Heterogeneous Transcoding for Next Generation Multimedia Video Codecs for Efficient Communication

by

SHREYANKA SUBBARAYAPPA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2020

Heterogeneous Transcoding for Next Generation Multimedia Video Codecs for Efficient Communication

The members of the Committee approve the doctoral dissertation of SHREYANKA SUBBARAYAPPA

Supervising Professors:

Dr. K R Rao

rao@uta.edu

Dr. Jonathan W Bredow

jbredow@uta.edu

Dr. Michael T Manry

manry@uta.edu

Dr. Venkat Devarajan

venkat@uta.edu

Dr. B Ramaswamy Karthikeyan

karthikeyan.ec.et@msruas.ac.in

Graduate Advisor:

Dr. William E. Dillon

dillon@uta.edu

ACKNOWLEDGEMENTS

I take this opportunity to offer my gratitude to my supervisor, Dr. K. R. Rao, who invested his precious time in guiding me and has been a constant and continuous support throughout my thesis with his patience and profound knowledge. My sincere gratitude and love go to my family and this thesis is dedicated to my parents, Prof. H. Subbarayappa (father) and Mrs. Anasuya Subbarayappa (mother) who have been the main source and motivation in pursuing my Doctorate degree. My sincere thanks to my husband Mr. Adarsh K. and son Ashrith Adarsh for encouraging me and being the emotional and moral support during the hard times when I had to stay away from home and concentrate towards completion of my doctoral degree. I love you all and I thank you all whole heartedly for being by my side all the time.

I would extend my thanks to my graduate advisors, Dr. William E. Dillon, Dr. Jonathan W Bredow, Dr. Michael T Manry, Dr. Venkat Devarajan and Dr. B. Ramaswamy Karthikeyan for taking out their time to serve in my doctoral committee.

I would also like to extend my appreciation to my university in India, Ramaiah University of Applied Sciences, where the foundation of my doctoral research developed and would like to thank all my colleagues from Electronics and Communication Department for supporting me in pursuing my Ph.D. far away from home.

I am grateful to all the teachers and professors who taught me during the years I spent at school/ university, in India as well as United States. Last but not the least, I would like to thank all my friends and relatives in India and United States for being my support and encouraging me, especially Spoorthi Manjunath for her help during my stay at Texas and being a family to me, during the time I was away from home.

November 5th, 2020

ABSTRACT

HETEROGENEOUS TRANSCODING FOR NEXT GENERATION MULTIMEDIA VIDEO CODECS FOR EFFICIENT COMMUNICATION

Shreyanka Subbarayappa, Ph.D.

The University of Texas at Arlington, 2020

Supervising Professor: K.R. Rao

Innovations in the communication systems and technology are growing tremendously and the growth seen is unimaginable in the last forty years. In multimedia communication systems, technology has transformed from analog television to digital television in the video domain. Mobile phones are known as smart phones as they are used, not only to make voice calls, but also used to send emails, video calls, transfer data, GPS, taking pictures and so on. Due to the wide spread user applications, compression on data becomes important to save system resources. Video has occupied 75% of major traffic of data transfer and is expected to cover 80% by 2021 [4]. Video is also continuously increasing in size from standard-definition (SD) to ultra-high definition (UHD - 4K, 8K and 12K) video. More data or size in video requires higher transmission bandwidth or more disk space to store, which is very expensive. This drives into the betterment in compression and hence a demand for a new codec. Several algorithms are implemented to achieve compression of Image or Video with respect to the user's demand on the quality of output as well as application it is used for. These algorithms working together are classified in terms of codecs and we use different codecs for different applications.

Advances in video compression technology are used to reduce the utilization of system resources, like processing time, memory use, network bandwidth and battery life. This is possible by reducing the complexity of the video codecs without compromising on the output video quality [2][3][5]. There are two distinct lines in the future video coding technology development work. EVC driven by MPEG team [126] and Versatile Video Codec (VVC) [7][8] driven by Joint Video Exploration Team (JVET). These codecs are the extended versions with advances in compression technologies when compared to the prior video codecs or reference codecs HEVC and H.264.

This thesis is fundamentally and entirely devoted to the theory and design of different algorithms used in present and future video codecs to obtain efficient implementation and reconstruction of codec outputs. It also addresses the most recent codecs being developed, i.e. EVC and VVC. Transcoding being one of the main applications in video technologies is implemented and studied between these four codecs.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ACRONYMS

GPS	Global positioning system
SD	Standard Definition
UHD	Ultra High Definition
AOM	Alliance for open media
VVC	Versatile video coding
JVET	Joint video exploration team
HEVC	High efficiency video coding
CD	Compact disc
DVD	Digital versatile disc
3D	3 dimentional
HVS	Human visual system
Fps	frames per second
QCIF	Quarter common intermediate format
SDO	Standards development organization
ISO	International standardization organization
IEC	International electrotechnical commission
ITU-T	Telecommunication standardization sector of the international telecommunication union
MPEG	Moving picture expert group
EVC	Essential video coding
SMPTE	Society of motion picture and television engineers
AV1	AOMedia video 1
AVS China	Audio video coding standard workgroup of china
BBC	British broadcast company
DCT	Discrete cosine transform
IDCT	Integer discrete cosine transform
PSNR	Peak signal to noise ratio
SSIM	Structural similarity index metric
MSE	Mean square error
BD-PSNR	Bjontegaard metric
BD-SSIM	Bjontegaard metric
QP	Quantization parameter
GOP	Group of pictures
BD-Rate	Bjontegaard metric
VTM	Video test model
IPR	Intellectual Property Right
HM	Hybrid Model
AVC	Advanced Video Coding
JVT	Joint Video Team
JM	Joint model

QTBT	Quad tree binary tree
TT	Ternary tree
CTU	Coding tree unit
DC	Discrete cosine
SAO	Sample adaptive offset
CABAC	Context adaptive binary arithmetic coding
CAVLC	Context adaptive variable length coding
MC	Motion compensation
VLD	Variable length decoding
RD	Rate distortion

1.1. Importance of compression

The uncompressed data in multimedia (graphics, audio, images, video and text) [4] needs a huge amount of storage space and the transmission bandwidth availability. In spite of fast progress in the processor speeds and system performance in digital communications, the demand in the data transmission usage bandwidth and data storage capacity surpasses all the abilities of the available technologies. In the recent times, the recent growth in multimedia-based web applications for data intensive contents has achieved the requirements for attaining better ways to encode the signals, videos and images but has achieved compression of such contents to storage and use in the communication technology. Image/video compression reduces the size of the file in bytes without degrading the image quality to an unacceptable level. The reduced size of the file permits more images or videos to be stored in a given amount of memory space or disk. It also helps in reducing the time taken to transmit the image/video over the Internet or download them from Web pages.

1.2. Objective of Image Compression

Images comprise of large amounts of information which also require a lot of storage space, larger transmission bandwidths and longer transmission times. The advantageous factor to compress an image is to preserve only the essential information required to reconstruct the image and also takes less storage space to store on CDs, DVDs or Blu Ray Discs or less bandwidth to transmit it through any medium. An image can be viewed as a matrix of pixel (intensity) values. In order to compress an image, the redundancies are exploited, for example, places or areas where there is no change or little change among pixel values. Hence the images with large areas of uniform colour have large redundancies. Conversely, images having frequent and large colour changes are less redundant and difficult to compress.

1.3. Methods of Data Compression [1]

Data compression is a technique used for reducing the information or content of the data which is useful to be stored easily or reconstructed quicker. There are two methods of data compression we observe in our everyday usage. One is lossless compression and the other is lossy compression which differs with respect to the reconstructed data quality and the size of data compressed.

1.3.1. Lossless Compression [2]

This class of data compression involves algorithms that allow the original data to be obtained or reconstructed from the compressed data. The lossless compression is used when the requirement of the reconstructed data be identical to the original data, or when no assumptions are required to be made for certain deviations which are not critical. Lossless compression is mostly used for medical imaging studios and text where every character and data are important. Typical examples for lossless compression are executable programs and source codes. The Figure 1.1. shows the comparison of lossless compression and lossy compression.

1.3.2. Lossy Compression [3]

This class of data compression is also known as ‘data encoding’ which compresses the data by discarding or losing some of it. The algorithms in lossy compression aim at minimizing the amount of data needed and to be handled, held or transmitted. This compression is used for all forms of multimedia data (images, video, text, graphics and audio), mostly used in applications such as internet telephony and streaming media domains where slight data loss is still acceptable. As an example, a picture is converted to a digital file by considering its data as an array of dots. These dots are known as pixels and the image has the color and brightness of each pixel. If the entire picture has the same color as blue, it can be compressed without any loss by considering it as “200 blue dots” instead of taking it as “blue dot, blue dot, (198 more times)..”.

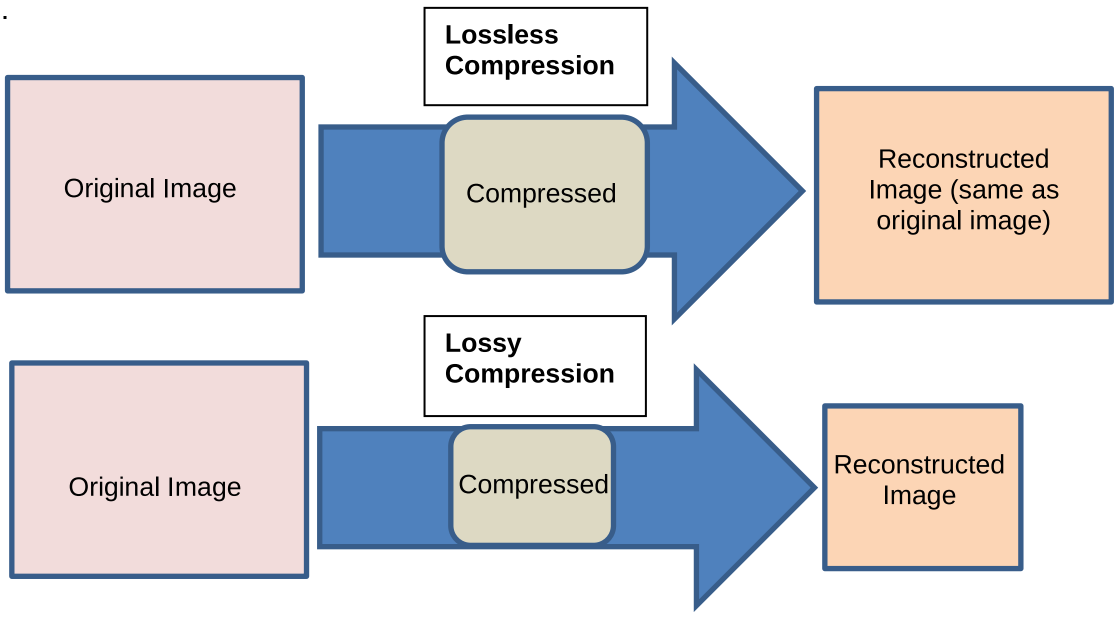


Figure 1.1. Comparison of Lossless and Lossy Compression

1.4. Significance of Video

Universal, high quality and seamless video has been the ultimate goal for researchers, standard bodies and companies over the last four decades, Few areas like consumer video storage and broadcast television, video mail, mobile video and video conferencing have clearly captured the market, while the screen content video, 3D video and 360° video with increase in the demand for video quality is increasing the market share day by day [5][6]. Nevertheless, there is certainty that digital video will continue to infuse the homes, networks and businesses and will remain to be the most important industry globally. The digital video industry is constantly evolving and is driven by a lot of commercial and the technical forces. The profitable commercial drive is due to the huge revenue of persuading businesses, industry and consumers to replace analog systems and older digital systems with efficient, new and high-quality digital products and to approve and adopt new entertainment and communication products. Technical drive is due to continuous improvements in dealing with performance, the ability for greater capacity storage, transmission mechanism along with research and development in the field of image and video processing technology. Figure 1.2. is an example of a home media eco system and the importance of video transmission.

Figure 1.2. Home media eco-system

1.5. Implication of video compression and its standardization

Obtaining a digital video from any source (camera or a saved video clip) and transmitting it to its destination (display) require a chain of processes and components. Main key to this chain is the process of compression (also known as encoding) and decompression (also known as decoding). These processes are used to reduce a bandwidth-intensive digital 'raw' video to a manageable size used for transmission in a medium or a storage and then reconstructing it back for display. Having the compression and decompression phenomenon ‘right’ can provide a major commercial and technical edge to a product, by getting higher image quality, more flexibility and greater reliability than competing solutions. Hence there is always a keen interest in continuing development and enhancement of video compression and decompression systems and methods.

Video compression also comprises of lossy compression. Temporal as well as spatial redundancies are video characteristics. Removing these redundancies and some lossy techniques like transform, scaling and Human Visual System facilitate very high compression. Table 1.1 gives the detailed explanation on compression methods achieved.

INFORMATION TYPE	**COMPRESSION TOOL**
Spatial Redundancy	Intra frame coding
Temporal Redundancy	Inter frame coding
High Frequency image coefficients	Transform and scaling
Bit-Stream Redundancy	Entropy coding
Human Visual System (HVS)	Perceptual coding

Table 1.1. Compression Methods

To calculate a raw uncompressed video, consider a video frame of size 720X486 (Standard Definition video). Each frame has 720X486 = 349920 pixels or about 0.3 MPixels. If we display our video in color format, we use three color channels (Red-Green-Blue) to represent each pixel. Hence to get the size of an uncompressed video frame, we multiply by 3 to the number of pixels of the frame (i.e 0.3 X3 = 0.9Megabytes). To calculate the size of a video to be played for 1 second, we multiply the frame rate to video frame byte size (i.e if number of frames per second (fps) = 30fps, then 30X0.9 Mbytes = 27 MB data per second of video). This huge amount of data directs us to the importance of compression since bandwidth usage and storage size play a crucial role.

Another example for calculating the video size in bits per second for one second video by taking QCIF data from Table 1.2. Resolution of QCIF being 176X144, we multiply this to the bitrate that is 30fps (i.e. 176X144X30 = 0.7 Kbps). Since each pixel is represented by 8 bits, we multiply 0.7 by 8 which gives us nearly 6 Mbps. This video is represented in the YUV format as 4:2:0 as shown in Figure 1.3., which tells us that we have 4 parts of Y (Luma), 1 part of U (Chroma) and 1 part of V (Chroma) samples present. This tells us that out of 3 channels we have only 1.5 channel, hence dividing this factor to 6Mbps (i.e

6Mbps X 1.5 = 9Mbps). A raw video is classified in terms of width and height of each content. This can be viewed in the Table 1.2. where each content type has a code and its bitrate is shown for the raw content with 30fps frame rate, with bit depth 8 bits/pixel and 4:2:0 video format of the content.

Code	Width	Height	Description	Bit rate @ 30 fps, 8 bits/pixel, 4:2:0 format
QCIF	176	144	Quarter CIF	9 Mbps
QVGA	320	240	Quarter Video Graphics Array	27 Mbps
CIF	352	288	Common Intermediate Format	36 Mbps
HVGA	640	240	Half Video Graphics Array	55 Mbps
VGA	640	480	Video Graphics Array	110 Mbps
SD	720	486	Standard Definition	125 Mbps
SVGA	800	600	Super Video Graphics Array	172 Mbps
XGA	1024	768	Extended Graphics Array	283 Mbps
XGA+	1152	768	Extended Graphics Array plus	318 Mbps
	1152	864		358 Mbps
SXGA	1280	1024	Super Extended Graphics Array	471 Mbps
SXGA+	1400	1050	Super Extended Graphics Array plus	529 Mbps
UXGA	1600	1200	Ultra Extended Graphics Array	691 Mbps
HD	1920	1080	High definition	746 Mbps
QXGA	2048	1536	Quad Extended Graphics Array	1.1 Gbps
UHD 4K	3840	2160	Ultra High Definition 4K Video	2 Gbps
4K	4096	2160	4K Video	3.2 Gbps
5K	5120	2700	5K Video	5 Gbps
6K	6144	3160	6K Video	7Gbps

UHD 8K	7680	4320	Ultra High Definition 8K Video	11 Gbps
8K	8192	4320	8K Video	12.7 Gbps

Table 1.2. Raw bitrates of uncompressed video

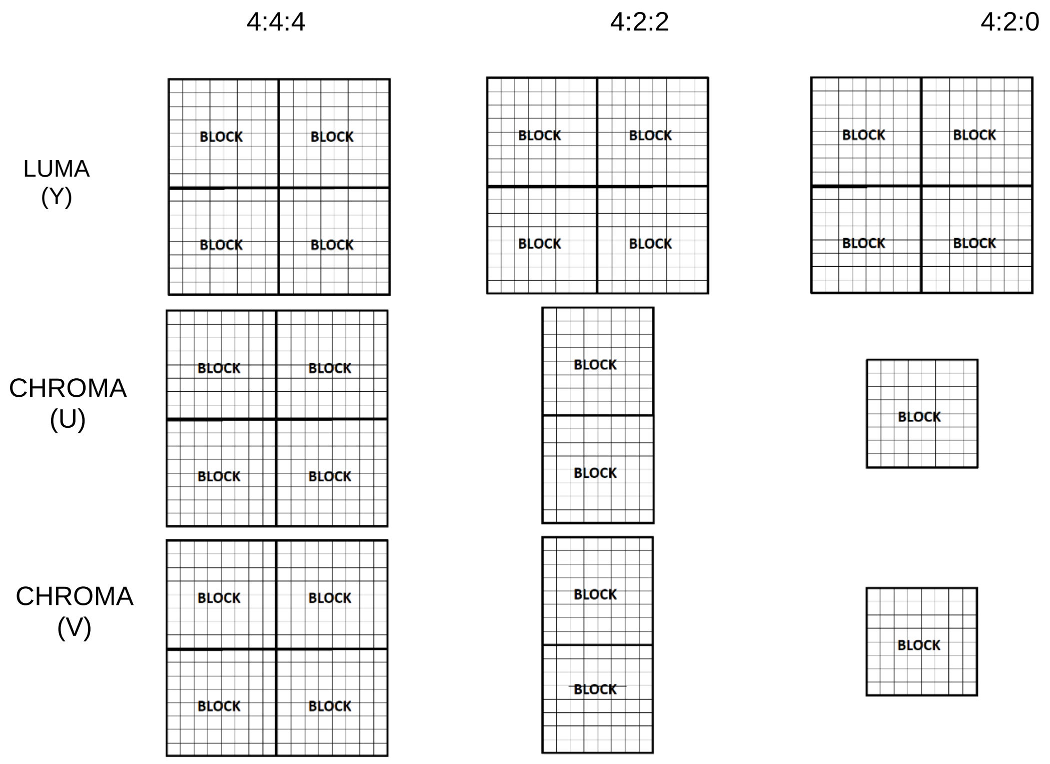


Figure 1.3. YUV formats for one macro block of an image

Multimedia is targeted for a wide range of applications from mobile video broadcast, video on demand, video conferencing and medical applications. Having such a wide range of applications, it is essential to standardize video compression. Standardization ensures implementation of compression technique from different vendors thus enabling end-users to make a choice to access from different video services for both software and hardware. Numerous video compression standards are developed for both open source and proprietary based depending on the user's application and end-usage [7][8].

1.6. History of Video Codecs

Multimedia is a blend of many sources like text, images, audio and video. The need and requirement for communication in this domain has become essential in today's fast-moving era. Hence the necessity for compression of multimedia sources for better performance and usage by the end-user. Compression of data is also known as Encoding or Encoder and decompression of data is known as Decoding or Decoder. The system consisting of Encoder and Decoder is known as Codec. There are plenty of algorithms developed in the video and audio domains whereas only some of them are standardized to form Codecs. Many audio and video coding standards are developed and still development is ongoing for future standards by SDO (Standards Development Organizations) and ISO/IEC (International Standardization Organization and the International Electrotechnical Commission).

Presently, the main organizations working for standardizing video compression are ITU-T, ISO, SMPTE, On2Google and AOMedia as shown in Figure 1.4.

