution of code files has the result of a designed form that users can interact with interface of the software. Designing of the form and connection between form and code files have been done via Microsoft Visual Studio 2015. In next paragraphs, technical specifications, the structure of code files and windows form will be explained in detail. Furthermore, all functions included in source files will be described for better comprehension of the program.

"MyForm.h" file was written in a combination of three programming languages: C [19], C++ [20] and C++/CLI [21] (Visual C++). All these languages can be recognized from the compiler via the Common Language Runtime (CLR). Also, the file has been categorized as a header file and is included in the specific directory, but isn't considered C++ header file due its .h extension. Actually, Visual Studio has defined this file as a C++ Form file, which provides three views for its development: the code view, the designer view and the class diagram view. Class diagram view wasn't needed, so it wasn't used in the project. Designer view is the mode that a developer is capable of designing onto a form the desirable Graphic User Interface (GUI) by adding buttons, menus, picture and text boxes, labels, etc. The program provides a toolbox with all the appropriate designing tools, which can be added into the form by choosing the object firstly and then adjusting size and location of the object by clicking on the form. Code view is a mode that gives the opportunity to programmers to write suitable code in order to define the functionality of the objects that are located in corresponding windows form. When a form file is created, the main program automatically generates all necessary code lines, which initialize an instance of the form class and define the form class properties. Furthermore, all additional form controls that has been imported in the form during development of the project, also generating code for declaring the form controls and defining attributes of the controls. Declarations are created outside of pragma region and definitions are added into pragma region of code form file. The initialization of controls and the definition of control properties are instantly placed by the program inside InitializeComponent function every time a control unit is added in the form.

First of all, it will be described "MyForm.h" file of "HEVC Video Analysis Tool" and its consistency. In the first lines of code file, it is declared all necessary libraries, where some of them are composed by the programmer and the others are provided by the compiler. The libraries that have been added are: openCV [15] for processing YUV images, wchar.h for wide character conversions, msclr\marshal.h for string conversions, yuv.h for using the functions of yuv.cpp, etc. After libraries, there are declarations and definitions of the structures and they are as follow:

- Struct frame. It includes integer members about frame width and height in pixels, number of slices and tiles in video frame, frame width and height in CTUs and CTU width and height in pixels. Also, there are two pointer declarations of slice and tile structs.
- Struct slice. It contains integer members about the ID number of start CTU in the slice and also the total CTUs of the previous slice in a frame.
- Struct tile. It consists of two integer members: the total CTUs of the tile and the ID number of first CTU in the tile.
- Struct YUV_Capture cap. It declares "cap" of type YUV_capture struct that has been defined in yuv.h header file.
- Struct frame *pics. It declares "pics", a pointer to frame structs that has been defined above.

The next step of the code file is to define a namespace named "Project3" and include in it the public class MyForm and declarations of using other namespaces such as System. All the functionality of the project "HEVC Video Analysis Tool" originates from MyForm class, which controls and manages the main functions and sets the properties of each control in the form. MyForm class consist of public, private or protected variable and object declarations, as well as function definitions. All MyForm variables outside functions are public integer or boolean and have been declared at the beginning of the class. For example, some of them are the bool type "forward", which controlling whether video plays forward or not, and the int type "total_frames", which stores the total number of frames in current video. Also, declaration of the objects are public and private and most of them are objects from controls such as buttons and picture boxes that have been added into the form. The other declared objects are Bitmap, File, Thread, etc. The most important thing that is deemed necessary is the analysis of all the functions of MyForm class for the purpose of understanding how the application manages a compressed video. The functions are as follow:

MyForm is a class constructor and is used to initialize form components. It calls `InitializeComponent` function in order to initialize form objects and define their properties of form objects. Additionally, it calculates with accuracy the distance between picture box right side and form right side and also the distance between picture box bottom side and form bottom side. This information is needed for keeping the same distance between picture box and form when window is maximized and minimized.

~MyForm is a class destructor and is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. However, its main task is to clean up any resources being used.

openToolStripMenuItem_Click is a function that is enabled when a user clicks on the button "Open" of folding menu "File". The process of this function is to show an open file dialogue window, where the user may choose a binary encoded video file (.bin) from computer storage devices for decoding and then to show a save file dialogue window, where the user may assign a name for the exported decoded video file (.yuv) and save it into the selected path. In case of something goes wrong at file selection, the function shows an error message to the user and terminates the procedure. If both files were successfully selected by the user, the main function calls another function named `ShowMyImage` with two string arguments: the filename path of open dialogue box and the filename path of save dialogue box.

ShowMyImage is a function that runs several computations and calls many sub-functions in order to show the first frame of decoded video into the video display box of window. Firstly, it defines a string variable called "command", which stores the command that the user will type it manually into command prompt for executing the decoder. In this variable, it has been merged three string variables: the name of decoder application and the string of open dialogue box and the string of save dialogue box that has been passed into function arguments through `openToolStripMenuItem_Click` function. After that, the function creates a process, where its task is to import the string variable with decoding command into command prompt and to execute it. Unless the process hasn't send an error, it continues by checking the exported file that has been produced by decoder. If file doesn't open, the function will be terminated. The next step is to call `CtuParser` function that defines the "pics" members of type frame struct and sets the size of struct depending on frames of decoded video. The most important part of `ShowMyImage` function is happened by taking the first frame from decoded video file and converting it into IplImage struct. For the purpose of achieving this process is used functions from "yuv.cpp" file and OpenCV libraries. At the end, it calls the functions: `DrawMyImage` with argument the IplImage struct, `SliceParser` and `TileParser` for defining members of "sl" and "ti" structs and it refreshes all picture boxes for showing partitions on the frame and vertical and horizontal ruler. All the extra functions that have been executed into this function will be described in detail later on. Briefly, the procedure of `ShowMyImage` function is to decode an encoded binary file that the user selected and then show the first frame from decoded video file into video display box of windows form. Additionally, the frame has been possibly partitioned into slices and tiles depending on configuration file parameters.

`DrawMyImage` is a function that has a single argument, the pointer to IplImage struct. Its procedure is to take some data from IplImage struct such as width and height of picture, and convert them to a Bitmap object in order to illustrate the image into picture box. Also, it calls the function `SetImage` with resulted Bitmap object as an argument.

`SetImage` is a function that have the Bitmap object as an argument and its main purpose is to make thread-safe the `pictureBox1` control of the form. Also, it checks if there are different running threads in the current application by comparing the thread ID of the calling thread to the thread ID of the creating thread. If so, it takes the right thread ID for defining the image. The result of this functions is to illustrate the image onto picture box.