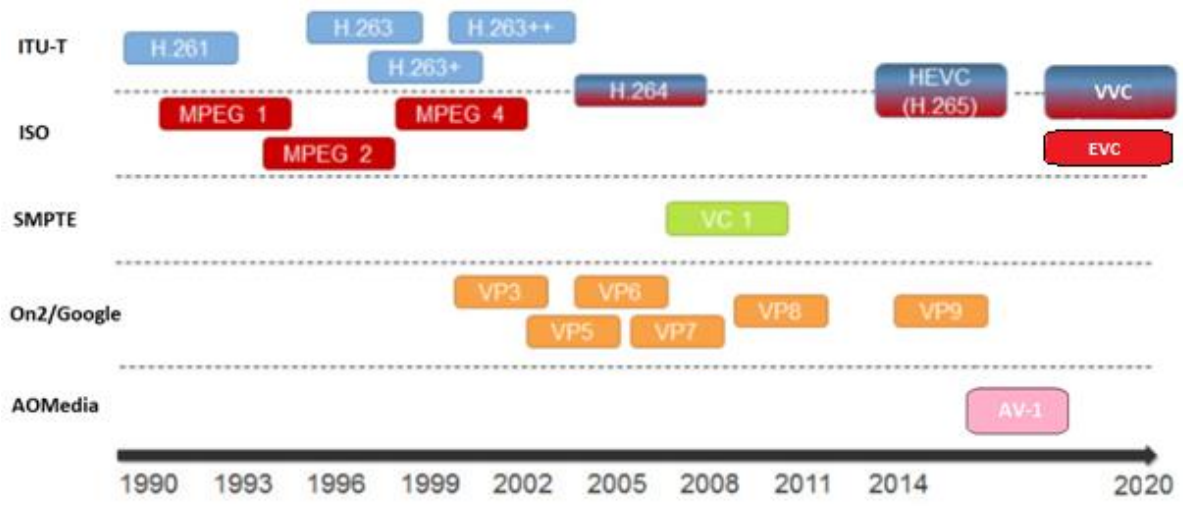


Figure 1.4. History of Video Codecs

ITU-T (Telecommunication Standardization Sector of the International Tele-communication Union) organization solely developed the ‘H’ series starting from H.261 in 1990 to H.263 in 2000 [9][10][11]. During the same time, another organization called ISO developed the MPEG standards. The MPEG series ranged from MPEG1 to MPEG4 with their applications similar to ‘H’ series [12][13]. Essential Video Coding (EVC) standard which is under development by the MPEG team of ISO/IEC JTC 1/SC 29/WG 11 is due by end of 2020. SMPTE (Society of Motion Picture and Television Engineers) focused mainly on VC-1 codec. Google came up with the ‘VP’ codec series, ranging from VP3 to VP9[14][15]. In 2005, ITU-T and ISO organizations merged to develop codecs like H.264 in 2005, HEVC [11] in 2014 and VVC [16][17] in 2020. AOMedia (Alliance for Open Media) developed AV-1[18][19] in 2015 which is a successor of VP9 [19]. There were other codecs developed as AVS China (Audio and Video coding Standard of China) [20], DIRAC [20] by BBC and company codecs by Microsoft and Real Networks. Each of these codecs use compression techniques depending on the type of application they are designed for.

2.1. Introduction

One of the most efficient and the most used video coding standard which was introduced in 2003 by the Joint Video Team (JVT) was called H.264/MPEG4-Part10 or advanced video coding (AVC) [21]. This codec was developed by Video Coding Experts Group (VCEG) of International Telecommunication Union – Telecommunication standardization sector (ITU-T) and Moving Picture Experts Group (MPEG) of ISO/IEC jointly known as Joint Video Team (JVT). This codec had a wide range of application usages from both non-interactive and interactive domains such as video telephony as the interactive and video on demand, broadcast, storage and streaming as non-interactive which constitute to network friendly video vision. H.264 is the extended video codec of H.263 [23] with a lot of additional tools which gives it more compression efficiency and better output quality video.

The joint video team built many extensions to the existing video codec which was known as FRExt or fidelity range extensions. Having these add-ons improved the output video quality along with enabling higher sample bit depth precision and color information having higher resolution. FRExt also includes higher YUV sampling structures as 4:2:2 and 4:4:4 as shown in Figure 1.3. Additional features such as integer transforms switching from 4x4 to 8x8 and vice versa, perceptual based quantization matrix, efficient lossless coding technique for inter-picture along with additional color space support were also added.

Scalable video coding (SVC) [23] allows the construction of bitstreams that contain sub-bitstreams that conform to H.264/AVC. For temporal bitstream scalability, i.e., the presence of a sub-bitstream with a smaller temporal sampling rate than the bitstream, complete access units are removed from the bitstream when deriving the sub-bitstream. In this case, high-level syntax and inter prediction reference pictures in the bitstream are constructed accordingly. For spatial and quality bitstream scalabilities, i.e. the presence of a sub-bitstream with lower spatial resolution or quality than the bitstream, network abstraction layer (NAL) units are removed from the bitstream when deriving the sub-bitstream. In this case, inter-layer prediction, i.e., the prediction of the higher spatial resolution or quality signal by data of the lower spatial resolution or quality signal, is typically used for efficient coding. The scalable video coding extension was completed in November 2007 [17].

The next major feature added to the standard was Multiview Video Coding (MVC). MVC enables the construction of bit streams that represent more than one view of a video scene. An important example of this functionality is stereoscopic 3D video coding. Two profiles were developed in the MVC work: one supporting an arbitrary number of views and designed specifically for two-view stereoscopic video. The Multiview Video Coding extensions were completed in November 2009 [24].

Like any other previous motion-based codecs, it uses the following basic principles of video compression [1]:

- Transform for reduction of spatial correlation
- Quantization for control of bit rate
- Motion compensated prediction for reduction of temporal correlation
- Entropy coding for reduction in statistical correlation.

The improved coding efficiency of H.264 can be attributed to the additional coding tools and the new features. Listed below are some of the improved techniques used in H.264 for the first time [25]:

- Adaptive intra-picture prediction
- Small block size transform with integer precision
- Multiple reference pictures and generalized B-frames
- Quarter pixel precision for motion compensation
- Variable block size
- Content adaptive in-loop deblocking filter and
- Improved entropy coding by introduction of CABAC (context adaptive binary arithmetic coding) and CAVLC (context adaptive variable length coding).

The increase in the coding efficiency and increase in the compression ratio result in a greater complexity of the encoder and the decoder algorithms of H.264, as compared to previous coding standards. In order to develop error resilience for transmission of information over the network, H.264 supports the following techniques [30]:

- Flexible macroblock ordering (FMO)
- Switched slice
- Arbitrary slice order (ASO)
- Redundant slice (RS)
- Data partitioning (DP)
- Parameter setting

2.2. Profiles of H.264

The H.264/AVC standard is composed of a wide range of coding tools. Also, the standard addresses a large range of bit rates, resolutions, qualities, applications and services. Not all the tools and all the bit rates are required for any given application at a given point of time. All the various tools of H.264 are grouped in profiles.

Profiles are defined as a subset of coding tools. They help to maximize the interoperability while limiting the complexity. Also, the various levels define the various parameters like size of decoded pictures, bit rate, etc.

The profiles defined for H.264 can be listed as follows [25]:

- Constrained baseline profile
- Baseline profile
- Main profile
- Extended profile
- High profile
- Progressive high profile
- Constrained high profile
- High 10 profile
- High 4:2:2 profile
- High 4:4:4 predictive profile
- High stereo profile
- High 10 intra profile
- High 4:2:2 intra profile
- High 4:4:4 intra profile
- CAVLC 4:4:4 intra profile
- Scalable baseline profile
- Scalable high profile
- Scalable high intra profile 18
- Scalable constrained high profile
- Stereo high profile
- Multiview high profile

Figure 2.1 illustrates the coding tools for the various profiles of H.264. The standard defines 21 sets of capabilities, which are referred to as *profiles*, targeting specific classes of applications.

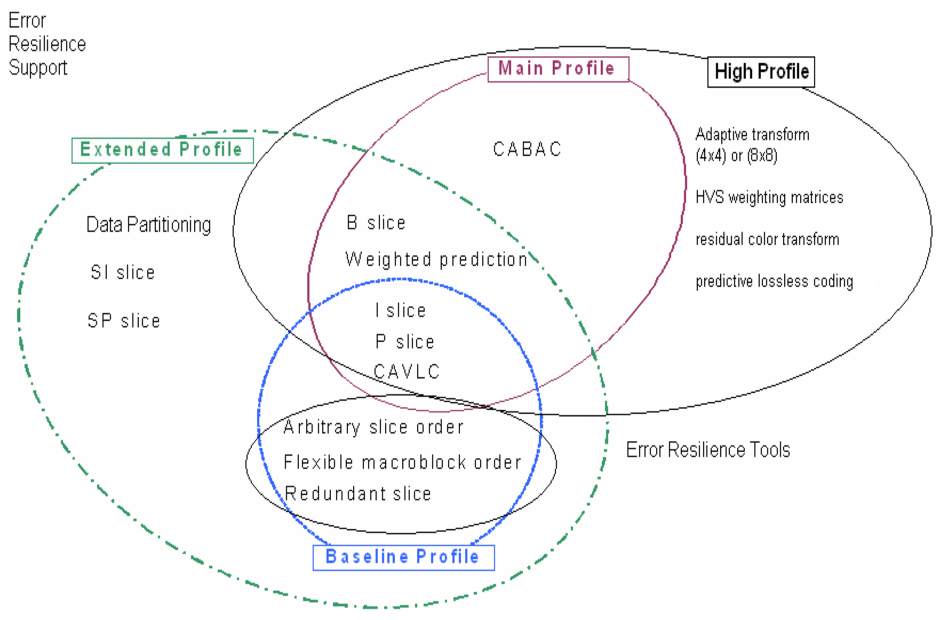


Figure 2.1. Different profiles in H.264 with distribution of various coding tools among the profiles

2.2.1. Baseline profile:

The list of tools included in the baseline profile are I (intra coded) and P (predictive coded) slice coding, enhanced error resilience tools of flexible macroblock ordering, arbitrary slices and redundant slices. It also supports CAVLC (context-based adaptive variable length coding). The baseline profile is intended to be used in low delay applications, applications demanding low processing power, and in high packet loss environments. This profile has the least coding efficiency among all the three profiles.

2.2.2. Main profile:

The coding tools included in the main profile are I, P, and B (bi directionally predictive coded) slices, interlace coding, CAVLC and CABAC (context-based adaptive binary arithmetic coding). The tools not supported by main profile are error resilience tools, data partitioning and SI (switched intra coded) and SP (switched predictive coded) slices. This profile is aimed to achieve highest possible coding efficiency.

2.2.3. Extended profile:

This profile has all the tools included in the baseline profile. As illustrated in the Figure 2.1 this profile also includes B, SP and SI slices, data partitioning, interlaced frame and field coding, picture adaptive frame/field coding and macroblock adaptive frame/field coding. This profile provides better coding efficiency than baseline profile. The additional tools result in increased complexity.

2.2.4. High profile defined in the FRExts amendment:

In September 2004 the first amendment of H.264/MPEG-4 AVC video coding standard was released [25]. A new set of coding tools were introduced as a part of this amendment. These are termed as ―Fidelity Range Extensions‖ (FRExts). The aim of releasing FRExts is to be able to achieve significant improvement in coding efficiency for higher fidelity material. It also has lossless representation capability: I PCM raw sample value macroblocks and entropy coded transform by-pass lossless macroblocks (FRExts only). The application areas for the FRExts tools are professional film production, video production and high-definition TV/DVD.

The FRExts amendment defines four new profiles (refer Figure 2.2.) [23]:

- High (HP)
- High 10 (Hi10P)
- High 4:2:2 (Hi422P)
- High 4:4:4 (Hi444P)

The other profiles constrained to intra use in H.264 are

- High 10 intra profile
- High 4:2:2 intra profile
- High 4:4:4 intra profile

Figure 2.2. gives us the specification of High profile in H.264.

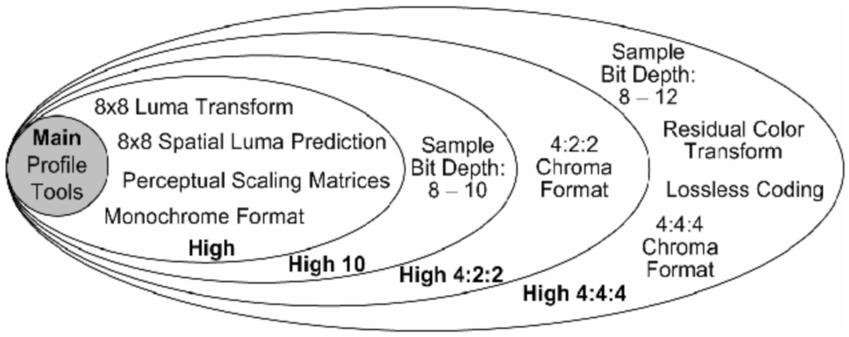


Figure 2.2. Tools introduced in FRExts and their classification under the new high profiles [31]

All four of these profiles build further upon the design of the prior main profile. The main aim behind introducing 8x8 transform in FRExts is that high fidelity video demands preservation of fine details and textures. To achieve this, larger basis functions are required. However, smaller size transform like 4x4 reduces ringing artifacts and reduces computational complexity. The encoder adaptively chooses between 4x4 and 8x8 transforms block sizes.

The transform selection process is limited by the following conditions

- If an inter-coded MB has sub-partition smaller than 8x8 (i.e. 4x8, 8x4, 4x4), then 4x4 transform is used.

- If an intra-coded MB is predicted using 8x8 luma spatial prediction, only 8x8 transform is used.

- Encoder-specified perceptual-based quantization scaling matrices.

The encoder can specify a matrix for scaling factor according to the specific frequency associated with the transform coefficient for use in inverse quantization scaling by the decoder. This allows optimization of the subjective quality according to the sensitivity of the human visual system, less sensitive to the coded error in high frequency transform coefficients [32].

2.3. Levels of H.264

As the term is used in the standard, a "*level*" is a specified set of constraints that indicate a degree of required decoder performance for a profile. For example, a level of support within a profile specifies the maximum picture resolution, frame rate, and bit rate that a decoder may use. A decoder that conforms to a given level must be able to decode all bitstreams encoded for that level and all lower levels.

Level	Maximum decoding speed (macroblocks/s)	Maximum frame size (macroblocks)	Maximum video bit rate for video coding layer (VCL) (Constrained Baseline, Baseline, Extended and Main Profiles) (kbits/s)	Examples for high resolution @ highest frame rate (maximum stored frames)
1	1,485	99	64	128×96@30.9 (8) 176×144@15.0 (4)
1b	1,485	99	128	128×96@30.9 (8) 176×144@15.0 (4)
1.1	3,000	396	192	176×144@30.3 (9) 320×240@10.0 (3) 352×288@7.5 (2)
1.2	6,000	396	384	320×240@20.0 (7) 352×288@15.2 (6)
1.3	11,880	396	768	320×240@36.0 (7) 352×288@30.0 (6)
2	11,880	396	2,000	320×240@36.0 (7) 352×288@30.0 (6)
2.1	19,800	792	4,000	352×480@30.0 (7) 352×576@25.0 (6)

2.2	20,250	1,620	4,000	352×480@30.7 (12) 352×576@25.6 (10) 720×480@15.0 (6) 720×576@12.5 (5)
3	40,500	1,620	10,000	352×480@61.4 (12) 352×576@51.1 (10) 720×480@30.0 (6) 720×576@25.0 (5)
3.1	108,000	3,600	14,000	720×480@80.0 (13) 720×576@66.7 (11) 1,280×720@30.0 (5)
3.2	216,000	5,120	20,000	1,280×720@60.0 (5) 1,280×1,024@42.2 (4)
4	245,760	8,192	20,000	1,280×720@68.3 (9) 1,920×1,080@30.1 (4) 2,048×1,024@30.0 (4)
4.1	245,760	8,192	50,000	1,280×720@68.3 (9) 1,920×1,080@30.1 (4) 2,048×1,024@30.0 (4)
4.2	522,240	8,704	50,000	1,280×720@145.1 (9) 1,920×1,080@64.0 (4) 2,048×1,080@60.0 (4)

Table 2.1. Levels with maximum H.264 property values

2.4. H.264 Encoder

Figure 2.3 illustrates the schematic of the H.264 encoder. H.264 encoder works on macroblocks and motion-compensation like most other previous generation codecs. Video is formed by a series of picture frames. Each picture frame is an image which is split down into blocks. The block sizes can vary in H.264. The encoder may perform intra-coding or inter-coding for the macroblocks of a given frame. Intra coded frames are encoded and decoded independently. They do not need any reference frames. Hence they provide access points to the coded sequence where decoding can start. H.264 uses nine spatial prediction modes in intra-coding to reduce spatial redundancy in the source signal of the picture as shown in Figure 2.6. These prediction modes are explained in Figure 2.4. Inter-coding uses inter-prediction of a given block from some previously decoded pictures. The aim to use inter-coding is to reduce the temporal redundancy by making use of motion vectors. Motion vectors give the direction of motion of a particular block from the current frame to the next frame. The prediction residuals are obtained which then undergo transformation to remove spatial correlation in the block. The transformed coefficients, thus obtained, undergo quantization. The motion vectors (obtained from inter-prediction) or intra-prediction modes are combined with the quantized transform coefficient information. They are then encoded using entropy code such as context-based adaptive variable length coding (CAVLC) or context-based adaptive binary arithmetic coding (CABAC) [25].

There is a local decoder within the H.264 encoder. This local decoder performs the operations of inverse quantization and inverse transform to obtain the residual signal in the spatial domain. The prediction signal is added to the residual signal to reconstruct the input frame. This input frame is fed in the deblocking filter to remove blocking artifacts at the block boundaries. The output of the deblocking filter is then fed to inter/intra prediction blocks to generate prediction signals. The various coding tools used in the H.264 encoder are explained in the sections 2.4.1 through 2.4.6.

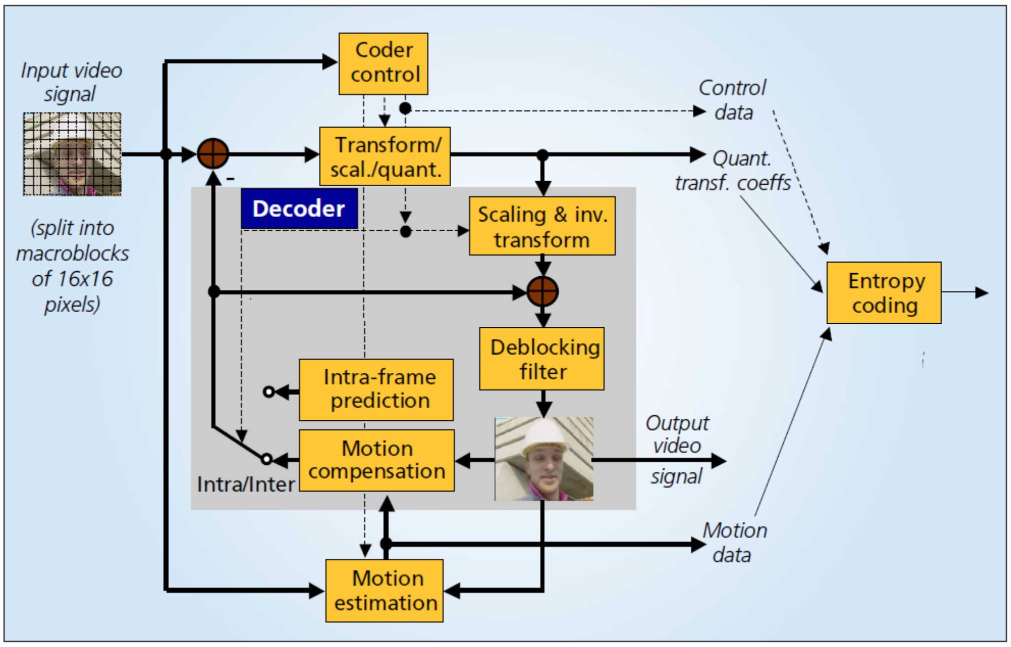


Figure 2.3. H.264 Video Coding Encoder Framework [33]

2.4.1. Intra-prediction

Intra-prediction uses the macroblocks from the same image for prediction. Two types of prediction schemes are used for the luminance component. These two schemes can be referred as INTRA_4x4 and INTRA_16x16 [38]. In INTRA_4x4, a macroblock of size 16x16 pixels are divided into 16 4x4 sub blocks. Intra prediction scheme is applied individually to these 4x4 sub blocks. There are nine different prediction modes supported as shown in Figure 2.4. In FRExts profiles alone, there is also 8x8 luma spatial prediction (similar to 4x4 spatial prediction) and with low-pass filtering of the prediction to improve prediction performance.