`UpdateButton` is a similar function as `SetImage`. It is used to define an image into a tool strip button. The call of this function happens safely by checking first if there are different threads in use. Furthermore, the only button that is updated from this function is the play/pause button which may be alternating during execution of the program.

`refreshPictureBox1` is a function that refresh the `pictureBox1` which is the video display box. Also, the call of this function happens safely by checking first if there are different threads in use.

`CtuParser` is a function that reads an input stream file named "ctu_info.txt", which is produced after decoding execution. Specifically, the text file includes data about each frame of the video, such as width and height in CTUs and pixels. There is parser inside a case loop that pass all data about CTUs and size of each frame into proper members of "pics" struct pointer. At the end of the function, the text file is deleted, because it won't be needed again.

`SliceParser` is a similar function as `CtuParser` that reads an input stream file named "slice_info.txt", which is produced after decoding execution. Specifically, the text file includes data about the number of CTUs in each slice and the start CTU in each slice. Also, there is parser inside a case loop that pass all data about slices of each frame into proper members of "sl" sub-struct pointer, which is nested inside "pics" struct pointer. At the end of the function, the text file is deleted, because it won't be needed again.

`TileParser` is a similar function as `CtuParser` and `TileParser` that reads an input stream file named "tile_info.txt", which is produced after the decoder execution. Specifically, the text file contains data about the number of CTUs in each tile and the first CTU in each tile. Also, there is parser inside a case loop that pass all data about tiles of each frame into proper members of "ti" sub-struct pointer, which is nested inside "pics" struct pointer. At the end of the function, the text file is deleted, because it won't be needed again.

pictureBox1_Grid is a function that controls the frame partitioning designing of main picture box by using grid menu buttons. The function manages three basic operations of partitioning: the CTU grid, the slice partitioning and the tile partitioning. Every of these operations may be enabled and disabled by a tool strip button at the top of windows form. So, it's logical that the function is triggered when one of these keys is clicked. However, the trigger is controlled by `PaintEventHandler` that means it is necessary the refreshing of picture box in order to show the partitioning. Also, it uses the class graphics, which helps for drawing CTU grid lines and slice and tile borders onto video display box. Each partitioning operation is coded inside a "for" repeat loop and makes calculations for the points that the lines will be drawn, exploiting the data that is carried in "pics" struct pointer. Additionally, before getting into a repeat loop, it firstly checks whether the proper grid button is enabled or not, as well as if there are slices and tiles at current frame to draw.

pictureBox2_VerticalRuler is a function that designs a vertical ruler on the left side of the video display box, which measures the number of CTU rows. As in `pictureBox1_Grid`, there is also a `PaintEventHandler` that controls the function. For example, when the frame size is changed by the user via tool strip buttons, the picture box that contains the vertical ruler is refreshed in order to draw the new coordinates for each CTU row. Also, it uses the class graphics for painting vertical ruler onto picture box and the operation of drawing is implemented into a "for" repeat loop, which calculates the exactly coordinates of vertical ruler.

pictureBox3_HorizontalRuler is a similar function as `pictureBox2 _VerticalRuler`. The only difference is that designs a horizontal ruler (instead of vertical ruler) on the top side of the video display box, which measures the number of CTU columns. Same as `pictureBox2_VerticalRuler`, it uses a `PaintEventHandler` and refreshes the horizontal ruler picture box every time something changes in video display box.

replaceChar is a function that replace a chosen character in a string with the characters "//". It takes two arguments: the string and the character that will be replaced. In this case, it is used to replace character "\" with characters "//" in each path filename.

InitializeComponent is a function that is created automatically by Visual Studio when the main project is built. As it referred, this function is called by `MyForm` constructor for the purpose initializing a new instance for each form control class and defining their properties. Also, it determines an object called resources that has been stored all the images from designer view of the form. In `InitializeComponent`, the images for each tool strip button is defined by getting the proper image file from resources object.

toolStripButton6_Click is a function that controls the `toolStripButton6` object. When the user clicks on this button, it moves the video one frame forward. It calls the function `YUV_read` from yuv.cpp file to find the next image of the video and then refreshes the video picture box in order to illustrate any possibly partitioning graphics on it. The button doesn't work when reaches the last frame.

toolStripButton4_Click is a function that controls the `toolStripButton4` object. When the user clicks on this button, it moves the video one frame backward. It uses the function `YUV_backread` from yuv.cpp file to find the previous image of the video and then refreshes the video picture box in order to illustrate any possibly partitioning graphics on it. The button doesn't work when reaches the first frame.

toolStripButton3_Click is a function that controls the `toolStripButton3` object. The user is able to play the video at normal speed or to pause the video by clicking this button. Specifically, the operation depends on button icon that appears on it each time. Firstly, the function checks a boolean variable, which defines whether the video is playing or not. If the video is paused, then it creates and initializes a new delegate thread that takes as an argument the function of playing procedure. So, the new thread runs function `PlayMyVideo` until reaches at the end frame of video or something stop it. If the video is playing and the button is clicked, then it suspends the new thread and pauses the video.

PlayMyVideo is a function that is run by a new produced thread for the purpose of playing the video. The function consists of a "while" repeat loop, where in each repeat illustrates the next frame of the video stream and freezes the process for a few milliseconds based on frames per second (fps) of the video. For example, if a video has 30 fps, the repeat procedure should wait a few milliseconds each time in order to achieve the desirable frame rate. Also, the procedure stops when the video sequence reaches the last frame or the thread is suspended.

toolStripButton8_Click is a function that controls the `toolStripButton8` object. When the user clicks on this button, it shows the last frame of the video. It runs the function `YUV_seekframe` from yuv.cpp file to find the last image of the video and then refreshes the video picture box in order to illustrate any possibly partitioning graphics on it.

toolStripButton1_Click is a function that controls the `toolStripButton1` object. When the user clicks on this button, it shows the first frame of the video. It executes the function `YUV_seekframe` from yuv.cpp file to seek the first image of the video and then refreshes the video picture box in order to show any possibly partitioning graphics on it.

PlayFastMyVideo is exactly the same function as `PlayMyVideo` with two differences. This function doesn't make the procedure sleep at all, because it has to project the frames of video onto the picture box as fast as possible. It is capable to play the video backward depending on boolean variable "forward".

toolStripButton7_Click is a function that controls the `toolStripButton7` object and follows the same operation as `toolStripButton3_Click` with one difference. The new delegate thread, which is created by clicking the button, will execute the function `PlayFastMyVideo` (instead of `PlayMyVideo`) until reaches at the end frame of video.