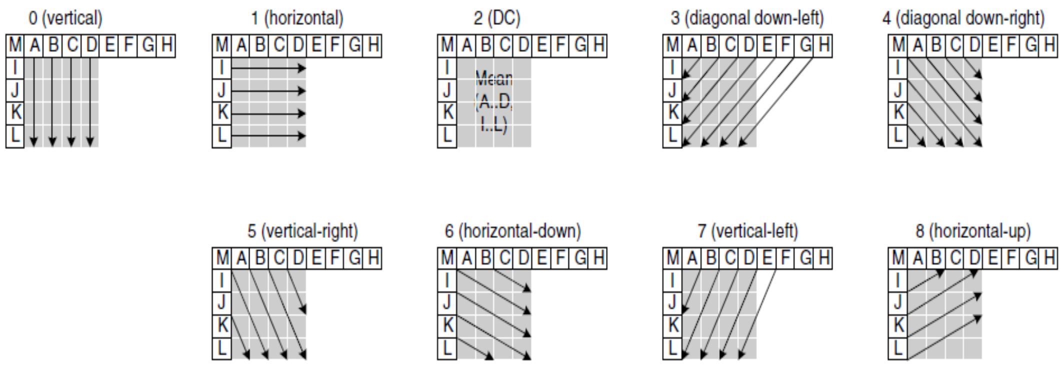


Figure 2.4. 4x4 Luma prediction (intra-prediction) modes in H.264 [33]

In Mode 0, the samples of the macroblock are predicted from the neighboring samples on the top. In Mode 1, the samples of the macroblock are predicted from the neighboring samples from the left. In Mode 2, the mean of all the neighboring samples is used for prediction. Mode 3 is in diagonally down-left direction. Mode 4 is in diagonal down-right direction. Mode 5 is in vertical-right direction. Mode 6 is in horizontal-down direction. Mode 7 is in vertical-left direction. Mode 8 is in horizontal up direction. The predicted samples are calculated from a weighted average of the previously predicted samples A to M.

For prediction of 16x16 intra prediction of luminance components, four modes are used as shown in Figure 2.5. The three modes of mode 0 (vertical), mode 1 (horizontal) and mode 2 (DC) are similar to the prediction modes for 4x4 block. In the fourth mode, the linear plane function is fitted in the neighboring samples.

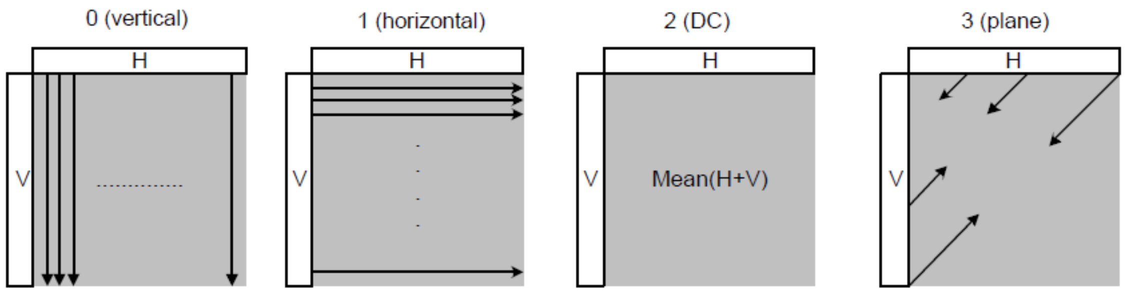


Figure 2.5. 16X16 Luma prediction (intra-prediction) modes in H.264 [33]

The chroma macroblock is predicted from neighboring chroma samples. The four prediction modes used for the chroma blocks are similar to 16x16 luma prediction modes. The number in which the prediction modes are ordered is different for chroma macroblock: mode 0 is DC, mode 1 is horizontal, mode 2 is vertical and mode 3 is plane. The block sizes for the chroma prediction depend on the sampling format. For 4:2:0 format, 8x8 size of chroma block is selected. For 4:2:2 format, 8x16 size of chroma block is selected. For 4:4:4 format, 16x16 size of chroma block is selected [33].

2.4.2. Inter-prediction

Inter-prediction is used to capitalize on the temporal redundancy in a video sequence. The temporal correlation is reduced by inter prediction through the use of motion estimation and compensation algorithms [33]. An image is divided into macroblocks; each 16x16 macroblock is further partitioned into 16x16, 16x8, 8x16, 8x8 sized blocks. A 8x8 sub-macroblock can be further partitioned into 8x4, 4x8, 4x4 sized blocks. Figure 2.6 illustrates the partitioning of a macroblock and a sub-macroblock [1]. The input video characteristics govern the block size. A smaller block size ensures less residual data; however smaller block sizes also mean more motion vectors and hence more number of bits required to encode these motion vectors.

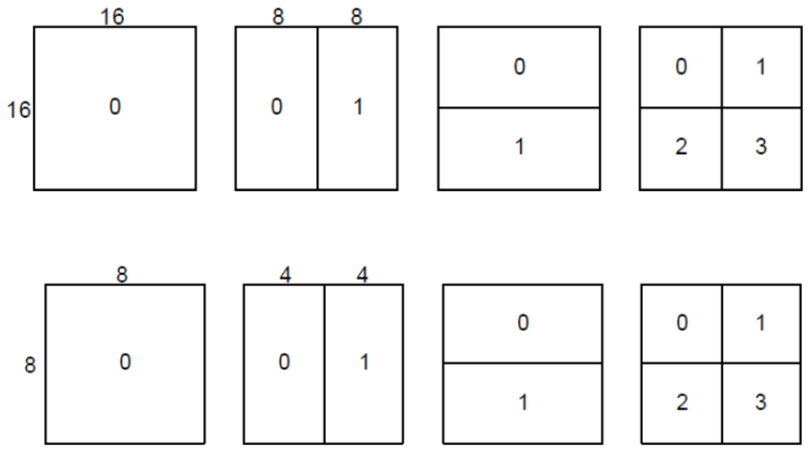


Figure 2.6. Macroblock portioning in H.264 for inter prediction [33] row1 (L-R) 16x16, 8x16, 16x8, 8x8 blocks and row2 (L-R) 8x8, 4x8, 8x4, 4x4 blocks

Each partition or sub-macroblock partition in an inter-coded macroblock is predicted from an area of the same size in a reference picture. The offset between the two areas (the motion vector) has quarter-sample resolution for the luma component and one-eighth-sample resolution for the chroma components. The luma and chroma samples at sub-sample positions do not exist in the reference picture and so it is necessary to create them using interpolation from nearby coded samples. Figures 2.7 and 2.8 illustrate half and quarter pixel interpolation used in luma pixel interpolation respectively. Six-tap filtering is used for derivation of half-pel luma sample predictions, for sharper sub pixel motion-compensation. Quarter-pixel motion is derived by linear interpolation of the half pel values, to save processing power.

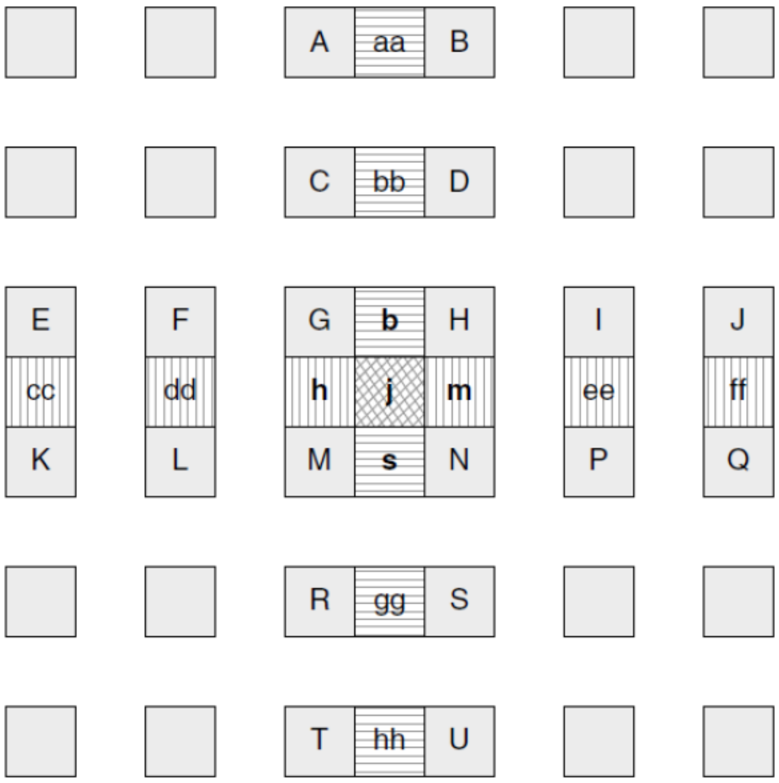


Figure 2.7. Interpolation of luma half-pel positions

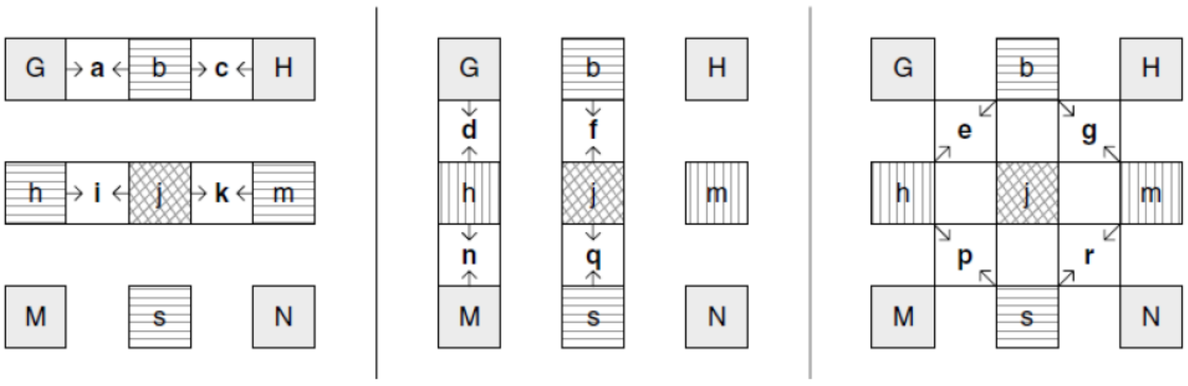


Figure 2.8. Interpolation of luma quarter-pel positions

The reference pictures used for inter prediction are previously decoded frames and are stored in the picture buffer. H.264 supports the use of multiple frames as reference frames. This is implemented by the use of an additional picture reference parameter which is transmitted along with the motion vector. Figure 2.9 illustrates an example with 4 reference pictures.

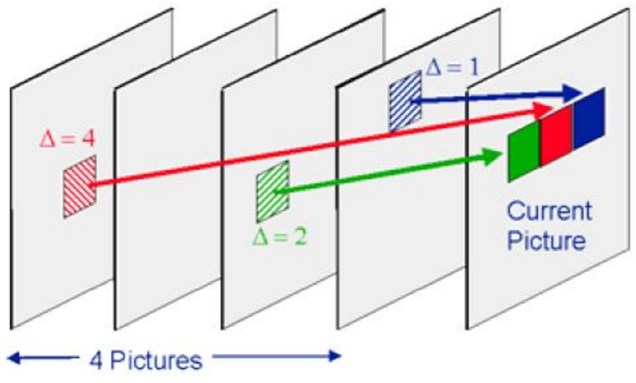


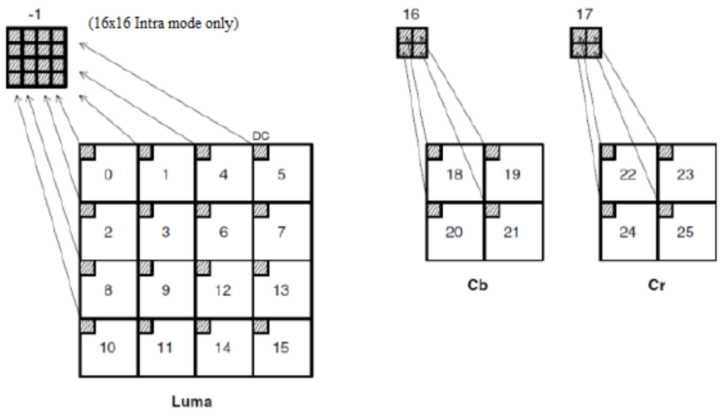
Figure 2.9. Motion compensated prediction with multiple reference frames [1]

2.4.3. Transform coding

There is high spatial redundancy among the prediction error signals. H.264 implements a block-based transform to reduce this spatial redundancy [33]. The former standards of MPEG-1 and MPEG-2 employed a two dimensional discrete cosine transform (DCT) for the purpose of transform coding of the size 8x8 [33]. H.264 uses integer transforms instead of the DCT. The size of these transforms is 4x4 [33]. The advantages of using a smaller block size in H.264 are stated as follows:

- The reduction in the transform size enables the encoder to better adapt the prediction error coding to the boundaries of the moving objects and to match the transform block size with the smallest block size of motion compensation.
- The smaller block size of the transform leads to a significant reduction in the ringing artifacts.
- The 4x4 integer transform has the benefit for removing the need for multiplications.H.264 employs a hierarchical transform structure, in which the DC coefficients of neighboring 4x4 transforms for luma and chroma signals are grouped into 4x4 blocks (blocks -1, 16 and 17) and transformed again by the Hadamard transform as shown in Figure 2.10 (a), (b), (c), (d) and (e).

As shown in Figure 2.10 (b) the first transform (matrix H1 is applied to all samples of all prediction error blocks of the luminance component (Y) and for all blocks of chrominance components (Cb and Cr). For blocks with mostly flat pixel values, there is significant correlation among transform DC coefficients of neighboring blocks. Hence, the standard specifies the 4x4 Hadamard transform (matrix H2 in figure 2.10 (c)) for luma DC coefficients (Figure 2.10 (c)) for 16x16 intra-mode only, and 2x2 Hadamard transform as shown in figure 2.10 (d) (matrix H3 in figure 2.10 (e)) for chroma DC coefficients.



(a)

$$\left(\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\begin{bmatrix}\\ & \mathbf{X} & \\ \\ \end{bmatrix}\begin{bmatrix}1&2&1&1\\1&1&-1&-2\\1&-1&-1&2\\1&-2&1&-1\end{bmatrix}\right)$$

(b)

$$\mathbf{Y}_D=\left(\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\begin{bmatrix}\\ & \mathbf{W}_D & \\ \\ \end{bmatrix}\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\right)/2$$

(c)

$$\mathbf{W}_{QD}=\begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}\mathbf{W}_D\end{bmatrix}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$$

(d)

$$H_1=\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\qquad H_2=\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\qquad H_3=\begin{bmatrix}1&1\\1&-1\end{bmatrix}$$

(e)

Figure 2.10. H.264 Transformation
(a) DC coefficients of 16 4x4 luma blocks, 4 4x4 Cb and Cr blocks [1]
(b) Matrix H1 (e) is applied to 4x4 block of luma/chroma coefficients X (a) [34]
(c) Matrix H2 (e) (4x4 Hadamard transform) applied to luma DC coefficients WD [34]
(d) Matrix H3 (e) (2x2 Hadamard transform) applied to chroma DC coefficients WD [34]
(e) Matrices H1, H2 and H3 of the three transforms used in H.264 [34]

2.4.4. Deblocking filter:

The deblocking filter is used to remove the blocking artifacts due to the block based encoding pattern. The transform applied after intra-prediction or inter-prediction is on blocks; the transform coefficients then undergo quantization. These block based operations are responsible for blocking artifacts which are removed by the in-loop deblocking filter as shown in Figure 2.11. It reduces the artifacts at the block boundaries and prevents the propagation of accumulated noise. The presence of the filter however adds to the complexity of the system [33]. Figure 2.11 illustrates a macroblock with sixteen 4x4 sub blocks along with their boundaries.

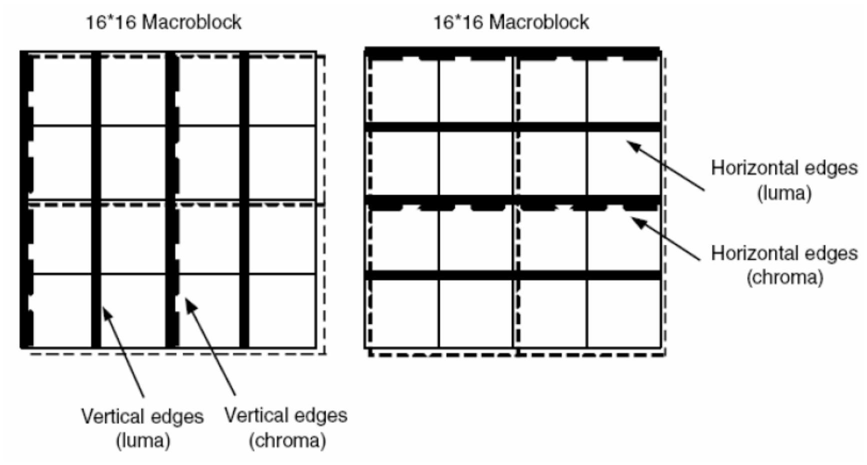


Figure 2.11. Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines) [1]

As shown in the Figure 2.11, the luma deblocking filter process is performed on the 16 sample edges – shown by solid lines. The chroma deblocking filter process is performed on 8 sample edges – shown in dotted lines.

H.264 employs deblocking process adaptively at the following three levels:

- At slice level – global filtering strength is adjusted to the individual characteristics of the video sequence.
- At block-edge level – deblocking filter decision is based on inter or intra prediction of the block, motion differences and presence of coded residuals in the two participating blocks.
- At sample level – it is important to distinguish between the blocking artifact and the true edges of the image. True edges should not be de blocked. Hence decision for deblocking at a sample level becomes important.

2.4.5. Entropy coding

H.264 uses variable length coding to match a symbol to a code based on the context characteristics. All the syntax elements except for the residual data are encoded by the Exp- Golomb codes [33]. The residual data is encoded using CAVLC. The main and the high profiles of H.264 use CABAC.

- Context-based adaptive variable length coding (CAVLC): After undergoing transform and quantization the probability that the level of coefficients is zero or +1 is very high [33]. CAVLC handles these values differently. It codes the number of zeroes and +1. For other values, their values are coded.
- Context-based adaptive binary arithmetic coding (CABAC): This technique utilizes the arithmetic encoding to achieve good compression. The schematic for CABAC is shown in Figure 2.12.

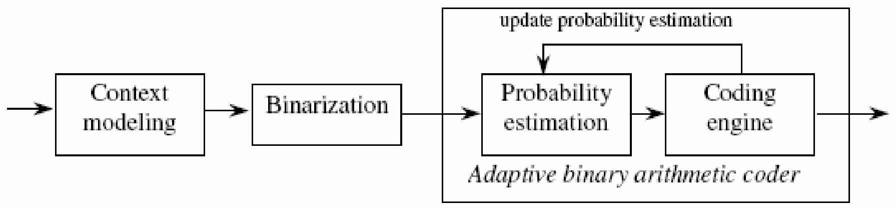


Figure 2.12. Schematic block diagram of CABAC [33]

CABAC consists of three steps:

- Step 1: Binarization: A non-binary value is uniquely mapped to a binary sequence
- Step 2: Context modeling: A context model is a probability model for one or more elements of binarized symbol. The probability model is selected such that corresponding choice may depend on previously encoded syntax elements.
- Step 3: Binary arithmetic coding: An arithmetic encoder encodes each element according to the selected probability model.

2.4.6. B-slices and adaptive weighted prediction

Bi-directional prediction which uses both past and future frames for reference can be very useful in improving the temporal prediction. Bi-directional prediction in H.264 uses multiple reference frames. Figure 2.13 shows bidirectional prediction from multiple reference frames. The video coding standards, before H.264, with B pictures use the bidirectional mode, with limitation that it allows the combination of a previous and subsequent prediction signals. In the previous standards, one prediction signal is derived from subsequent inter-picture, another from a previous picture, the other from a linear averaged signal of two motion compensated prediction signals.

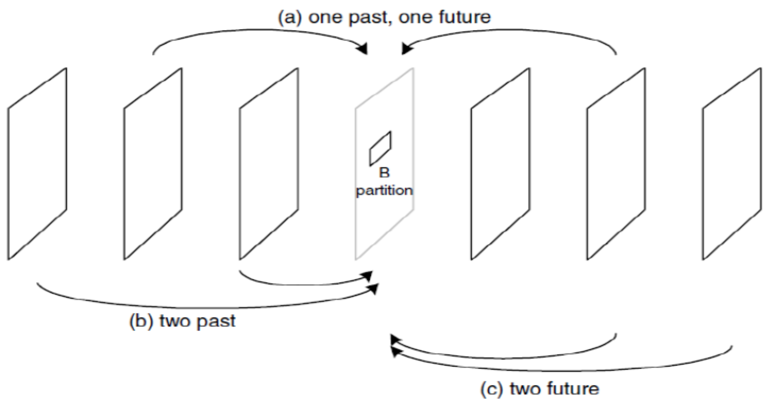


Figure 2.13. Partition prediction examples in a B macroblock type: (a) past/future, (b) past, (c) future [33]

H.264 supports forward/backward prediction pair and also supports forward/forward and backward/backward prediction pair [33]. Figure 2.13 (a) and Figure 2.13 (b) describe the scenario where bidirectional prediction and multiple reference frames respectively are applied and a macroblock is thereby predicted as a linear combination of multiple reference signals using weights as described in Equation 2.1. Two forward references for prediction are beneficial for motion compensated prediction of a region just before scene change. Two backward reference frames are beneficial for frames just after scene change. H.264 also allows bi-directionally predictive-coded slice which may also be used as reference for inter-coding of other pictures. Except H.264, all the existing standards consider equal weights for reference pictures. Equal weights of reference signals are averaged and the prediction signal is obtained. H.264 also uses weighted prediction [33]. It can be used for a macroblock of P slice or B slice. Different weights can be assigned to the two different reference signals and the prediction signal is calculated as follows:

$$p = w_1 * r_1 + w_2 * r_2 \qquad \text{Equation 2.1.}$$

In Equation (2.1), p is the prediction signal, r_1 and r_2 are the reference signals and w_1 and w_2 are the prediction weights.

2.5. H.264 Decoder

The H.264 [34] decoder works similar in operation to the local decoder of H.264 encoder. An encoded bit stream is the input to the decoder. Entropy decoding (CABAC or CAVLC) takes place on the bit stream to obtain the transform coefficients. These coefficients are then inverse scanned and inverse quantized. This gives residual block data in the transform domain. Inverse transform is performed to obtain the data in the pixel domain. The resulting output is 4x4 blocks of residual signal. Depending on inter predicted or intra-predicted, an appropriate prediction signal is added to the residual signal. For an inter-coded block, a prediction block is constructed depending on the motion vectors, reference frames and previously decoded pictures. This prediction block is added to the residual block to reconstruct the video frames. These reconstructed frames then undergo deblocking before they are stored for future use for prediction or being displayed. Figure 2.14 illustrates the decoder.

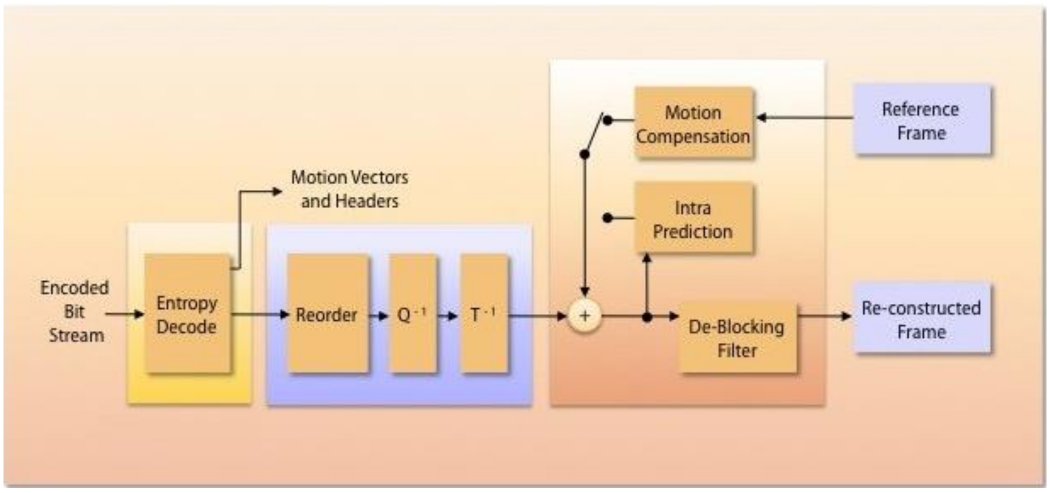


Figure 2.14. H.264 Decoder block diagram [33]

2.6. Summary

This chapter outlines the coding tools of H.264 codec. Having had a good understanding of H.264, next chapter describes the coding tools of High Efficiency Video Coding (HEVC).

CHAPTER 3: HEVC Codec Overview

3.1. Introduction:

High Efficiency Video Coding (HEVC) is the latest Video Coding format in use in the market which is second most popular codec after H.264 [11]. It challenges the state-of-the-art H.264/AVC Video Coding standard by being able to reduce the bit rate by 50%, retaining the same video quality. It came into existence in the early 2012 although Joint Collaborative Team on Video Coding (JCT-VC) was formed in January 2001 to carry out developments on HEVC, and ever since then a huge range of development has been going on. On 13 April 2013, HEVC standard also called H.265 was approved by ITU-T. Joint Collaborative Team on Video Coding (JCT-VC), is a group of video coding experts from ITU-T Study Group (VCEG) and ISO/IEC JTC 1/SC 29/WG 11 (MPEG).

HEVC is designed to address existing applications of H.264/MPEG-4 AVC and to focus on two key issues: increased video resolution and increased use of parallel processing architectures. It primarily targets consumer applications as pixel formats are limited to 4:2:0 8-bit and 4:2:0 10-bit. Main, Main 10 and Main intra profile (Main Still Picture profile) were finalized in 2013 [11]. The next revision of the standard, in July 2014 [35], had enabled new use-cases with the support of additional pixel formats such as 4:2:2 and 4:4:4 and bit depth higher than 10-bit [36], embedded bit-stream scalability and 3D video [37].

Multimedia consumer applications have a very large market [11] [37]. The revenues involved in digital TV broadcasting and DVD, Blu-ray distributions are substantial. Thus, standardization of video coding is essential. Standards simplify inter-operability between encoders and decoders from different manufacturers, they make it possible for different vendors to build platforms that incorporate video codecs, audio codecs, security and rights management and they all interact in well-defined and consistent ways. There are numerous video compression standards, both open source and proprietary, depending on the applications and end-usage.

An increasing diversity of services, the growing popularity of HD video, and the emergence of beyond-HD formats (e.g., 4k×2k or 8k×4k resolution) [38] are creating even stronger needs for coding efficiency superior to H.264/MPEG-4 AVC's capabilities. The need is even stronger when higher resolution is accompanied by stereo or multiview capture and display. Moreover, the traffic caused by video applications targeting mobile devices and tablet PCs, as well as the transmission needs for video-on-demand services, are imposing severe challenges on today's networks. An increased desire for higher quality and resolutions is also arising in mobile applications [11]. Storage space and bandwidth are limited. Even if high bandwidth technology (e.g. fiber-optic cable) was in place, the per-byte-cost of transmission would have to be very low before it would be feasible to use it for the staggering amounts of data required by HDTV and ultra HDTV.

3.2. HEVC Coding Design

3.2.1. Encoder and Decoder in HEVC:

Since H.261, all the video compression standards have been employing hybrid approach for the video coding layer [39]. Similarly, HEVC does the same. Figure 3.1 shows the encoder/decoder block diagram to create a compressed video bit stream. The compressed bit stream is stored or transmitted. A video decoder decompresses the bit stream to create a sequence of decoded frames [40]. The video encoder performs the following process to generate the bitstream yielding to HEVC standard. Figures 3.2. and 3.3. depict the block diagram of a hybrid video encoder and decoder, respectively. Each picture is partitioned into block shaped multiple units; the same exact partitioning is conveyed to the decoder. After partitioning each picture, it is intra or inter predicted. The first picture of a video sequence and the first picture at each clean random-access point is coded using intra prediction only. The rest of the pictures in a video sequence is coded using inter prediction. Linear spatial transformation is performed on the residual signal (the difference between the original picture unit and the prediction). The transform coefficients along with the prediction information are quantized and entropy encoded. In order to generate identical predictions for successive data, the encoder clones the decoder processing loop [11]. The loop performs inverse scaling, inverse transform to construct residual signal which is then added to the prediction. It is further processed to remove the artifacts induced by block-wide processing and quantization. The decoded picture buffer stores the final picture representation for prediction of successive data.

The video decoder decodes the encoded bitstream by performing the following steps:

1. Entropy decoding and extracting the elements of the coded sequence.
2. Rescaling and inverting the transform stage.
3. Predicting each unit and adding the prediction.

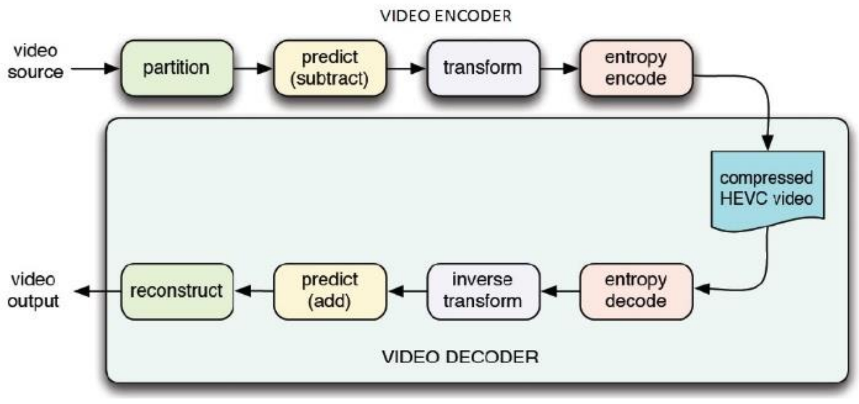


Figure 3.1. Encoder and Decoder coding techniques in HEVC [40]

3.2.2. Coding tree units and coding units:

HEVC supports highly flexible partitioning of a video sequence [11]. Each frame of the sequence is split into rectangular or square regions (units or blocks). The conventional macroblock is replaced by coding tree unit (CTU) in HEVC, which has a size selected by encoder and can be larger than a macroblock [11]. Each CTU consists of a luma CTB and the two chroma CTBs and syntax elements. In a CTU, a luma CTB has a picture area of $L \times L$ samples of the luma component and the corresponding chroma CTBs cover each $(L/2) \times (L/2)$ samples of each of the two chroma components. CTBs have always square shapes. The value of L may be equal to 16, 32, or 64 as determined by an encoded syntax element specified in the sequence parameter set (SPS). The value of L can be 16, 32 or 64. The larger CTBs are useful when encoding high-resolution video content and also enable better compression [41] [42].

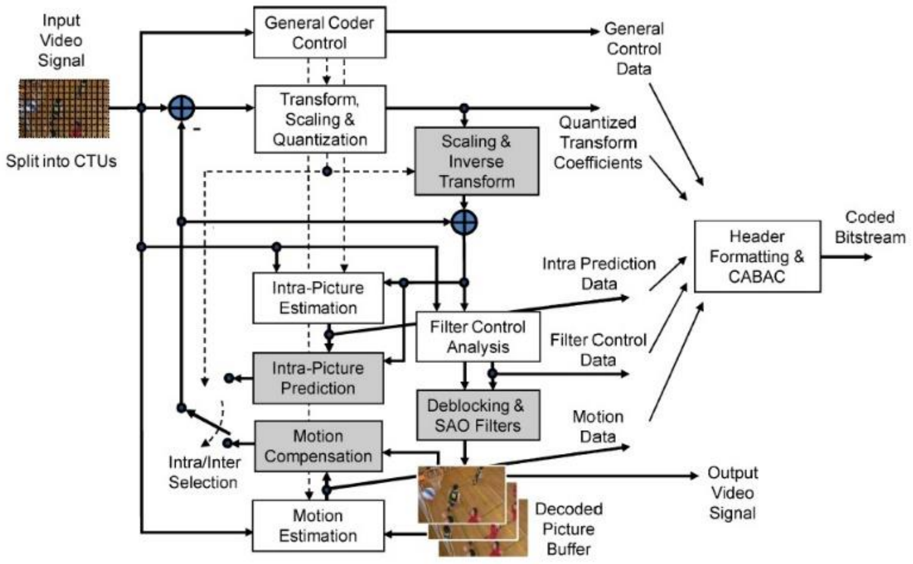


Figure 3.2. HEVC Videc Encoder Block Diagram [11]

Each CTB is split into one or multiple coding units (CUs). Each CU consists of one luma coding block (CB), two chroma CBs and associated syntax. Figure 3.4. shows the structure of CTU and CU in a video frame. The minimum luma CB size is computed from *L* and the maximum depth of a quad-tree and is always 8×8 or larger (in units of luma samples). Figure 3.5. and Figure 3.6 shows the quad-tree structure. The coding mode, intra or inter prediction, is selected at the CU level. CBs have always square shapes. CU has an associated partitioning into prediction units (PUs) and transform units (TUs). A CU is the root for both prediction unit (PU) and transform unit (TU) as shown in Figure 3.7. Figure 3.8. shows examples of various CTU sizes and CU sizes suitable for different resolutions and types of content. For example, for an application using 1080p content that is known to include only simple global motion activities, a CTU size of 64 (*L*=64) and maximum depth of 2 may be appropriate choice. For more general 1080p content, which may also include complex motion activities of small regions, a CTU size of 64 and maximum depth of 4 would be preferable [43]. Pictorial representations of various block divisions for HEVC in a frame is shown in Figure 3.9.

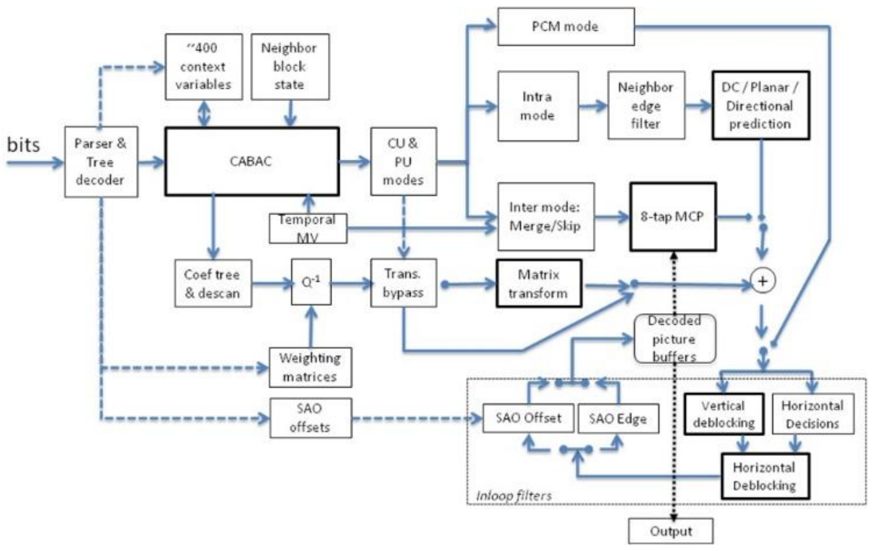


Figure 3.3. HEVC Video Decoder Block Diagram

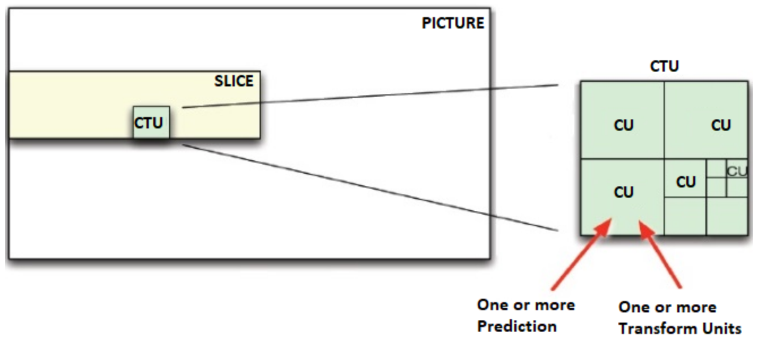


Figure 3.4. Coding tree unit (CTU) and Coding unit (CU)