Also, the procedure may be stopped when one of the buttons `toolStripButton3` or `toolStripButton7` are pressed.

toolStripButton2_Click is a function that controls the `toolStripButton2` object and has the exactly same functionality with `toolStripButton7_Click`, but it has one change. This time, it rewinds the video backward, instead of playing it on fast forward. The check whether the video goes forward or backward is set by a boolean variable named "forward" (if true, the video plays forward).

toolStripButton11_Click is a function that controls the `toolStripButton11` object. When the user clicks on this button, it defines the size mode of video picture box to normal size. So, if video resolution is larger than picture box size, then it won't show the whole size of the picture. Also, it adapts the horizontal and the vertical ruler coordinates considering the size of the picture box.

toolStripButton10_Click is a function that controls the `toolStripButton10` object. When the user clicks on this button, it defines the size mode of video picture box to stretched image. It means that the size of each frame stretches in order to adapt to picture box size. Also, it adapts the horizontal and the vertical ruler coordinates considering the size of the picture box.

toolStripButton9_Click is a function that controls the `toolStripButton9` object. When the user clicks on this button, it defines the size mode of video picture box to auto size. Auto size mode means that the size of picture box stretches in order to adapt to the size of each frame. If the video resolution is too high, picture box size may overcome screen borders. Also, it adapts the horizontal and the vertical ruler coordinates considering the size of the picture box.

slicesToolStripMenuItem_Click is a function that controls the `slicesToolStripMenuItem` object. The user may enable or disable the slice partitioning view on the video picture box by checking or unchecking the menu button, respectively. Also, it calls refresh function of video picture box in order to run `pictureBox1_Grid` function and draw the graphics.

tilesToolStripMenuItem_Click is a function that controls the `tilesToolStripMenuItem` object. The user may enable or disable the tile partitioning view on the video picture box by checking or unchecking the menu button, respectively. Also, it calls refresh function of video picture box in order to run `pictureBox1_Grid` function and draw the graphics.

MyForm_Resize is a function that is triggered when a user changes the size of the windows form. It essentially adapts the picture box size considering the new form size. Every time it resizes the picture box, it keeps a constant distance from the form borders.

toolStripButton12_Click is a function that controls the `toolStripButton12` object. The user may enable or disable CTU grid view on the video picture box by checking or unchecking the menu button, respectively. Also, it calls refresh function of video picture box in order to run `pictureBox1_Grid` function and draw the graphics.

exitToolStripMenuItem_Click is a function that controls the `exitToolStripMenu` object. The "Exit" menu button is located inside the folding menu named "File". The user may terminate the windows form by clicking on this button.

closeToolStripMenuItem_Click is a function that controls the `closeToolStripMenu` object. The "Close" menu button is located inside the folding menu named "File". The user may close the video stream and clear the video picture box by clicking on this button, if there is already a video running in picture box.

Note that it has been added code into file "TDecSlice.cpp" of decoder project named "TLibDecoder" for the purpose of exporting data from decoding procedure. The result of this code is the creation of the text files "ctu_info.txt", "slice_info.txt" and "tile_info.txt", where their data used in `CtuParser`, `SliceParser` and `TileParser` function, respectively. Specifically, they provide information about frames, CTUs, slice and tile partitioning.

After the analyzation of "MyForm.h" file, the next step is to proceed into the main file. To be more specific, the file, which contains the main function of application, creates a form object and runs the windows form, is "MyForm.cpp". In this file has been included the library of "MyForm.h" file and has been used two namespaces: the System and the System::Windows::Forms. There is also a line before main function that defines the use of STAThread. STAThreadAttribute indicates that the COM threading model for the application is single-threaded apartment. This attribute must be present on the entry point of any application that uses Windows Forms; if it is omitted, the Windows components might not work correctly. If the attribute is not present, the application uses the multithreaded apartment model, which is not supported for Windows Forms.

There are also as mentioned two assistant code files: the header file "yuv.h" and the source file "yuv.cpp". Both files contributes to project a sequence of YUV pictures onto the video display box of application. They include the Open Source Computer Vision Library version 3.2.0 named openCV [15], which contains a significant struct named IplImage and a number of functions. IplImage struct has several public members that gives the opportunity to divide an image into Y luminance image and into C_b and C_r chroma image. Header file yuv.h consists of an enum variable named YUV_ReturnValue declaration that defines a set of named function results, a struct named YUV_Capture definition that includes all members about a YUV image such as width and height of image and declarations of "yuv.cpp" functions.

Source file "yuv.cpp" has got five significant functions that are able to manage a YUV video file and capture a specific frame from YUV video. The functions that make up it are:

- `YUV_init`. This function receives four arguments: the YUV video file to read, the width and height of video and the instance `YUV_Capture` of captured video frame. Its procedure is to initialize the members of `YUV_Capture` instance by calling the openCV function `cvCreateImage`. If one of the IplImage struct members of `YUV_Capture` instance has a null value, it will call `YUV_cleanup` function in order to deallocate memory block of the structs and it will also return the appropriate message.
- `YUV_read`. This function reads the next frame from a previously-instantiated `YUV_Capture` instance. It takes `YUV_Capture` instance as an argument, which reads the capture and stores the result in "ycrcb". The procedure is to read from the YUV video file the bytes of next frame that includes the Y bytes of Luminance image (width X length = number of Y bytes), as well as the C_b and C_r bytes of Chroma image ((width X length) / 4 = number of C_b and C_r bytes). In case there isn't any error while reading the file, it merges the three IplImage structs, Y, C_b and C_r image from the next frame, into a new YUV IplImage image named "ycrcb" and it returns an approval message.
- `YUV_backread`. This function reads the previous frame from a previously-instantiated `YUV_Capture` instance. It takes `YUV_Capture` instance as an argument, which reads back the capture and stores the result in "ycrcb". Firstly, the video file pointer goes a number of bytes equal to two YUV frames back by using the function `fseek`. Then, it reads from the YUV video the bytes of next frame (actually the bytes of previous frame that was demanded) that includes the Y bytes of Luminance image (width X length = number of Y bytes), as well as the C_b and C_r bytes of Chroma image ((width X length) / 4 = number of C_b and C_r bytes). In case there isn't any error while reading the file, it merges the three IplImage structs, Y, C_b and C_r image from the previous frame into a new YUV IplImage image named "ycrcb" and it returns an approval message.
- `YUV_seekframe`. This function seeks a specific video frame and reads it from a previously-instantiated `YUV_Capture` instance. This function receives two arguments: the `YUV_Capture` instance that reads the capture and stores the result in "ycrcb" and the number of frame that seeks in video sequence. The first step is to execute the function `fseek`, which sets the video file pointer to the appropriate point in the bytes of selected frame. Then, it reads from the YUV video file the bytes of next frame that includes the Y bytes of Luminance image (width X length = number of Y bytes), as well as the C_b and C_r bytes of Chroma image ((width X length) / 4 = number of C_b and C_r bytes). In case there isn't any error while reading the file, it merges the three IplImage structs, Y, C_b and C_r image from the selected frame, into a new YUV IplImage image called "ycrcb" and it returns an approval message.
- `YUV_cleanup`. This function receives the `YUV_Capture` instance as an argument. As it mentioned this function deallocates the memory of the `YUV_Capture` members that allocated during initialization, if the instance isn't already null.