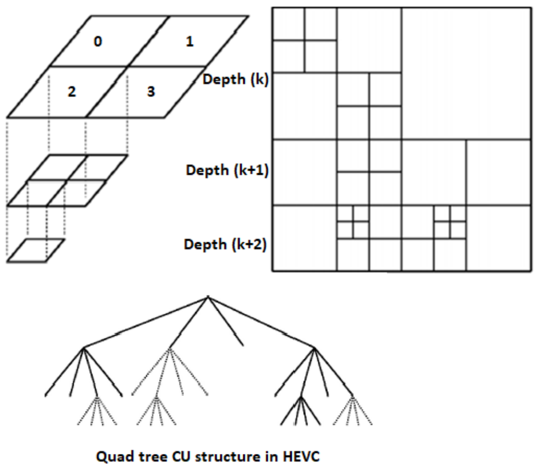


Figure 3.5. Quad Tree CU structure in HEVC

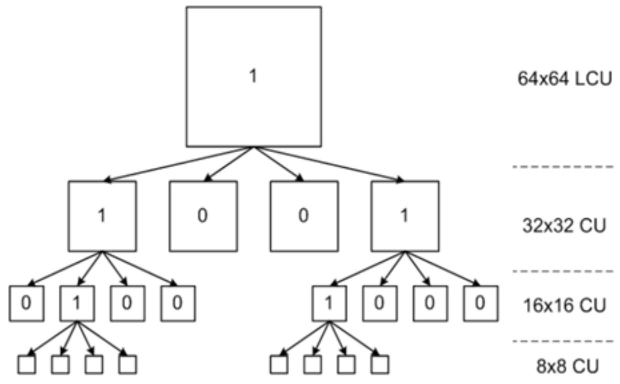


Figure 3.6. Quad Tree Splitting flags are 1's and 0's

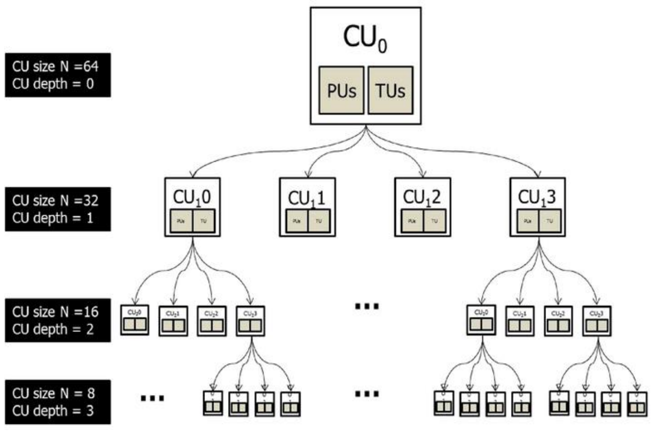


Figure 3.7. Coding structure in HEVC

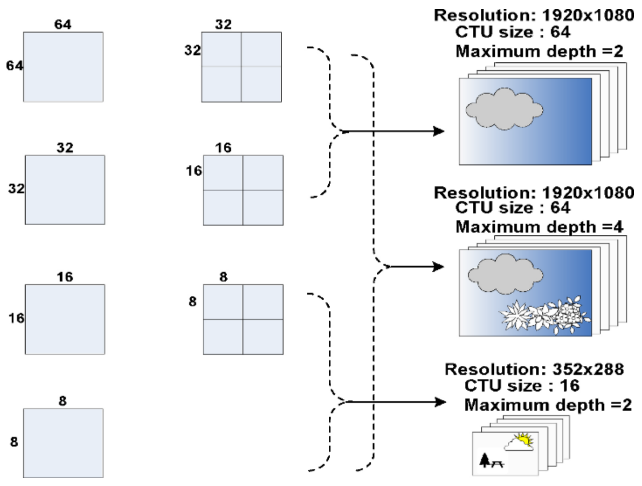


Figure 3.8. Flexible CU partitioning

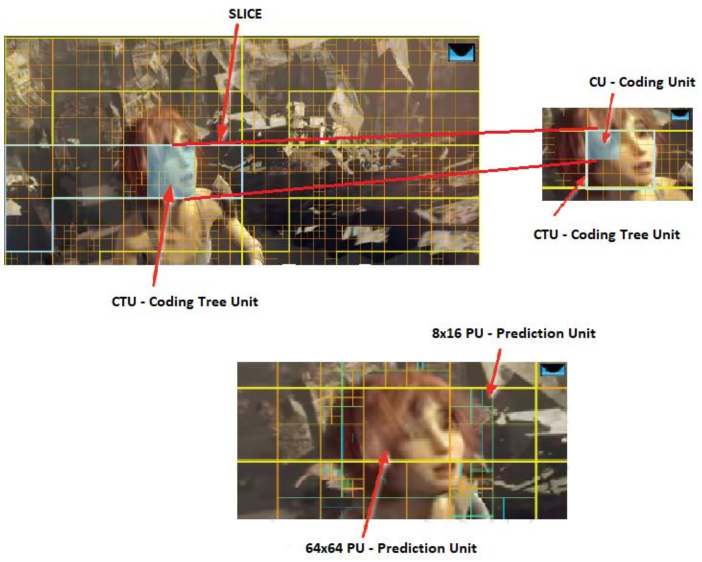


Figure 3.9. Pictorial representations of various block divisions for HEVC in a frame

3.2.3. Prediction units:

The luma and chroma PBs, together with the associated prediction syntax, form the PU. The luma and chroma CBs are split into luma and chroma prediction blocks (PBs) based on prediction-type decision [11]. The prediction mode for the CU is signaled as being intra or inter, according to whether it uses intrapicture (spatial) prediction or inter-picture (temporal) prediction. The size of PB can vary from 64×64 to 4×4 samples. For the prediction mode intra, except for the smallest CB size all PB size is same as the CB size that is allowed in the bitstream. Intra prediction can be performed on only square partitions. When CB has to be split into four PB which have their own intra-picture prediction mode, a flag is enabled. The reason for allowing this split is to enable distinct intrapicture prediction mode selections for blocks as small as 4×4 in size. When the luma intrapicture prediction operates with 4×4 blocks, the chroma intrapicture prediction also uses 4×4 blocks (each covering the same picture region as four 4×4 luma blocks). The actual region size at which the intrapicture prediction operates (which is distinct from the PB size, at which the intrapicture prediction mode is established) depends on the residual coding partitioning.

When the prediction mode is signaled as inter, it is specified whether the luma and chroma CBs are split into one, two, or four PBs. The splitting into four PBs is allowed only when the CB size is equal to the minimum allowed CB size, using an equivalent type of splitting as could otherwise be performed at the CB level of the design rather than at the PB level. When a CB is split into four PBs, each PB covers a quadrant of the CB. When a CB is split into two PBs, six types of this splitting are possible. The partitioning possibilities for interpicture-predicted CBs are depicted in Figure 3.10. The upper partitions illustrate the cases of not splitting the CB of size M×M, of splitting the CB into two PBs of size M × (M/2) or (M/2) × M, or splitting it into four PBs of size (M/2) × (M/2). The lower four partition types in Figure 3.10. are referred to as asymmetric motion partitioning (AMP) and are only allowed when M is 16 or larger for luma. For intrapicture-predicted CBs, only M×M and (M/2) × (M/2) are supported.

To minimize worst-case memory bandwidth, PBs of luma size 4×4 is not allowed for interpicture prediction, and PBs of luma sizes 4×8 and 8×4 are restricted to unipredictive coding.

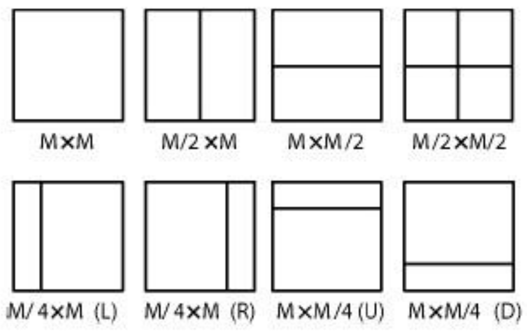


Figure 3.10. Modes for splitting a CB into PB’s. L=LEFT, R=RIGHT, U=UP, D=DOWN

3.2.4. Transform units:

The prediction residual is coded using block transforms. A TU tree structure has its root at the CU level [11]. The luma CB residual may be identical to the luma transform block (TB) or may be further split into smaller luma TBs. The same applies to the chroma TBs. Integer basis functions similar to those of a discrete cosine transform (DCT) are defined for the square TB sizes 4×4, 8×8, 16×16, and 32×32. For the 4×4 transform of luma intrapicture prediction residuals, an integer transform derived from a form of discrete sine transform (DST) is alternatively specified.

Only square CB and TB partitioning is specified, where a block can be recursively split into quadrants, as illustrated in Figure 3.11. For a given luma CB of size M×M, a flag signals whether it is split into four blocks of size M/2×M/2. If further splitting is possible, as signaled by a maximum depth of the residual quadtree indicated in the SPS, each quadrant is assigned a flag that indicates whether it is split into four quadrants. The leaf node blocks resulting from the residual quadtree are the transform blocks that are further processed by transform coding. The encoder indicates the maximum and minimum luma TB sizes that it will use. Splitting is implicit when the CB size is larger than the maximum TB size. Not splitting is implicit when splitting would result in a luma TB size smaller than the indicated minimum.

The chroma TB size is half the luma TB size in each dimension, except when the luma TB size is 4×4, in which case a single 4×4 chroma TB is used for the region covered by four 4×4 luma TBs. In the case of intrapicture-predicted CUs, the decoded samples of the nearest-neighboring TBs (within or outside the CB) are used as reference data for intrapicture prediction. In contrast to previous standards, the HEVC design allows a TB to span across multiple PBs for interpicture-predicted CUs to maximize the potential coding efficiency benefits of the quadtree-structured TB partitioning.

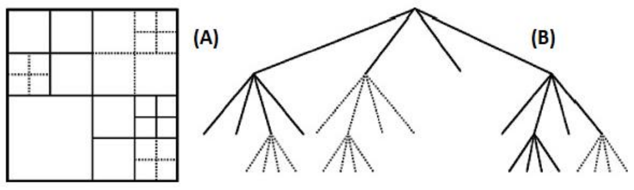


Figure 3.11. Subdivision of a CTB into CB's. Solid line indicate CB boundaries and dotted lines indicate TB boundaries. (A) CTB with its partitioning (B) Corresponding Quad Tree [11]

3.2.5. Slices and Tiles:

Slices are processed in the order of a raster scan [11]. A picture may be split into one or several slices as shown in Figure 3.12. (A) so that a picture is a collection of one or more slices. Slices are self-contained in the sense that, given the availability of the active sequence and picture parameter sets, their syntax elements can be parsed from the bitstream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without the use of any data from other slices in the same picture. Tiles are self-contained and independently decodable rectangular regions of the picture. The main purpose of tiles is to enable the use of parallel processing architectures for encoding and decoding. Multiple tiles may share header information by being contained in the same slice. Alternatively, a single tile may contain multiple group of CTUs as shown in Figure 3.12. (B).

3.2.6. Intrapicture Prediction:

Prediction operates according to the TB size, and previously decoded boundary samples from spatially neighboring TBs are used to form the prediction signal [11]. The possible prediction directions are shown in Figure 3.13. Directional prediction with 33 different directional orientations is defined for (square) TB sizes from 4×4 up to 32×32. Alternatively, planar prediction (assuming an amplitude surface with a horizontal and vertical slope derived from the boundaries) and DC prediction (a flat surface with a value matching the mean value of the boundary samples) can also be used. For chroma, the horizontal, vertical, planar, and DC prediction modes can be explicitly signaled, or the chroma prediction mode can be indicated to be the same as the luma prediction mode (and, as a special case to avoid redundant signaling, when one of the first four choices is indicated and is the same as the luma prediction mode, the Intra_Angular mode is applied instead) [39]. The number of supported prediction modes varies based on the PU size (see Table 3.1) [44]. From an encoding perspective, the increased number of prediction modes will require good mode selection heuristics to maintain a reasonable search complexity (see Table 3.2.) [45].

3.2.7. Interpicture Prediction:

Compared to intrapicture-predicted CBs, HEVC supports more PB partition shapes for interpicture-predicted CBs [11]. The partitioning modes of PART_2N×2N, PART_2N×N, and PART_N×2N indicate the cases when the CB is not split, split into two equal-size PBs horizontally, and split into two equal-size PBs vertically, respectively. Figure 3.14. shows Intra and Inter frame prediction modes for HEVC.

3.2.8. Motion vector signalling:

Advanced motion vector prediction (AMVP) is used, including derivation of several most probable candidates based on data from adjacent PBs and the reference picture [11]. A merge mode for MV coding can also be used, allowing the inheritance of MVs from temporally or spatially neighbouring PBs. Moreover, compared to H.264/MPEG-4 AVC, improved skipped and direct motion inferences are also specified.

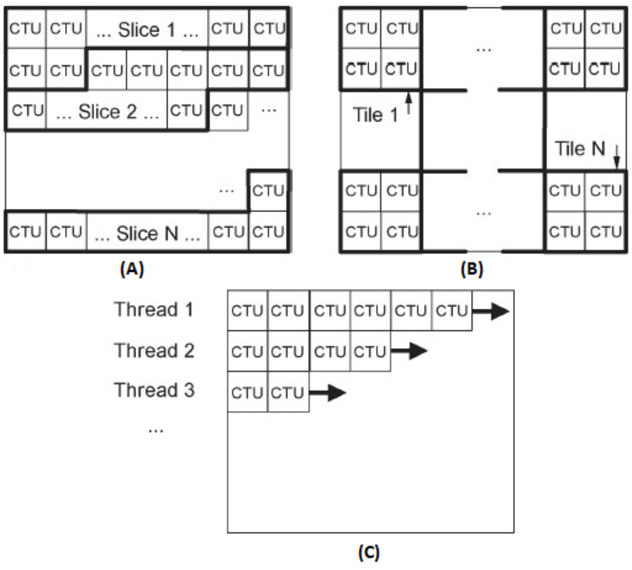


Figure 3.12. Subdivision of a picture (A) Slices, (B) Tiles and (C) Illustration of wave front parallel processing [11]

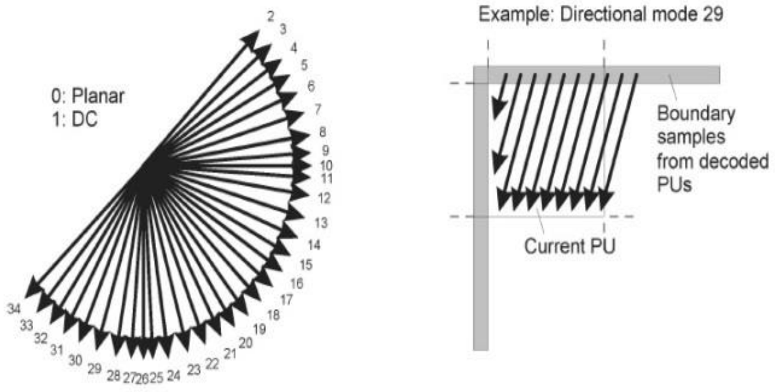

0: Planar
1: DC
Example: Directional mode 29
Boundary samples from decoded PUs
Current PU

Figure 3.13. Modes and directional orientations for intrapicture prediction [11]

PU Size	Intraprediction Modes	Number of Intra Prediction Modes
64×64	0–2, 34	4
32×32	0–34	35
16×16	0–34	35
8×8	0–34	35
4×4	0–16, 34	18

Table 3.1. Luma intra prediction modes supported by different PU sizes [46]

Size of PB	Number of PBs in a 64×64 CU	Number of modes to be tested in each PB	Total number of modes to be tested at this level
32×32	4	35	140
16×16	16	35	560
8×8	64	35	2240
4×4	256	18	4608
		Total	7548

Table 3.2. Total number of modes to be tested [47]

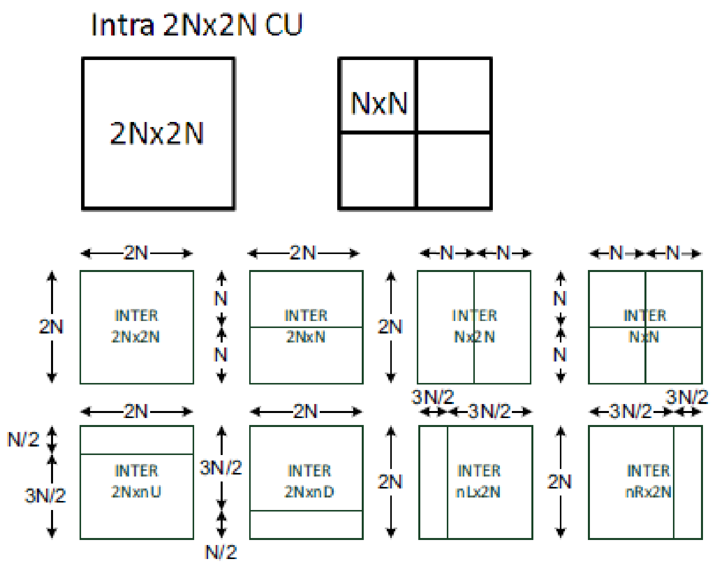


Figure 3.14. Intra and Inter frame prediction modes for HEVC [48]

3.2.9. Motion compensation:

Like MPEG-4/AVC, HEVC specifies motion vectors in 1/4-pel, but uses an 8-tap filter for luma (all positions), and a 4-tap 1/8-pel filter for chroma as shown in Figure 3.15. Because of the 8-tap filter, any given N×M sized block requires extra pixels on all sides (3 left/above, 4 right and below) to provide the filter with the data it needs. With small blocks like an 8×4, (8+7) × (4+7) = 15×11 pixels are needed. The HEVC standard limits the smallest block to be uni-directional and 4×4 is not supported since more small blocks require more memory read, thus increasing more memory access, more time and more power. The HEVC standard also supports weighted prediction for both uni- and bi-directional PUs. However, the weights are always explicitly transmitted in the slice header; there is no implicit weighted prediction like in MPEG-4/AVC. Quarter-sample precision is used for the motion vectors. 7-tap (weights: -1, 4, -10, 58, 17, -5, 1) or 8-tap (weights: -1, 4, -11, 40, 40, -11, 4, 1) filters are used for interpolation of fractional-sample positions. The 8-tap filter is applied for half sample positions and the 7-tap filter is applied for quarter sample positions. Similar to H.264/MPEG-4 AVC, multiple reference pictures are used as shown in Figure 2.10. For each PB, either one or two motion vectors can be transmitted, resulting either in unipredictive or bipredictive coding, respectively. A scaling and offset operation can be applied to the prediction signal/signals in a manner known as weighted prediction.

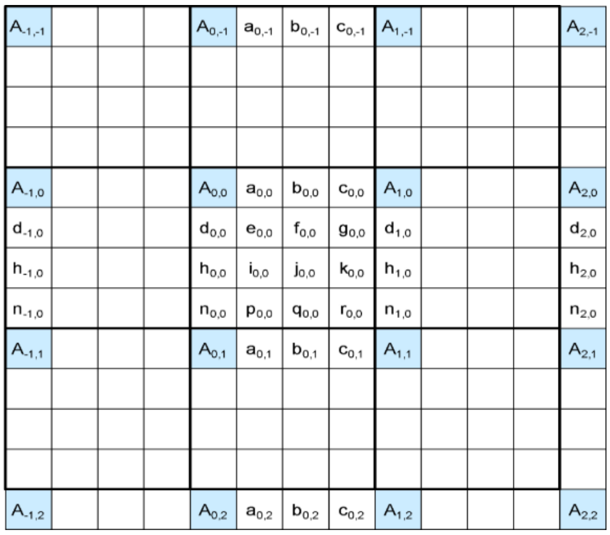


Figure 3.15. Integer and Fractional sample position for luma interpolation [11]

3.2.10. Transform, Scaling and Quantization

HEVC uses transform coding of the prediction error residual in a similar manner as in prior standards [11]. The supported transform block sizes are 4×4, 8×8, 16×16, and 32×32. Smaller size transform matrices are embedded in larger size transform matrices. This simplifies implementation, since a 32×32 matrix, can do 4×4, 8×8, 16×16, and 32×32 transform [49]. Transform matrices 4×4 through 32×32 INTDCTs (embedded) are shown. Transform matrices for each size transform are as follows.