Software Demonstration

Let's proceed now to the software demonstration of "HEVC Video Analysis Tool". The user is able to interact with a single form, when the program is running. The initial windows form of application is shown in Figure 29. Additionally, all remarkable tools and objects of the form has been noted in the figure.

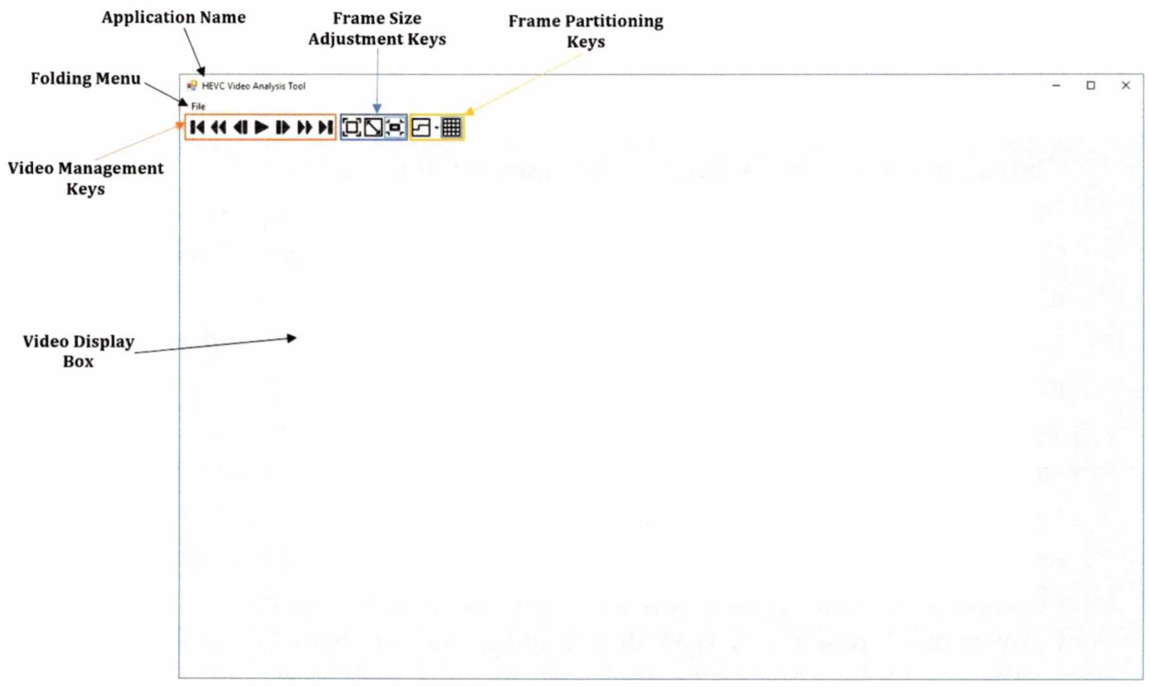
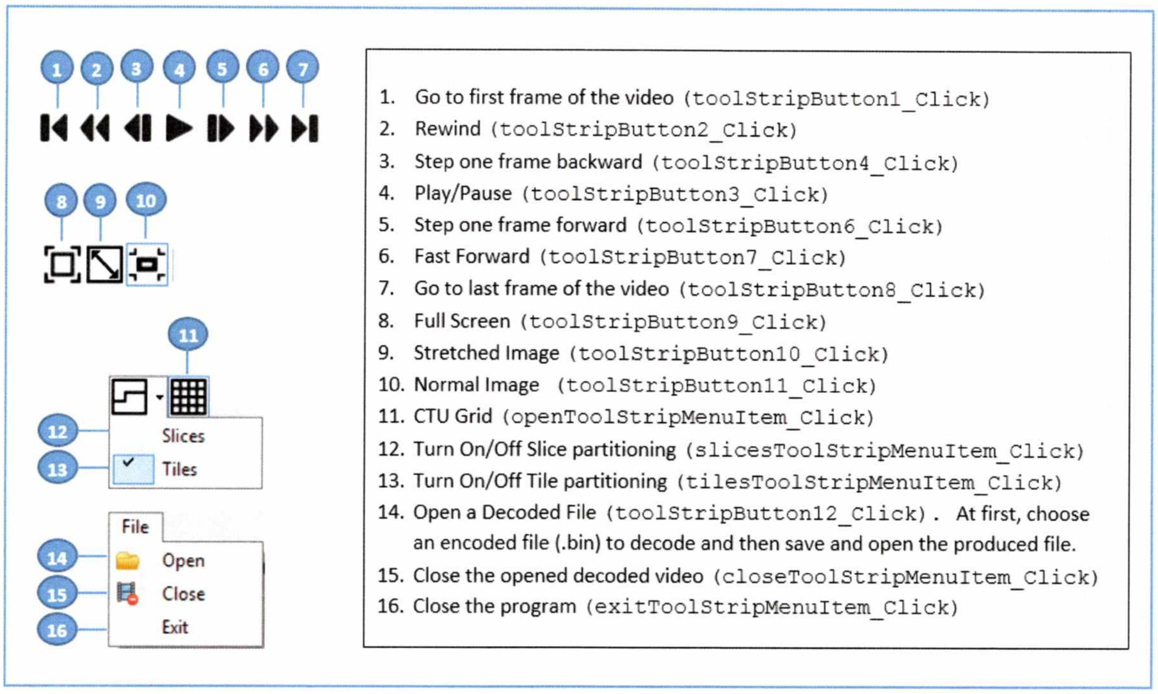


Figure 29. Initial Windows Form of "HEVC Video Analysis Tool"

Figure 30 summarizes the operation of each button in the windows form, as well as includes the proper function in "Myform.h" file that matches to each button. All functions that the figure contains has been described at the previous subchapter in detail. So, the buttons, the folding menu and the video display box form the user interface (UI), where its goals are to achieve the interaction and the control between application and user without any issue.

The first step of user, when running the program, is to define the file name of encoded video file (.bin) at first open dialogue box that will be decoded via HEVC decoder. However, the decoder is a different executable file and will be run through the command line as it described. After the open dialogue box, there is a second save dialogue box, where the user has to set a file name (.yuv) for saving the YUV decoded video. Since both selections of files have made, the application proceeds into decoding and then into illustration of the first decoded video frame onto picture box.