nS = 4

{64, 64, 64, 64}

{83, 36,-36,-83}

{64,-64,-64, 64}

{36,-83, 83,-36}

nS = 8

{64, 64, 64, 64, 64, 64, 64, 64}

{89, 75, 50, 18,-18,-50,-75,-89}

{83, 36,-36,-83,-83,-36, 36, 83}

```
{75,-18,-89,-50, 50, 89, 18,-75}
{64,-64,-64, 64, 64,-64,-64, 64}
{50,-89, 18, 75,-75,-18, 89,-50}
{36,-83, 83,-36,-36, 83,-83, 36}
{18,-50, 75,-89, 89,-75, 50,-18}
```

nS = 16

```
{64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64}
{90 87 80 70 57 43 25  9 -9-25-43-57-70-80-87-90}
{89 75 50 18-18-50-75-89-89-75-50-18 18 50 75 89}
{87 57  9-43-80-90-70-25 25 70 90 80 43 -9-57-87}
{83 36-36-83-83-36 36 83 83 36-36-83-83-36 36 83}
{80  9-70-87-25 57 90 43-43-90-57 25 87 70 -9-80}
{75-18-89-50 50 89 18-75-75 18 89 50-50-89-18 75}
{70-43-87  9 90 25-80-57 57 80-25-90 -9 87 43-70}
{64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64}
{57-80-25 90 -9-87 43 70-70-43 87  9-90 25 80-57}
{50-89 18 75-75-18 89-50-50 89-18-75 75 18-89 50}
{43-90 57 25-87 70  9-80 80 -9-70 87-25-57 90-43}
{36-83 83-36-36 83-83 36 36-83 83-36-36 83-83 36}
{25-70 90-80 43  9-57 87-87 57 -9-43 80-90 70-25}
{18-50 75-89 89-75 50-18-18 50-75 89-89 75-50 18}
{ 9-25 43-57 70-80 87-90 90-87 80-70 57-43 25 -9}
```

nS = 32

```
{64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64}
{90 90 88 85 82 78 73 67 61 54 46 38 31 22 13  4 -4-13-22-31-38-46-54-61-67-73-78-82-85-88-90-90}
{90 87 80 70 57 43 25  9 -9-25-43-57-70-80-87-90-90-87-80-70-57-43-25 -9  9 25 43 57 70 80 87 90}
{90 82 67 46 22 -4-31-54-73-85-90-88-78-61-38-13 13 38 61 78 88 90 85 73 54 31  4-22-46-67-82-90}
{89 75 50 18-18-50-75-89-89-75-50-18 18 50 75 89 89 75 50 18-18-50-75-89-89-75-50-18 18 50 75 89}
{88 67 31-13-54-82-90-78-46 -4 38 73 90 85 61 22-22-61-85-90-73-38  4 46 78 90 82 54 13-31-67-88}
{87 57  9-43-80-90-70-25 25 70 90 80 43 -9-57-87-87-57 -9 43 80 90 70 25-25-70-90-80-43  9 57 87}
```

```
{85 46-13-67-90-73-22 38 82 88 54 -4-61-90-78-31 31 78 90 61  4-54-88-82-38 22 73 90 67 13-46-85}
{83 36-36-83-83-36 36 83 83 36-36-83-83-36 36 83 83 36-36-83-83-36 36 83 83 36-36-83-83-36 36 83}
{82 22-54-90-61 13 78 85 31-46-90-67  4 73 88 38-38-88-73 -4 67 90 46-31-85-78-13 61 90 54-22-82}
{80  9-70-87-25 57 90 43-43-90-57 25 87 70 -9-80-80 -9 70 87 25-57-90-43 43 90 57-25-87-70  9 80}
{78 -4-82-73 13 85 67-22-88-61 31 90 54-38-90-46 46 90 38-54-90-31 61 88 22-67-85-13 73 82  4-78}
{75-18-89-50 50 89 18-75-75 18 89 50-50-89-18 75 75-18-89-50 50 89 18-75-75 18 89 50-50-89-18 75}
{73-31-90-22 78 67-38-90-13 82 61-46-88 -4 85 54-54-85  4 88 46-61-82 13 90 38-67-78 22 90 31-73}
{70-43-87  9 90 25-80-57 57 80-25-90 -9 87 43-70-70 43 87 -9-90-25 80 57-57-80 25 90  9-87-43 70}
{67-54-78 38 85-22-90  4 90 13-88-31 82 46-73-61 61 73-46-82 31 88-13-90 -4 90 22-85-38 78 54-67}
{64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64 64-64-64 64}
{61-73-46 82 31-88-13 90 -4-90 22 85-38-78 54 67-67-54 78 38-85-22 90  4-90 13 88-31-82 46 73-61}
{57-80-25 90 -9-87 43 70-70-43 87  9-90 25 80-57-57 80 25-90  9 87-43-70 70 43-87 -9 90-25-80 57}
{54-85 -4 88-46-61 82 13-90 38 67-78-22 90-31-73 73 31-90 22 78-67-38 90-13-82 61 46-88  4 85-54}
{50-89 18 75-75-18 89-50-50 89-18-75 75 18-89 50 50-89 18 75-75-18 89-50-50 89-18-75 75 18-89 50}
{46-90 38 54-90 31 61-88 22 67-85 13 73-82  4 78-78 -4 82-73-13 85-67-22 88-61-31 90-54-38 90-46}
{43-90 57 25-87 70  9-80 80 -9-70 87-25-57 90-43-43 90-57-25 87-70 -9 80-80  9 70-87 25 57-90 43}
{38-88 73 -4-67 90-46-31 85-78 13 61-90 54 22-82 82-22-54 90-61-13 78-85 31 46-90 67  4-73 88-38}
{36-83 83-36-36 83-83 36 36-83 83-36-36 83-83 36 36-83 83-36-36 83-83 36 36-83 83-36-36 83-83 36}
{31-78 90-61  4 54-88 82-38-22 73-90 67-13-46 85-85 46 13-67 90-73 22 38-82 88-54 -4 61-90 78-31}
{25-70 90-80 43  9-57 87-87 57 -9-43 80-90 70-25-25 70-90 80-43 -9 57-87 87-57  9 43-80 90-70 25}
{22-61 85-90 73-38 -4 46-78 90-82 54-13-31 67-88 88-67 31 13-54 82-90 78-46  4 38-73 90-85 61-22}
{18-50 75-89 89-75 50-18-18 50-75 89-89 75-50 18 18-50 75-89 89-75 50-18-18 50-75 89-89 75-50 18}
{13-38 61-78 88-90 85-73 54-31  4 22-46 67-82 90-90 82-67 46-22 -4 31-54 73-85 90-88 78-61 38-13}
{ 9-25 43-57 70-80 87-90 90-87 80-70 57-43 25 -9 -9 25-43 57-70 80-87 90-90 87-80 70-57 43-25  9}
{ 4-13 22-31 38-46 54-61 67-73 78-82 85-88 90-90 90-90 88-85 82-78 73-67 61-54 46-38 31-22 13 -4}
```

Two-dimensional transforms are computed by applying 1-D transforms in the horizontal and vertical directions. The elements of the core transform matrices were derived by approximating scaled DCT basis functions. For the transform block size of 4×4, an alternative integer transform derived from a DST is applied to the luma residual blocks for intrapicture prediction modes. Since the rows of the transform matrix are close approximations of values of uniformly scaled basis functions of the orthonormal DCT, the prescaling operation that is incorporated in the dequantization of H.264/MPEG-4 AVC is not needed in

HEVC. For quantization, HEVC uses essentially the same uniform reconstruction quantizer (URQ) scheme controlled by a quantization parameter (QP) as in H.264/MPEG-4 AVC. The range of the QP values is defined from 0 to 51, and an increase by 6 doubles the quantization step size such that the mapping of QP values to step sizes is approximately logarithmic. Quantization scaling matrices are also supported.

3.2.11. Entropy coding

Context adaptive binary arithmetic coding (CABAC) is used for entropy coding [11]. This is similar to the CABAC scheme in H.264/MPEG-4 AVC [30] but has undergone several improvements to improve its throughput speed (especially for parallel-processing architectures) and its compression performance, and to reduce its context memory requirements.

3.2.12. In-loop deblocking filtering

A deblocking filter similar to the one used in H.264/MPEG-4 AVC is operated within the interpicture prediction loop. However, the design is simplified in regard to its decision-making and filtering processes and is made more friendly to parallel processing.

3.2.13. Sample adaptive offset (SAO)

Sample adaptive offset is applied to the reconstruction signal after the deblocking filter by using the offsets given in each CTB [50]. The encoder makes a decision on whether or not the SAO is applied for current slice. If SAO is enabled for current slice, the current slice allows each CTB select one of five SAO types as shown in Table 3.3. The concept of SAO is to classify pixels into categories and reduces the distortion by adding an offset to pixels of each category. SAO operation includes Edge Offset (EO) which uses edge properties for pixel classification as SAO type 1-4 and Band Offset (BO) which uses pixel intensity for pixel classification as SAO type 5. Each CTB will have its own SAO parameters include sao_merge_left_flag, sao_merge_up_flag, SAO type and four offsets. If sao_merge_left_flag is equal to 1 current CTB will reuse the SAO type and offsets of left CTB, otherwise current CTB will not reuse SAO type and offsets of left CTB. If sao_merge_up_flag is equal to 1, current CTB will reuse SAO type and offsets of upper CTB, otherwise current CTB will not reuse SAO type and offsets of upper CTB.

SAO type	Sample adaptive offset type to be used	Number of categories
0	None	0
1	1-D 0-degree pattern edge offset	4
2	1-D 90-degree pattern edge offset	4
3	1-D 135-degree pattern edge offset	4
4	1-D 45-degree pattern edge offset	4
5	Band offset	4

Table 3.3. Specification of SAO type [50]

3.2.14. Special “Transform Skip” Coding Modes

For certain types of content (especially screen content with graphics and text elements) more efficient compression is sometimes achieved when the transform is skipped (i.e. the residual is directly quantized and entropy coded) [37]. Furthermore, it is also possible to skip the quantization and loop filtering processes to enable lossless encoding of CUs.

3.3. Encoder and Decoder in HEVC

A number of design aspects new to the HEVC standard improve flexibility for operation over a variety of applications and network environments and improve robustness to data losses [11]. However, the high-level syntax architecture used in the H.264/MPEG-4 AVC standard [30] has generally been retained, including the following features.

1. Parameter set structure: Parameter sets contain information that can be shared for the decoding of several regions of the decoded video [11]. The parameter set structure provides a robust mechanism for conveying data that are essential to the decoding process. The concepts of sequence and picture parameter sets from H.264/MPEG-4 AVC are augmented by a new video parameter set (VPS) structure.

2. NAL unit syntax structure: Each syntax structure is placed into a logical data packet called a network abstraction layer (NAL) unit [11]. Using the content of a two-byte NAL unit header, it is possible to readily identify the purpose of the associated payload data.

3. Slices: A slice is a data structure that can be decoded independently from other slices of the same picture, in terms of entropy coding, signal prediction, and residual signal reconstruction [11]. A slice can either be an entire picture or a region of a picture. One of the main purposes of slices is resynchronization in the event of data losses. In the case of packetized transmission, the maximum number of payload bits within a slice is typically restricted, and the number of CTUs in the slice is often varied to minimize the packetization overhead while keeping the size of each packet within this bound.

4. Supplemental enhancement information (SEI) and video usability information (VUI) metadata: The syntax includes support for various types of metadata known as SEI and VUI [11] [51]. Such data provide information about the timing of the video pictures, the proper interpretation of the color space used in the video signal, 3-D stereoscopic frame packing information, other display hint information, and so on.

3.4. Parallel Decoding Syntax and Modified Slice Structuring

Finally, four new features are introduced in the HEVC standard to enhance the parallel processing capability or modify the structuring of slice data for packetization purposes. Each of them may have benefits in particular application contexts, and it is generally up to the implementer of an encoder or decoder to determine whether and how to take advantage of these features.

1. Tiles: The option to partition a picture into rectangular regions called tiles has been specified [11]. The main purpose of tiles is to increase the capability for parallel processing rather than provide error resilience. Tiles are independently decodable regions of a picture that are encoded with some shared header information. Tiles can additionally be used for the purpose of spatial random access to local regions of video pictures. A typical tile configuration of a picture consists of segmenting the picture into rectangular regions with approximately equal numbers of CTUs in each tile. Tiles provide parallelism at a more coarse level of granularity (picture/ subpicture), and no sophisticated synchronization of threads is necessary for their use.

2. Wavefront parallel processing: When wavefront parallel processing (WPP) is enabled, a slice is divided into rows of CTUs [11] [50]. The first row is processed in an ordinary way, the second row can begin to be processed after only two CTUs have been processed in the first row, and the third row can begin to be processed after only two CTUs have been processed in the second row, and so on. The context models of the entropy coder in each row are inferred from those in the preceding row with a two-CTU processing lag. WPP provides a form of processing parallelism at a rather fine level of granularity, i.e., within a slice. WPP may often provide better compression performance than tiles (and avoid some visual artifacts that may be induced by using tiles). Figure 3.12 (C) illustrates wavefront parallel processing.

3. Dependent slice segments: A structure called a dependent slice segment allows data associated with a particular wavefront entry point or tile to be carried in a separate NAL unit [11], and thus potentially makes that data available to a system for fragmented packetization with lower latency than if it were all coded together in one slice. A dependent slice segment for a wavefront entry point can only be decoded after at least part of the decoding process of another slice segment has been performed. Dependent slice segments are mainly useful in low-delay encoding, where other parallel tools might penalize compression performance.

3.5. Summary

This chapter outlines the coding tools of High Efficiency Video Coding (HEVC). Having had a good understanding of H.264 and HEVC, next chapter describes the coding tools of next generation video codecs, Versatile Video Coding (VVC) and Essential Video Coding (EVC).

4.1. Introduction of versatile video coding

VVC is a coding standard which was out on July 6th, 2020 by the Joint Video Experts Team (JVET) of ITU-T and ISO-IEC. VVC is shown to provide an additional bitrate saving of about 35% on top of HEVC for equivalent perceptual quality with support for lossless and subjective lossless compression. Versatile video coding uses the subset of the tools in Joint Exploration Model (JEM). VVC supports up to 8K resolution, 360^o videos and High Dynamic Range (HDR) video formats.

4.1.1. Coding Structure

The QTBT (Quad Tree – Binary Tree) structure comes to deal with those higher resolutions by enabling more flexible partition shapes. Instead of the hierarchical quadtree structure, employed by HEVC, that splits a Coding Tree Unit (CTU) into three units: namely, Coding Unit (CU), Prediction Unit (PU) and Transform Unit (TU), the QTBT structure as shown in Figure 4.3. uses only one processing unit called CU. This latter may have either a square or rectangular shape. The QTBT schemas starts with a quadtree partition that divides a CTU into square shapes denoted as leaf nodes. The quadtree leaf nodes can be further partitioned by symmetric horizontal splitting or symmetric vertical splitting, which represent the binary tree structure. The decision of intra prediction mode is made on the CU level since no further splitting of PU or TU is involved. To accommodate the increased number of directional intra modes, an intra mode coding method with 6 MPMs is used as shown in Figure 4.1.

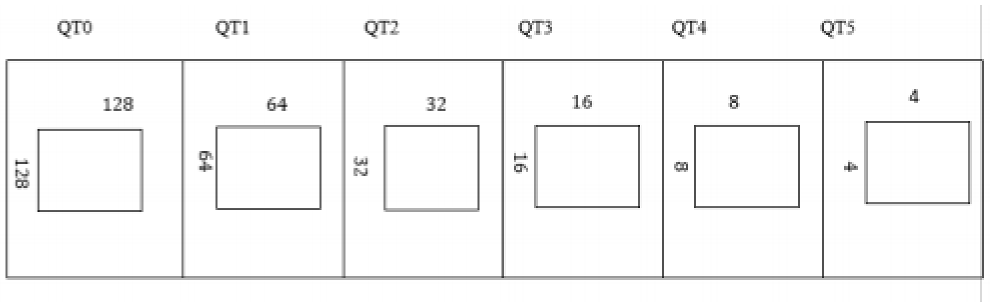


Figure 4.1. Quadtree partition [72]

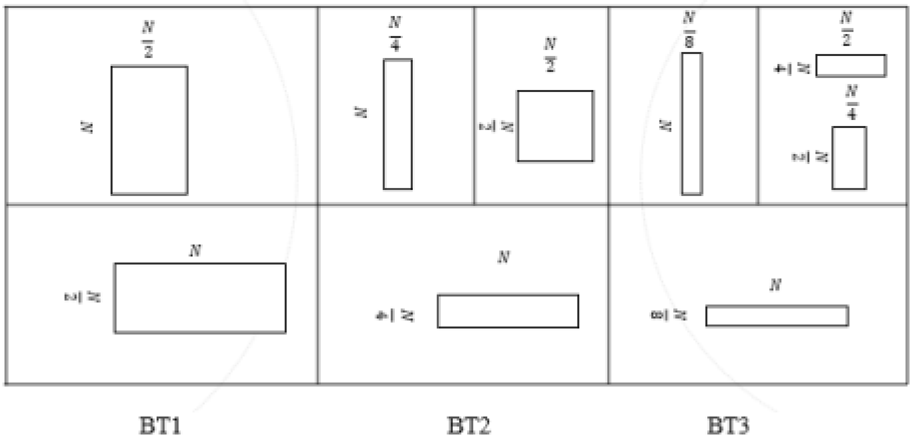


Figure 4.2. Binary Tree Partition [72]

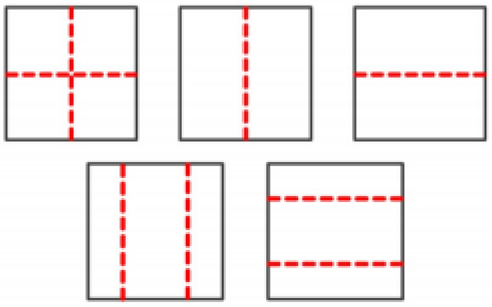


Figure 4.3. Block Partitioning of QTBT [72]

4.1.2. Intra Prediction

Each sub partition is processed in a similar way as any intra predicted block in VVC: first, a prediction and a residual signal is generated. Then, the residual is transformed, quantized and entropy coded and finally the non-zero coefficients are sent to the decoder. After each sub partition has been processed, its reconstructed samples can be used to calculate the prediction signal of the next sub partition, which will repeat the same steps until all sub partitions have been coded.

All sub partitions within a block using ISP utilize the same intra mode, which is hence signaled only once for the whole block. Besides, the intra mode will be selected from the most probable modes (MPM) list, which implies that the MPM flag is not sent to the decoder (it is inferred to be 1). Furthermore, the MPM list has been modified for the ISP case to exclude the DC mode and to prioritize horizontal intra modes (angular modes lower than the diagonal mode) if the split is horizontal and vertical intra modes (angular mode greater than or equal to the diagonal mode) if it is vertical. Figure 4.4. shows an intra prediction modes of VVC.

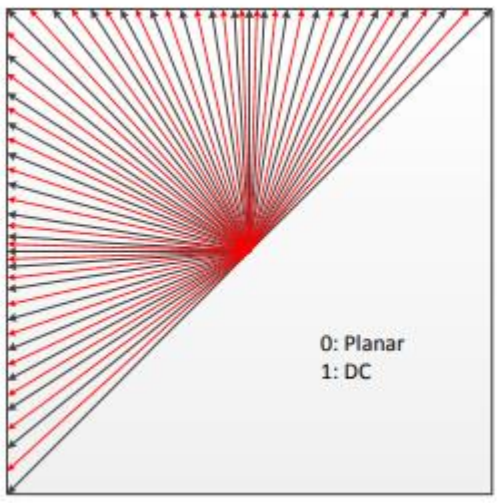


Figure 4.4. Intra Prediction modes of VVC [72]

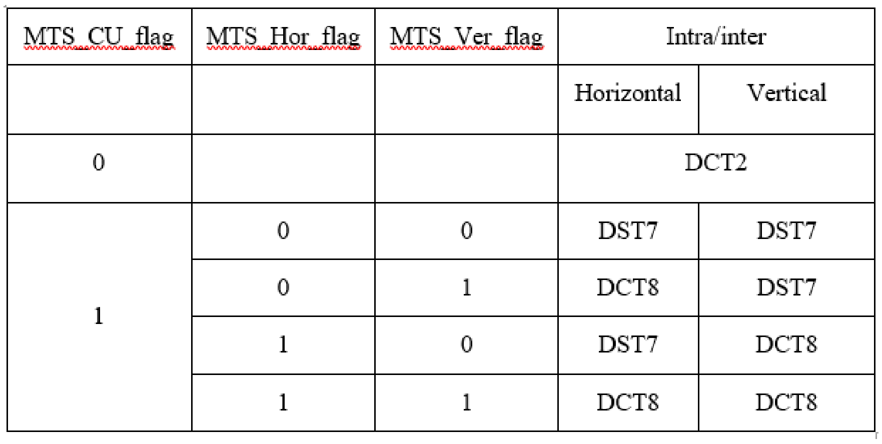
4.1.3. Inter Prediction

In conventional inter prediction mode, only translational motion model is applied. However, zoom-in/out and rotation cannot be well fit by the translational model utilized in the previous video coding standards. Therefore, the affine motion model, which is capable of representing non-translational motion. The current CU is divided into 4×4 subblocks, and the MV of each subblock is calculated. The luma MV precision is rounded to 1/16-sample precision.

When the AMVP mode is selected, an affine_flag is signalled to indicate whether affine prediction is used. If the affine prediction is applied, the syntax of inter_dir, ref_idx, mvp_index, and MVDs of the two CPs are signalled. In this case, the two CPMVs of the current CU are obtained by applying affine motion estimation. An affine MVP pair candidate list containing two affine MVP pairs is generated. The signalled mvp_index is used to indicate which one of two MVP pairs is selected for predicting the two CPMVs and generating the two MVDs. The MVP pair is generated by two affine candidates. Similar to the affine merge mode, one is the spatial inherited affine candidate, and the other is the corner derived affine candidate. If the neighboring CUs are coded with affine mode, the spatial inherited affine candidates can be generated. The affine motion model of the neighboring affine coded block is used to generate the MVP pair.