1. Go to first frame of the video (`toolStripButton1_Click`)
2. Rewind (`toolStripButton2_Click`)
3. Step one frame backward (`toolStripButton4_Click`)
4. Play/Pause (`toolStripButton3_Click`)
5. Step one frame forward (`toolStripButton6_Click`)
6. Fast Forward (`toolStripButton7_Click`)
7. Go to last frame of the video (`toolStripButton8_Click`)
8. Full Screen (`toolStripButton9_Click`)
9. Stretched Image (`toolStripButton10_Click`)
10. Normal Image (`toolStripButton11_Click`)
11. CTU Grid (`openToolStripMenuItem_Click`)
12. Turn On/Off Slice partitioning (`slicesToolStripMenuItem_Click`)
13. Turn On/Off Tile partitioning (`tilesToolStripMenuItem_Click`)
14. Open a Decoded File (`toolStripButton12_Click`). At first, choose an encoded file (.bin) to decode and then save and open the produced file.
15. Close the opened decoded video (`closeToolStripMenuItem_Click`)
16. Close the program (`exitToolStripMenuItem_Click`)

Figure 30. List of Buttons with their function

In order to check the correctness of the application and prove that there isn't any major error or bug, it was applied a test. Firstly, in this test was used the YUV video file "ParkScene_1920x1080_24.yuv" and also slice and tile parameters were set in configuration file as follows:

- SliceMode: 3
- SliceArgument: 5
- TileUniformSpacing: 0
- NumTileColumnsMinus1: 5
- TileColumnWidthArray: 4 4 5 6 7
- NumTileRowsMinus1: 3
- TileRowHeightArray: 3 5 4

In few words, these parameters partition each frame of the video into both slices and tiles. Also, each frame has got 5 unevenly slices and 24 unevenly tiles. Slices have been set to contain a maximum of 5 tiles each one. Frames have been divided into 6 tile columns and 4 tile rows. Tile 1st, 2nd and 6th column have 4 CTUs width, 3rd column has 5 CTUs width, 4th column has 6 CTUs width and 5th column has 7 CTUs width. Tile 1st row has 3 CTUs height, 2nd and 4th row have 5 CTUs height and 3rd row has 4 CTUs height. The operation of slice and tile parameters have been mentioned at Chapter 4 in detail.

The YUV video file is encoded manually with slice and tile parameters via HEVC encoder and then is decoded by following the first step of "HEVC Video Analysis Tool" application, as it described above. Figure 31 illustrates the result of this test.

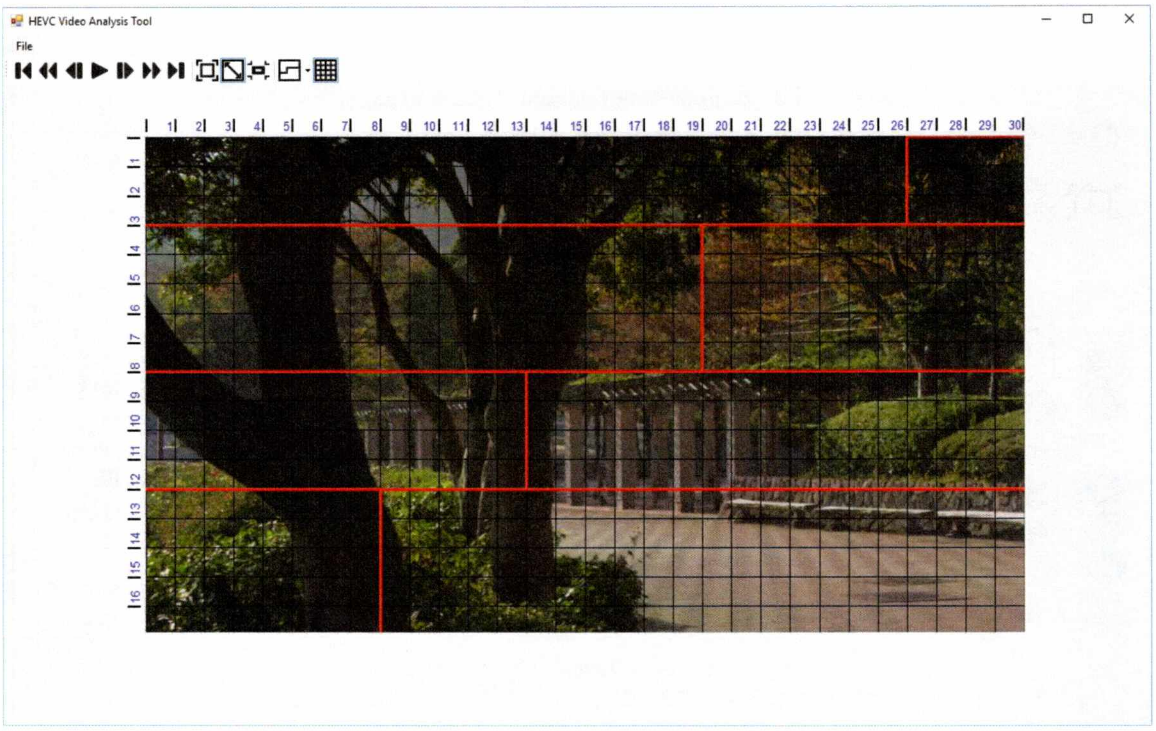


Figure 31. 1st Frame of decoded YUV video "ParkScene"

As can be seen from the figure, the 1st frame has already been partitioned into slices and has already been enabled CTU grid. Furthermore, the video display box has been adjusted to stretched image, that's why the whole frame of video can been shown. The reason is that the keys, which set these options, have been adjusted to be enabled when the application starts. The user is able to turn on both slice and tile partitioning at the same time by checking the appropriate menu buttons. In case slice and tile borders coincide on each other, then only tile borders will be appeared. As Figure 32 shows, tile borders overcome slice borders. That's happens because tile borders has the highest priority and are drawn after all other graphic designs. Second priority has slice borders and last priority has CTU grid. Also, it is noticed from Figure 31 and Figure 32 that slice borders are red-lined, tile border are green-yellow-lined and CTU grid is black-lined.

Figure 33 shows a frame that has disabled both partitioning views and CTU grid. Besides stretched image view in video display box, the application provides 2 other views: the full screen and normal image size. Figure 34 presents the normal image size view, in which it can be observed that size of the video frame is larger than video display box size. The result shows that only 15 entire CTUs columns and 8 entire CTUs rows appears in Figure 34.

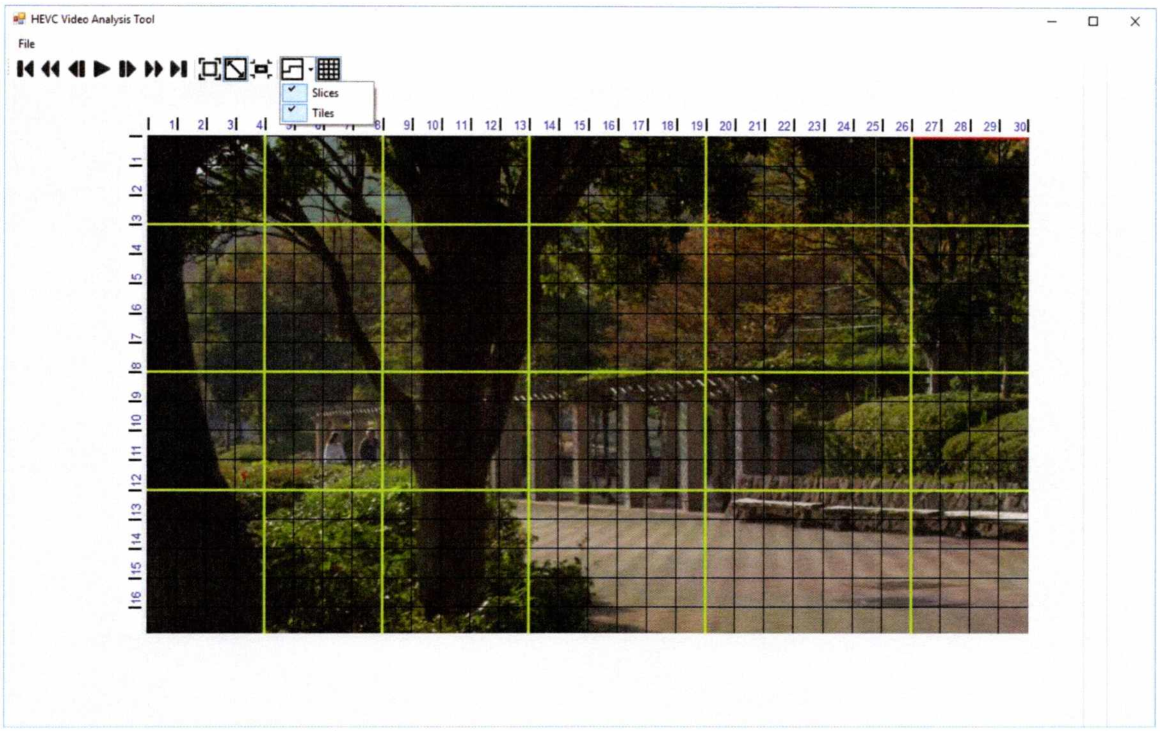


Figure 32. 40th Frame of decoded YUV video "ParkScene"

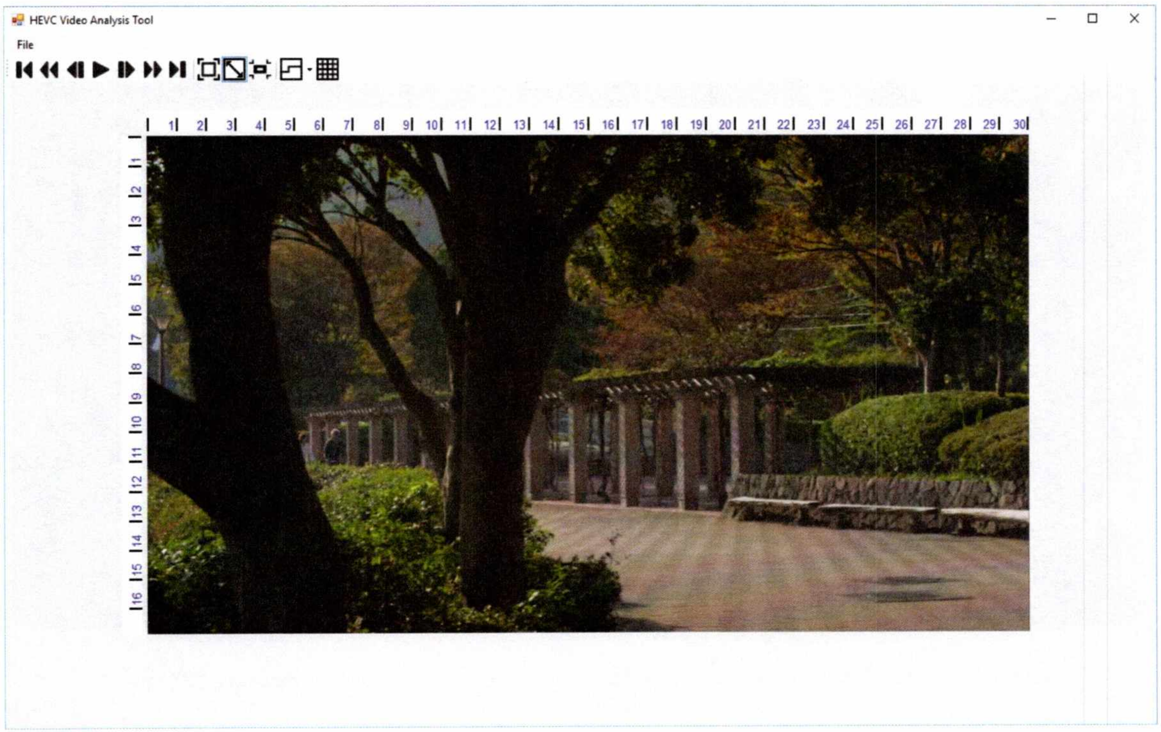


Figure 33. 20th Frame of decoded YUV video "ParkScene" without partitioning view and CTU grid