4.1.4. Transform and Quantization

In the current version of VVC, the Multiple Transform Selection (MTS) scheme uses five different transforms pairs (Horizontal Transform, Vertical Transform) comprising the Discrete Cosine Transform Type II (DCT-II), the Discrete Sine Transform Type VII (DST-VII) and the Discrete Cosine Transform Type VIII (DCT-VIII). Particularly, the pairs can be (DCT-II, DCT-II), (DST-VII, DST-VII), (DST-VII, DCTVIII), (DCT-VIII, DST-VII) and (DCT-VIII, DCT-VIII) as shown in Table 4.1. At the decoder side, the one that is used is indicated by a syntax element selected by the encoder. In the ISP case, however, the transform pair is implicitly decided according to the intra mode and the original block dimensions. These combinations have been extracted with a slight change consisting of using only the DCT-II if the transform has a length smaller than 4 or greater than 16. This restriction has been added in order to reduce the hardware implementation complexity of the algorithm. Logically, if a sub partition has a width or a height equal to 1, the transform is only applied on the dimension with a length greater than 1.



MTS_CU_flag	MTS_Hor_flag	MTS_Ver_flag	Intra/inter	
			Horizontal	Vertical
0			DCT2	
1	0	0	DST7	DST7
	0	1	DCT8	DST7
	1	0	DST7	DCT8
	1	1	DCT8	DCT8

Table 4.1. VVC Intra/Inter Transform code selection [72]

4.1.5. Coefficient Coding

The coefficients of the residual signals of the sub partitions are entropy-coded in the same way as regular blocks in VVC with the following modifications:

- The context of the Coded Block Flag (CBF) of each sub partition is the value of the CBF of the previously coded sub partition (in the case of the first sub partition a default value of 0 is assumed).
- At least one CBF of a block using ISP is assumed to be non-zero. Therefore, if a block contains n sub partitions and the first $n - 1$ have a zero CBF, then the n-th CBF is inferred to be zero and hence it is not explicitly signaled.
- The Last Position syntax element only requires one coordinate to be sent if the sub partition is a line.
- Let w and h be the width and height of each sub partition respectively. Then, if $w \geq 8$ and $h \leq 2$, the coefficient sub-blocks used for the significance map [9] will have a $16 h \times h$ shape. Analogously, if $w \leq 2$ and $h \geq 8$, coefficient sub-blocks will have a $w \times 16 w$ shape. In all other cases, they will have the same 4×4 shape used in regular intra-predicted blocks. As a result, all coefficient sub-blocks will have 16 samples.

4.1.6. Entropy Coding

In the entropy coder, the context-based adaptive binary arithmetic coding (CABAC) with multi-hypothesis probability estimation and context-dependent updating speed is applied. The CABAC core engine is the same as that in VTM 4.0. Each context variable has two probabilities, P1 and P2. The average of P1 and P2 is used as the final probability for the arithmetic coder engine. The P1 and P2 are updated with different speeds, where the faster updating speed is designed for faster convergence of probability and the slower updating speed is designed for higher robustness of probability. However, in VTM 4.0 CABAC, a 9-bits∗64-columns∗ 512-rows look-up-table (LUT) is used for deriving rangeOne and rangeZero.

4.1.7. In-Loop Filtering

1. Deblocking Filter **–** The deblocking filter is similar to that in HEVC with two major extensions added. First, in order to handle the blocky artifacts for large blocks, length-adaptive filtering with three new long tap filters for HEVC strong filtering cases is introduced. This is due to the observation that HEVC filter cannot effectively reduce blocky artifacts at large block boundaries in high resolution video (e.g., UHD). Second, the number of samples to be read or modified is limited to support parallel deblocking for 4×N or N×4 blocks.

2. Sample Adaptive offset – Sample adaptive offset (SAO) is adopted in HEVC. SAO classifies samples of one CTU into different groups and applies an offset to samples of the same group to reduce sample distortions. In HEVC, SAO supports four classes of classification in edge offset (EO) and 32 different sets of selected bands in band offset (BO). In this paper, two SAO modifications are added. One is to add three EO classes to improve the sample classification, as shown in Fig. 15, and the other is to remove the constraint related to the offset sign of EO. In HEVC, only smooth filtering is allowed for EO because of the offset sign constraint. In this paper, the offset signs of EO are explicitly coded to support both smoothing and sharpening. Encoder is allowed to choose between them.

4.2. Introduction to Essential video coding (EVC)

EVC is launched by MPEG and is due mid-2020. The goal of EVC is to provide the same compression efficiency as HEVC but with clear licensing conditions like royalty-free baseline profiling and a managed IPR in the main profile and consist of individually switchable enhancements. EVC also supports up to 8K resolution videos.

4.2.1. Coding Structure

A coding tree unit (CTU) is the basic unit of the proposed coding scheme. Maximal CTU size is 128x128. A CTU can be further partitioned recursively by binary and triple tree (BTT) structure. For instance, if width and height of a block are the same it can be represented as a 1:1 ratio CU or a square CU, and if the width is equal to 64 and the height is equal to 16 it can be represented as a 1:4 ratio CU. CU partitioning is conducted based on the allowed CU shapes and their allowed maximum and minimum sizes. In order to support 64x64 pipeline the ternary tree split and 1:4 or 4:1 ratio CU are disallowed when a CU size is greater than 64x64.

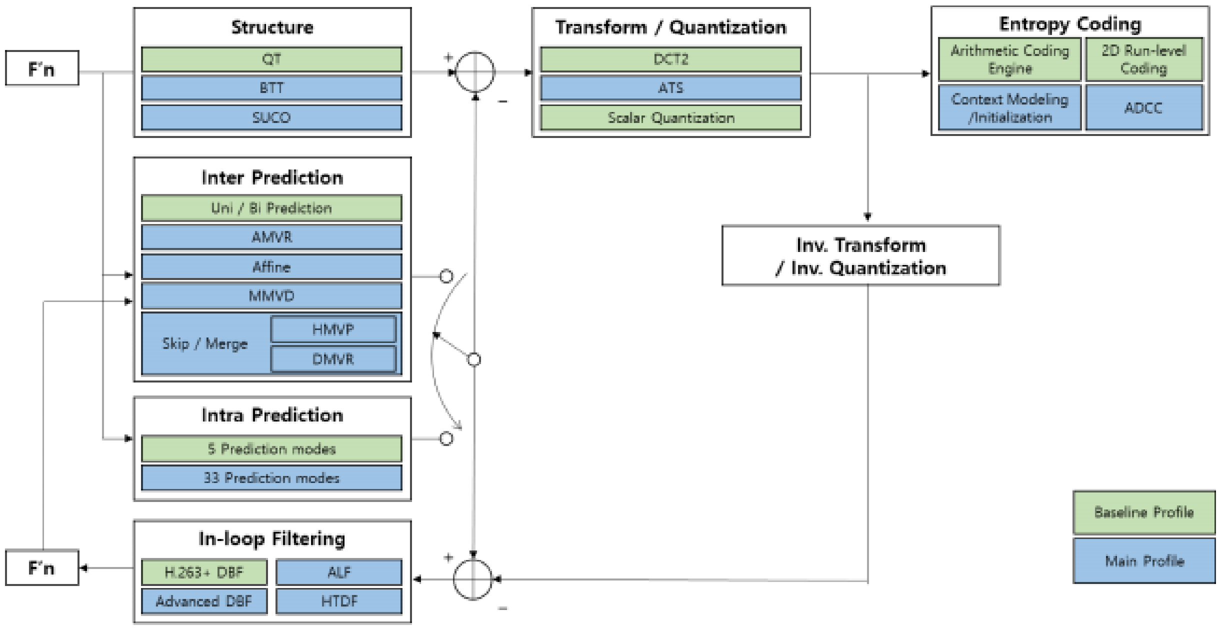


Figure 4.5. EVC Encoder Block Diagram [67]

The proposed structure has two CU split modes i.e. the binary split and the triple split mode, and each split mode has two directions, vertical and horizontal. Thus, a CU can be partitioned by four different split modes: BI_VER_SPLIT, BI_HOR_SPLIT, TRI_VER_SPLIT, and TRI_HOR_SPLIT. These split modes and the QTBT coding order are shown in the below figures.

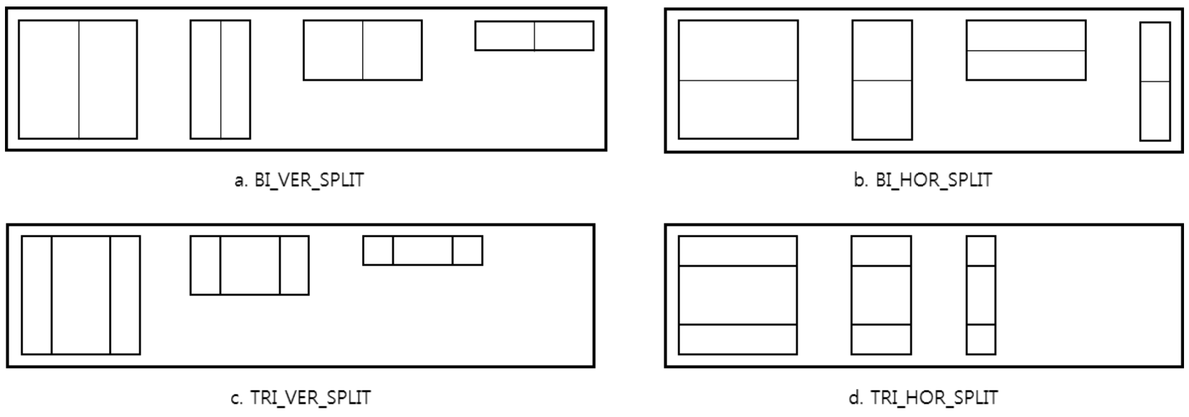


Figure 4.6. Bi/Tri Split modes

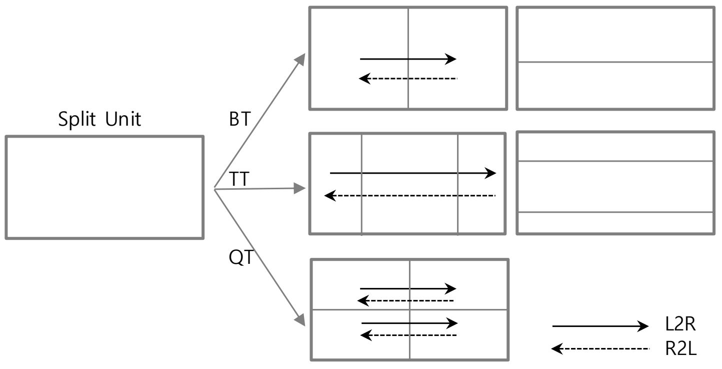


Figure 4.7. QT/TT/BT split coding order

4.2.2. Intra Prediction

For the Baseline configuration 5 directional prediction modes are employed: DC, horizontal (H), vertical (V), diagonal left (DL), diagonal right (DR). A codeword for prediction mode of the current block is adaptively assigned by using a mapping table between symbol and prediction mode which is selected based on the prediction modes of neighbouring upper and left blocks. To exploit spatial correlation efficiently based on flexible coding structure, a total of 33 intra prediction modes for luma component and 5 modes for chroma component are applied. DC, Bi-linear, Plane, DM modes, and 30 angular modes are applied, with straightforward extension for flexible block size.

Intra Block Copy (IBC) is another mode that is used in order to address requirements for Screen Content material. Technical aspects of the proposed IBC implementation are summarized follows

- IBC mode (IBC flag) is signaled in a CU level. The IBC mode is considered as a prediction mode other than intra and inter prediction modes. There is no need to include current picture as one of the reference pictures in the reference picture list 0. The motion vector of IBC is derived in integer pixel.

- The maximum block size for IBC coded block is signaled in SPS and up to 64 luma samples for either size.
- The coding of block vector (BV) is straightforward without using prediction. The coding engine reuses the one used in mvd coding. The BV considered is in integer resolution.

The whole reference block for IBC mode should be already reconstructed. In addition, some search range constraints are imposed such that the allowed search range for IBC mode is the reconstructed part of the current CTU plus some areas of the immediate left CTU (given the CTU size is 128x128). According to the location of the current block in the current CTU, the available reference areas change in a way that when a 64x64 region starts encoding/decoding, the whole region is considered as already coded. Therefore, the collocated 64x64 region in the left CTU can no longer be used for IBC reference.

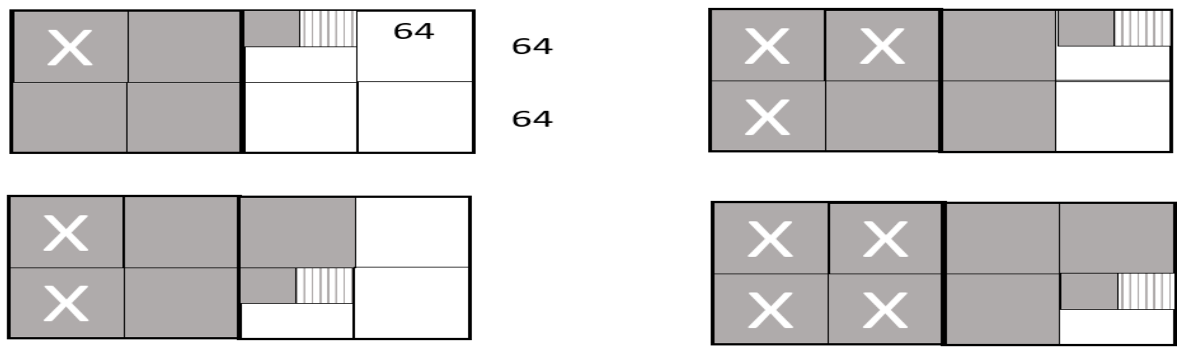


Figure 4.8. Intra-Block Copy (with a vertical split at 128x128) [67]

4.2.3. Inter Prediction

The inter prediction for Baseline configuration exploits three neighbouring motion vectors and a motion vector of temporally co-located blocks. After choosing one of the candidate motion vectors as a predictor, the index of the predictor is coded. Then the difference between the motion vector for the current block and the predictor may be coded based on encoder side decision. If the difference between the current block and the predictor is relatively small, the motion vector difference and a block residue are not coded, which is called the skip mode. Otherwise, the motion vector difference and the block residue are coded and signalled in the bitstream. The bi-directional prediction is a linear combination of two motion compensated blocks that involve two motion vectors, a forward and backward motion vector.

Motion model of the video content for each currently coded block is described conventionally through two parameters: reference index (refIdx) indicating the picture stored in decoded picture buffer (DPB) utilized as a reference for the current block, and the motion vector (MV) – amount of displacement in *x* and *y* directions between current block and the spatial position of the reference block in reference picture indicated by refIdx. Each current block can be predicted either from a single reference (uni-direction) or from two references in so called bi-directional prediction mode.

MV can be signaled either in merge or AMVP mode. Both signaling mechanism utilizes *motion vector prediction* (MVP) list (of different size) constructed from motion information available from spatial or temporal neighboring of the currently coded blocks. In the merge mode, an index that specify a certain entry in the MVP list is signaled and fully describe motion information for the current block.

Following are the multiple inter prediction modes that are used –

1. Merge/Skip Modes
2. Adaptive Motion Vector Resolution
3. Merge with Motion Vector Difference
4. Affine Prediction
5. Decoder-side Motion Vector Refinement
6. History-based Motion Vector Prediction

4.2.4. Transform and Quantization

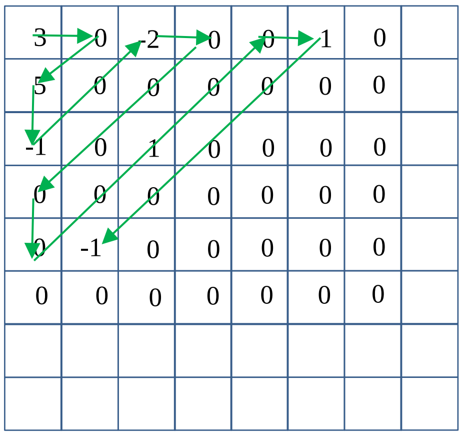
Transforms (i.e., DCT2) are applied to a residual block between an original block and the corresponding prediction block, as a conventional hybrid video codec does. Since transforms are applied to coding blocks, the transform size is equal to the coding block size, i.e. from 2x2 to 64x64. After the transform is conducted, scalar quantization is applied to the transformed coefficients. The range of the quantization parameter (QP) is from 0 to 51 and a scaling factor (SF) corresponding to each QP is defined by a look-up table.

In order to support 64x64 pipeline, the maximum allowed transform size is set to 64. If length of a side of CU is greater than the maximum transform size, the side is automatically split into two partitions. For intra coded block, a flag is used to signal to the decoder whether ATS applied or not. If encoder selects using ATS in a CU as core transform, two more flags are signaled to decoder to indicate which type is used, respectively for the horizontal and vertical directions. The value 0 indicate DST-VII is used and value 1 indicate DCT-VIII is used**.**

For an inter-predicted CU that have residuals, it is signaled to indicate whether the whole residual block or a sub-part of the residual block should be decoded. When only a sub-part of the residual block coded, the part of the residual block is coded with inferred transform type and the other part of the residual block is zeroed out. The sub-part position information and corresponding transform type are illustrated. The sub-part that contains residual information can be half or one quarter size of the current CU. ATS is allowed for CU with width and height both no larger than 64. The transform type is derived based on the position of the sub-block, instead of signaling the transform type as done for intra coded CU. For example, the horizontal and vertical transforms position 0 sub-part is DCT-8 and DST-7, respectively. When at least one side of the residual TU is greater than 32, transform is set as DCT-2.

4.2.5. Coefficient Coding

Transform coefficients of coded block after quantization are scanned in a predefined scan pattern and entropy coded. In the baseline configuration of the ETM, a run-length based coefficient coding method is used. Visualization of propose method is given. The table below gives an example of coded symbols for a chunk of transform coefficients.



3	0	-2	0	0	1	0	
5	0	0	0	0	0	0	
-1	0	1	0	0	0	0	
0	0	0	0	0	0	0	
0	-1	0	0	0	0	0	
0	0	0	0	0	0	0	

Table 4.2. Coefficient Scan Method [67]

Position	0	1	2	3	4	5	6	7	8	9
Coeffs stream	3	0	5	-1	0	-2	0	0	0	0
Output stream	0,2,0,0		1,4,0,0	0,0,1,0		1,1,1,0				
Position	10	11	12	13	14	15	16	17	18	19
Coeffs stream	0	0	1	0	0	1	0	0	0	-1
Output stream			6,0,0,0			2,0,0,0				3,0,1,1

Table 4.3. Coded Symbols for the coefficients [67]

To further employ statistical properties of transform coefficients a bit-plane based coefficient coding method, so called Advanced Coefficient Coding (ADCC), is utilized in the Main Profile of EVC instead of the run-length coding method currently used.

ADCC method utilizes the following design elements:

1. Fixed zig-zag scan pattern.
2. Signalling coordinates of the last-nonzero transform coefficient in the scan order.
3. Parsing transform coefficients in chunks of 16.
4. Signalling coefficients within each processing chunk as a sequence of significance and levels flags, sign flag and remaining levels.

4.2.6. Loop Filter

Most block-based video coding schemes introduce noticeable blocking artifacts. A loop filter based on H.263 was employed to increase objective and subjective image quality for Baseline configuration. Following the deblocking Adaptive Loop filter (ALF) with block-based filter adaption is applied. For the luma component, one among 25 filters is selected through the classification process for each 4×4 block, based on local statistics estimates, such as gradient and directionality. To benefit from symmetrical properties of filters, utilized ALF employs filter coefficient transformation process.