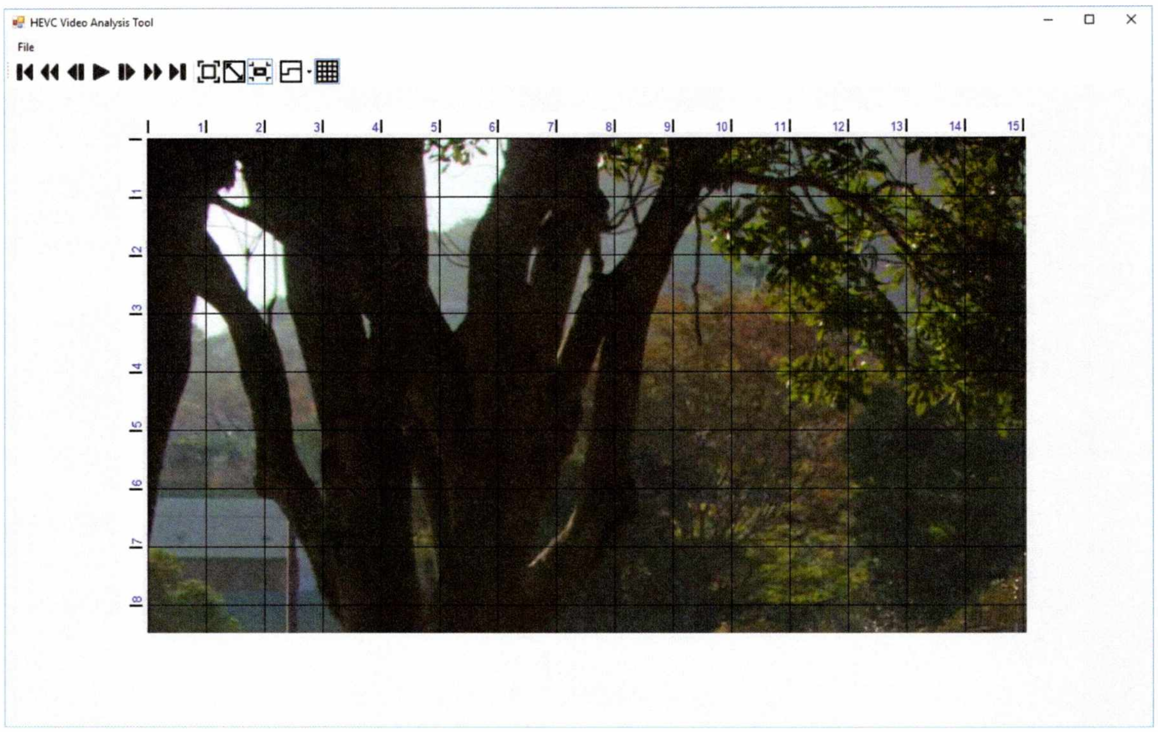


Figure 34. 200th Frame of decoded YUV video "ParkScene" with Normal Image view and CTU grid

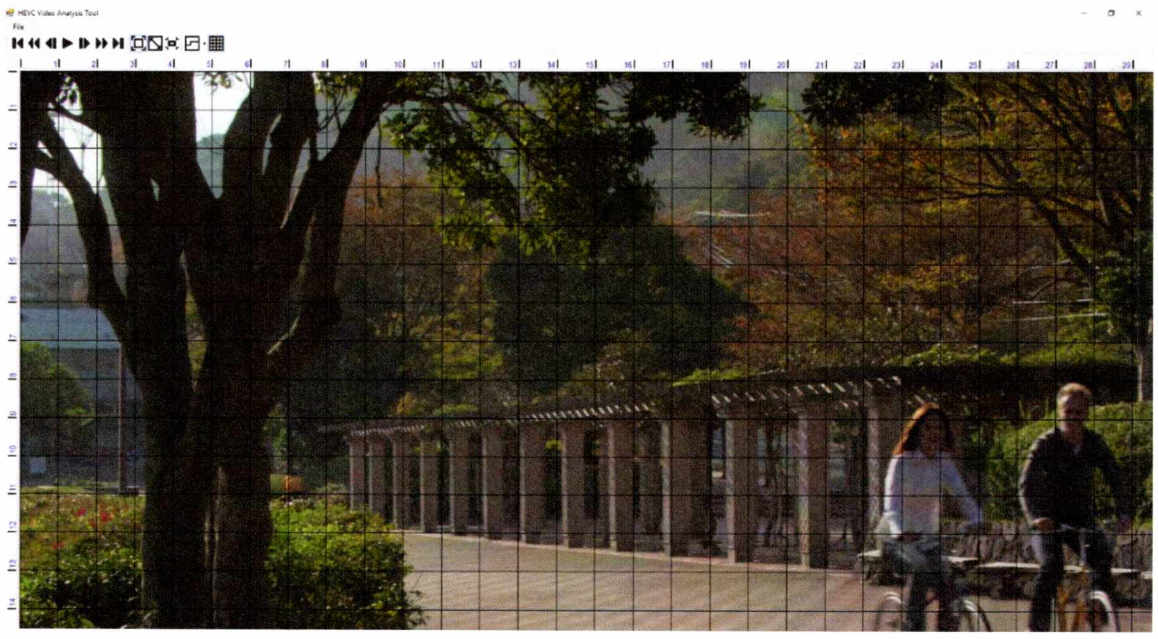


Figure 35. 210th Frame of decoded YUV video "ParkScene" with Full Screen view and CTU grid

Figure 35 shows the full screen view, where the video display box is expanded to whole size of the form and also windows form is maximized in order to fit the full resolution of the video. Full screen view depends on video resolution and computer screen resolution. So, if resolution of video is higher than screen resolution, then the application won't be able to appear all pixels of video in the screen. Another thing that isn't mentioned at previous figures is the horizontal ruler and the vertical ruler that measure CTUs columns and CTUs rows in the frame, respectively. At last, the user is capable of closing the video by clicking on “File” folding menu and then on “Close” button. Additionally, the software can decode and play a video after closing another one. The application can be terminated either by clicking on the top right button (X) or by clicking on “File” folding menu and then on “Exit” button.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

Within the framework of this master thesis was the implementation of "HEVC Video Analysis Tool" software application. The goal of this project was to develop an integrated software that its basic tasks are to decode a compressed video file, to design a YUV video player, to demonstrate the slice and tile partitioning of HEVC decoded video, as well as to show the CTU grid.

For developing this software application theoretical background in HEVC video standard and experience in programming were necessary. In order to help the reader to have a complete understanding of the results of the presented application, a brief overview about video coding and the evolution of video standards through the years is presented. Specifically, we focused on HEVC standard, in which all encoder and decoder operations are analyzed, and particularly reference was made to slice and tile partitioning, which constitutes the basic operation of "HEVC Video Analysis Tool" software application.

The implementation of "HEVC Video Analysis Tool" software application required the usage of HEVC encoder and decoder, as well as other programs. The programs that were used are: Microsoft Visual Studio 2015 for designing the graphics user interface (GUI), Notepad++ for editing parameters in configuration file and Command Prompt for executing HEVC encoder and decoder. Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 provided the HEVC encoder and decoder that were prerequisites for programming the software and taking the results. The combination of all these tools had the result of developing a reliable, convenient, flexible, expandable and user-friendly software. Program bugs have been largely assimilated and the user can ascertain that the program works extremely well.

Future work includes the extension of "HEVC Video Analysis Tool" software application with a numerous of new functions, that would satisfy the needs of more demanding users. Such extensions could include (but not limit to) the designing of a panel that will inform the user about number and type of frames, number of CTUs, slices and tiles in each frame etc., as well as the partitioning of CTUs into Coding Units (CUs) and CUs into Prediction Units (PUs) and Transform Units (TUs). Last but not least extensions providing deep statistical analysis of the decoded video could help researchers activated to the field of video coding.

REFERENCES

[1] ITU-T Rec. H.265 and ISO/IEC 23008-2 (2013) High efficiency video coding. Final draft approval Jan. 2013 (formally published by ITU-T in June, 2013, and in ISO/IEC in Nov. 2013)

[2] ITU-T Rec. H.264 and ISO/IEC 14496-10 (2003) Advanced video coding (May 2003 and subsequent editions)

[3] Sruthi S., Dr. Shreelekshmi R., "Video Compression – from Fundamentals to H.264 and H.265 Standards", International Journal Of Engineering And Computer Science ISSN:2319-7242, Vol. 4 Issue 7, Page No. 13468-13473, July 2015.

[4] Rao K. R., Kim D. N., Hwang J. J., "Video Coding Standards and Video Formats", Signals and Communication Technology, ISBN: 978-94-007-6741-6, Chapter 2, pp. 449., 2014

[5] Vivienne Sze, Madhukar Budagavi, "Design and Implementation of Next Generation Video Coding Systems (H.265/HEVC Tutorial)", ISCAS Tutorial, 2014.

[6] Vivienne Sze, Madhukar Budagavi, Gary J. Sullivan, "High Efciency Video Coding (HEVC) - Algorithms and Architectures", Springer, 2014

[7] Iain E. Richardson , "The H.264 Advanced Video Compression Standard, A John Wiley and Sons, Ltd., Publication, UK, 2nd Edition, 2010

[8] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[9] F. Bossen, "Common Test Conditions and Software Reference Configurations", document JCTVC-H1100, JCT-VC, San Jose, CA, Feb. 2012.

[10] Papadopoulos Panagiotis, "Efficient Slice and Tile Based Parallelization af Video Encoding in HEVC", Master Thesis, Department of Computer Science, University of Thessaly, Greece, March 2017

[11] H. Migallón, P. Piñol, O. López-Granado, V. Galiano1, M. P. Malumbres, "Performance analysis of frame partitioning in parallel HEVC encoders", Springer Science and Business Media New York, 10 November 2016

[12] HM 16.15 reference software. http://hevc.hhi.fraunhofer.de

[13] x265 HEVC encoder. http://x265.org

[14] Microsoft Visual Studio 2015. https://www.visualstudio.com/

[15] openCV Open Source Computer Vision Library. https://opencv.org/

[16] T. Von Roden, "H.261 and MPEG1-a comparison", IEEE, Praktische Inf. IV, Mannheim Univ., Germany, March 1996.

[17] Yuen-Wen Lee, "Efficient MPEG-2 encoding of interlaced video", IEEE, Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Vol. 23, Issue: 1-2, pp. 61 – 67, Jan.-April 1998.

[18] Lajos Hanzo, Peter Cherriman, Jurgen Streit, "Comparative Study of the H.261 and H.263 Codecs", Wiley-IEEE Press, 1st Edition, pp. 295 – 337, 2008

[19] Brian W. Kernighan, Dennis Ritchie, "C Programming Language", Prentice Hall Software Serries, 2nd Edition, March 1988.

[20] Robert Lafore , "Object-Oriented Programming in C++", Pearson Education, 4th Edition, 18 Dec. 1997.

[21] Julian Templeman, "Microsoft Visual C++/CLI Step by Step", Pearson Education, 15 August 2013.

[22] "Common Language Runtime (CLR)", MSDN Library, 14 November 2013.

[23] C. C. Chi, M. A. Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1827-1838, Dec. 2012.

[24] Command Prompt. https://www.computerhope.com/issues/chusedos.htm.

[25] P. Piñol, H. M. Gomis, O. M. L. Granado, and M. P. Malumbres, "Slice-based parallel approach for HEVC encoder," Journal of Supercomputing, vol. 71(5), pp. 1882-1892, 2015

[26] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, “An overview of tiles in HEVC,” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 969-977, Dec. 2013.

[27] Ahmet Kondoz ,“Scalable Video Coding”, Wiley Telecom, 1st Edition, 2009.

[28] Wen-Hsiao Peng, Frederick G. Walls, Robert A. Cohen, Jizheng Xu, Jörn Ostermann, Alexander MacInnis, Tao Lin, “Overview of Screen Content Video Coding: Technologies, Standards, and Beyond”, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, Vol. 6, Issue 4, pp. 393 - 408 December 2016.

[29] Miska M. Hannuksela, Ye Yan, Xuehui Huang, Houqiang Li, “Overview of the multiview high efficiency video coding (MV-HEVC) standard”, Image Processing (ICIP), 2015 IEEE International Conference on mage Processing (ICIP), 27-30 Sept. 2015.

[30] M. Koziri, P. Papadopoulos, N. Tziritas, A.N. Dadaliaris, T. Loukopoulos, and S.U. Khan, “Slice-Based Parallelization in HEVC Encoding: Realizing the Potential Through Efficient Load Balancing,” Proc. 18th Int. Workshop on Multimedia Signal Processing (MMSP 2016), IEEE, Montreal, Canada, Sept. 2016, pp. 1-6.

[31] P. Papadopoulos, M.G. Koziri, N. Tziritas, T. Loukopoulos, I. Anagnostopoulos, and G.I. Stamoulis, “Performance Evaluation of Batch Encodings in HEVC Using Slice Level Parallelism,” Proc. 20th Panhellenic Conf. on Informatics (PCI 2016), ACM, Patras, Greece, Nov. 2016, no. 70.

[32] M. G. Koziri, P. K. Papadopoulos, N. Tziritas, T. Loukopoulos, S. U. Khan, and A. Y. Zomaya, “Efficient Cloud Provisioning for Video Transcoding: Review, Open Challenges and Future Opportunities,” IEEE Internet Computing (IC), 2017.

[33] M. Koziri, P. K. Papadopoulos, N. Tziritas, N. Giachoudis, T. Loukopoulos, S. U. Khan, and G.I. Stamoulis, “Heuristics for Tile Parallelism in HEVC,” Proc. 25th European Signal Processing Conf. (EUSIPCO 2017), IEEE, Kos, Greece, Aug. 2017, pp. 1514-1518.

[34] M.G. Koziri, P. Papadopoulos, N. Tziritas, A.N. Dadaliaris, T. Loukopoulos, S.U. Khan, and C.-Z. Xu, “Adaptive Tile Parallelization for Fast Video Encoding in HEVC,” Proc. 12th Int. Conf. on Green Computing and Communications (GreenCom 2016), IEEE, Chengdu, China, Dec. 2016, pp. 738-743.

D
621.388
DHM