The following loop filters are used in Essential Video Coding –

1. Deblocking Filter
2. Adaptive Loop Filter
3. Hadamard Transform Domain Filter

4.2.7. Entropy Coding

The binary arithmetic coding scheme in JPEG is applied as the entropy coding engine of the proposed codec. After a binarization process of the given symbol, an arithmetic coding engine encodes each binary value with the corresponding context that stores the occurrence probability of a given value. After each binary value is encoded, the probability is updated by using a look-up table and the binary value of symbol is stored in the corresponding context. To code the transformed and quantized coefficient values, run/level symbols are generated after scanning with a zig-zag pattern. Each run or level symbol is binarized by unary coding and the binary value is coded with the corresponding run or level contexts. The sign value and the last symbol indicate whether the level is the last one in the block should be followed to each level. The sign value is coded with fixed length coding and the last symbol is coded with the arithmetic engine.

Some entropy coding technical details are summarized below:

- 9 bits are used for storing current probability of the LPS;
- 14 bits are used for representing probability range.
- 10 bits are used for representing context model initialization values.

4.3. Summary

This chapter outlines the coding tools of Versatile Video Coding (VVC) and Essential Video Coding (EVC). Having had a good understanding of H.264, HEVC, VVC and EVC codecs, the next chapter describes one of the multimedia applications, Transcoding, and its functionalities in detail.

5.1. Introduction to Video Transcoding

The Video Transcoding is a process of altering a video sequence from one format to another. Here, the format conversion means in terms of “range of operations” [52] [53]. That is conversion from one compressed format (extension) to another, bitrate change, changing the header information. Other than these basic format changes, a transcoder is used for many other applications, like framerate change, spatial resolution change, statistical multiplexing, adding information like adding company logo, digital watermark or to enhance error resilience [54]. Some of these applications are shown in Figure 5.1.

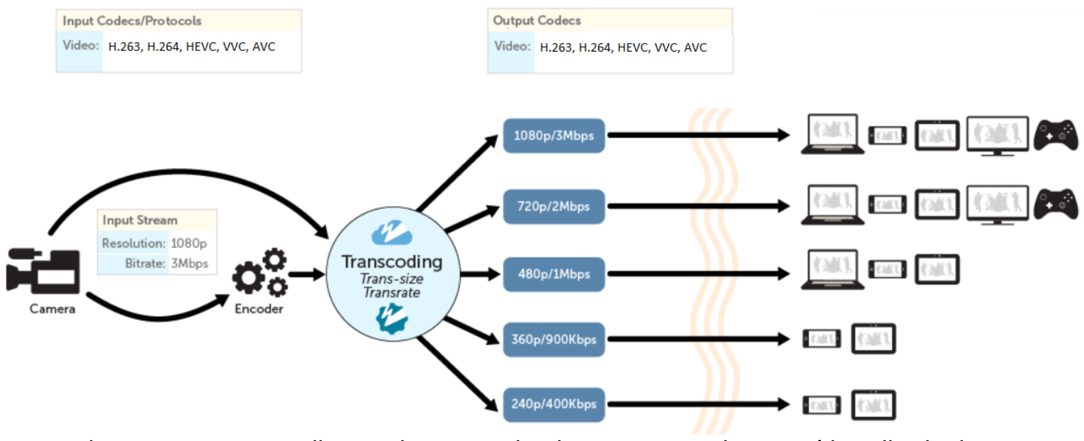


Figure 5.1. Transcoding and communication across various multimedia devices

5.2. Transcoding Architectures

There are different standard transcoding architectures for changing bit rate, spatial resolution, format conversion [52] [55]. They are

1. Open Loop Transcoding Architecture
2. Closed Loop Transcoding Architecture
3. Cascaded Pixel Domain Architecture
4. Motion Compensation in the DCT domain.

Open loop system is considered very simple since we do not store the frame memory and we do not need IDCT to achieve reduced rate. For achieving better coding efficiency and improved quality, re-quantization approach is used followed by the variable-length codes as shown in Figure 5.2. However, in the open-loop architectures, we see a drift effect. Drift effect is the loss of high frequency information in the bit-stream.

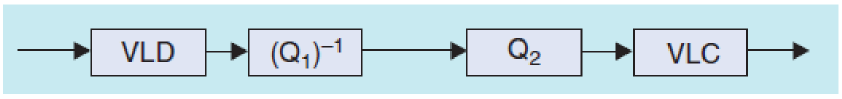


Figure 5.2. Open Loop, partial decoding to DCT coefficients and further re-quantizing [52]

In general, Figure 5.3. shows a closed-loop system. This system aims at removing the mismatch between residual and predictive components by approximating the cascaded pixel decoder-encoder architecture. This simplified scheme requires only one reconstruction loop with one DCT and one IDCT. With the exception of this slight inaccuracy, this architecture is mathematically equivalent to a cascaded decoder-encoder approach.

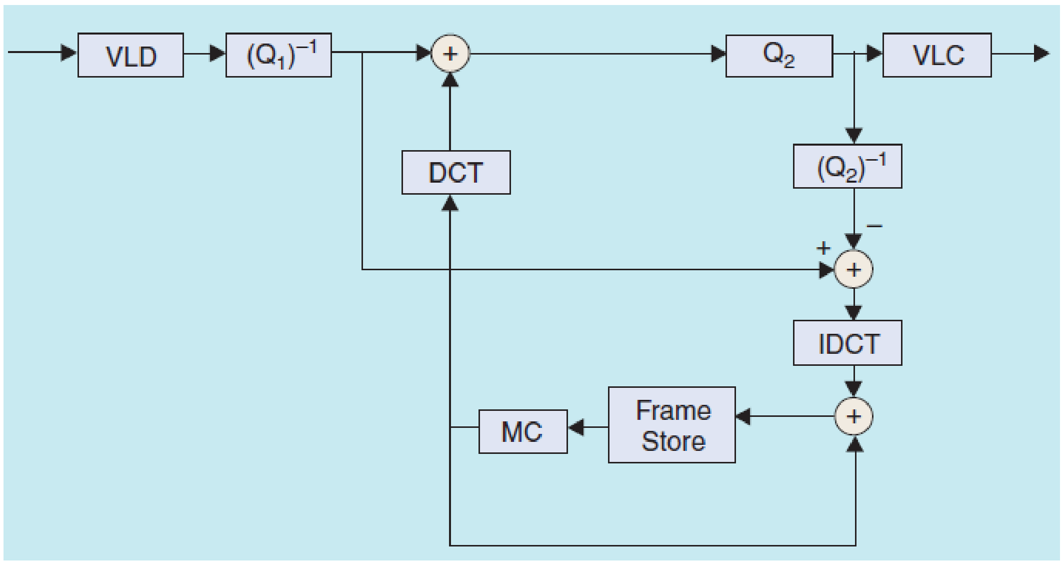


Figure 5.3. Closed-loop, drift compensation for re-quantized data [52]

Figure 5.4. shows cascaded decoder-encoder architecture. The main difference in the structure of cascaded decoder encoder architecture and closed-loop architecture is that the reconstruction of the bitstream in the cascaded pixel-domain architecture is done in the spatial domain. Hence it requires two reconstruction loops with two IDCTs and one DCT.

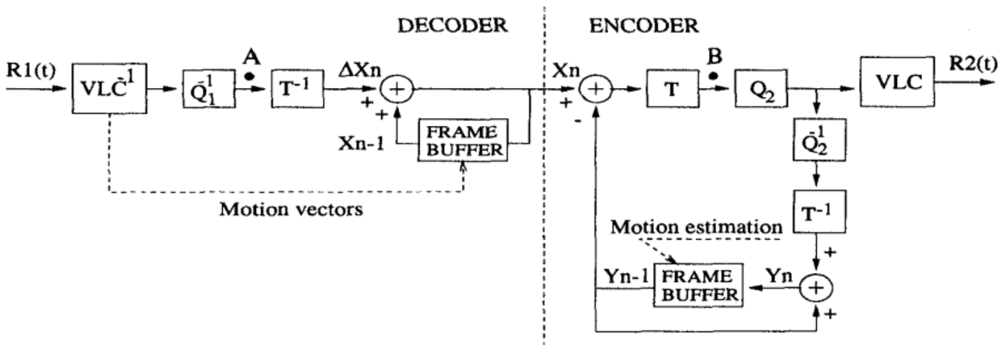


Figure 5.4. Cascade decoder-encoder architecture [52]

The closed-loop architecture described in the section 5.3. provides an effective transcoding structure in which the MB reconstruction is performed in the DCT domain. However, since the memory stores spatial domain pixels, the additional DCT/IDCT is still needed. This can be avoided though by utilizing the compressed-domain methods for MC proposed by Chang and Messerschmidt. In this way, it is possible to reconstruct reference frames without decoding to the spatial domain; several architectures describing this reconstruction process in the compressed domain have been proposed. It was found that decoding completely in the compressed-domain could yield equivalent quality to spatial-domain decoding. However, this was achieved with floating-point matrix multiplication and proved to be quite costly.

Different transcoding architectures for spatial resolution reduction, temporal resolution reduction like Motion Vector Mapping, DCT-Domain Down Conversion, Conversion of MB Type, Motion Vector re-estimation, Residual re-estimation are discussed in the coming sections.

5.3. Choice of Transcoding Architecture

The cascaded pixel domain transcoding architecture gives optimum results in terms of complexity, quality and cost [52]. The cascaded pixel domain transcoder offers greater flexibility in the sense that it can be used for bit rate transcoding, spatial/temporal resolution downscaling and for other coding parameter changes as well. Since in the case of standards format transcoding, it is required to take into consideration the different coding characteristics of H.264, HEVC and VVC, hence the flexibility of conversion between the codec parameters is a key issue.

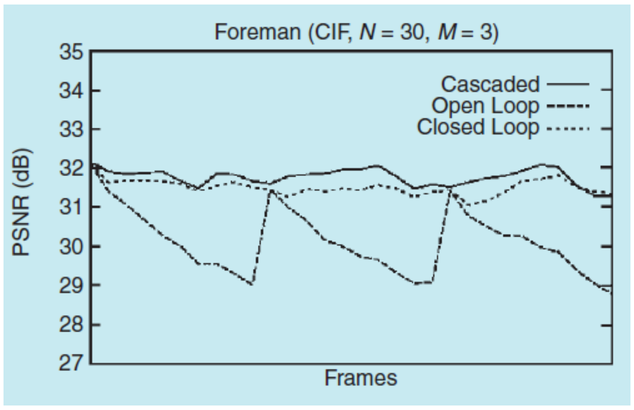


Figure 5.5. Frame based comparison of open loop, closed loop and cascaded pixel domain architecture

It is evident from Figure 5.5. that the open-loop architecture suffers from severe drift, and the quality of the simplified closed-loop architecture is very close to that of the cascaded pixel-domain architecture. Cascaded pixel-domain scheme is considered as ideal transcoder since it comprises of one full decoder and one full encoder. Another benefit of this approach is that decoding is usually fast since it does not involve motion estimation and predictions can be made for frames based on variable length decoding (VLD) of motion vectors from the encoded bitstream. The quality of transcoded video in turn is dependent upon the input to encoder stage. So better the input to encoding stage of transcoder, better the end video quality. Figure 5.6. denotes a general block diagram of the proposed cascaded-pixel-domain approach

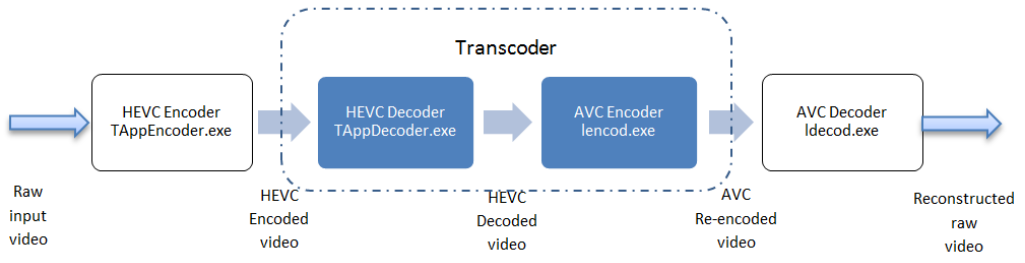


Figure 5.6. General block diagram of the proposed transcoding architecture (cascaded-pixel-domain)

5.4. Importance of Video Transcoding

Video compression standards significance is due to the bandwidth usage, display rate, network connectivity, computational capacity and others available to the end user in the most convenient way. For reproducing and delivering a video and other multimedia data flexibly according to the end-user's capability and requirement, video content should be dynamically adapted to the user's environment. Figure 5.7. and 5.8. shows the functionalities in video transcoding.

Need for transcoding:

a. When the target device does not support the format of the data.

b. When the target device does not have enough space or bandwidth.

c. When the format of the data is outdated or obsolete.

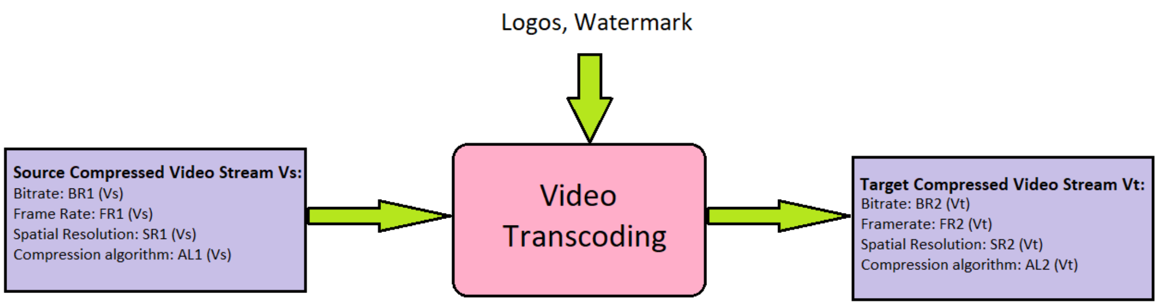

Logos, Watermark
Source Compressed Video Stream Vs:
Bitrate: BR1 (Vs)
Frame Rate: FR1 (Vs)
Spatial Resolution: SR1 (Vs)
Compression algorithm: AL1 (Vs)
Video Transcoding
Target Compressed Video Stream Vt:
Bitrate: BR2 (Vt)
Framerate: FR2 (Vt)
Spatial Resolution: SR2 (Vt)
Compression algorithm: AL2 (Vt)

Figure 5.7. Video Transcoding functionalities

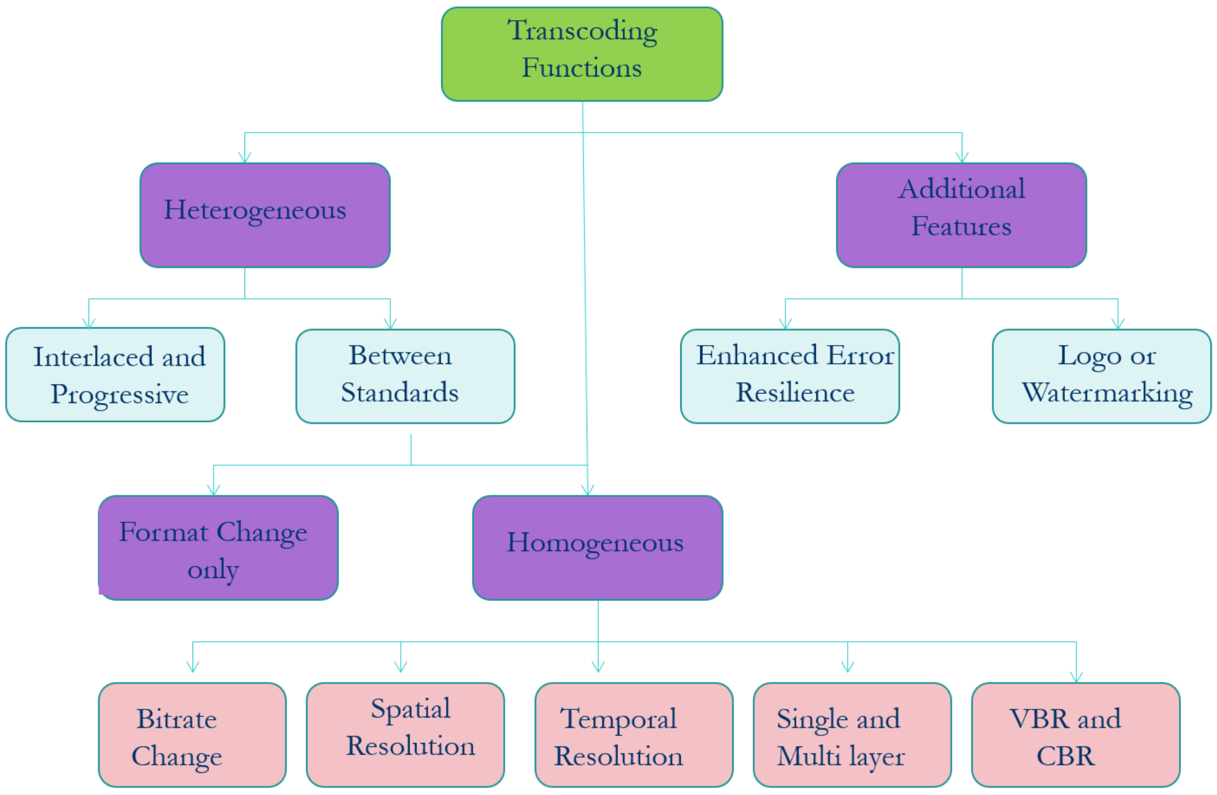


Figure 5.8. Transcoding Functions

5.5. Criteria to obtain optimal results after transcoding

1. The quality obtained by direct decoding and re-encoding of the output bitstream and the quality obtained by the transcoded bitstream should be comparable.
2. To avoid multigeneration deterioration, it is advised to use the information in the input stream as much as possible.
3. This process should be low in complexity, cost efficient and should achieve the best quality possible.

5.6. Summary

Transcoding is one of the main applications in video technology. This chapter covers the different types of Transcoding techniques available and the advantages and disadvantages of each type. The functionalities of transcoding is also pointed out. The next chapter focuses on formulating the research problem and addressing the methodologies of the objectives defined.

6.1. Identification of Research Gap

Heterogeneous/Homogeneous video transcoding for video codecs H.264, HEVC, VVC and EVC – Transcoding is one of the main process in video communication. This method is always in demand till the developments of new video codecs ongoing [55]. The VVC (Versatile Video Coding) and EVC (Essential Video Coding) being the next generation video codecs is the main focus point as the codec brings a lot of application uses along with its development. Since VVC and EVC are still not out in the market, there is a lot of scope in transcoding other codecs to VVC/EVC and vice versa. All these video codecs are open source other than EVC, and used in the industry standards, it is advantages to choose them as the main codec. Along with heterogeneous functionality, we can also look at homogeneous properties like bitrate change and frame rate change which are the main features for wireless communication.

6.2. Research Questions

The questions related to research that arise after a thorough literature review are as follows:

1. What video codecs to use for heterogeneous transcoding and the reason for its choice?
   - H.264 is the most used codec in the market covering more than 80% of video traffic [56]. HEVC codec as it is the latest codec out in the market by the ISO-ITU-T organization. VVC and EVC codec as they are still not out in the market and have a lot of application uses it come with.
2. Can a comparative study be made to check which codec performs better compression and which codec is more efficient with its implementation using open source software’s?
   - Yes. This is the initial stage where the implementation of each of the codecs is done separately along with looking at the comparative study with respect to its performance and efficiency.
3. Which transcoding architecture is chosen to obtain optimal results with good quality output and coding efficiency?
   - ‘Cascaded Pixel Open Loop Transcoding Architecture’ is looked upon as it has the best output video quality after the transcoding application. We also see the drifting effect that exists in the other architectures can be ignored in this architecture.

4. Can hybridizing heterogeneous and homogeneous transcoding lead to a seamless transcoding with improved features like bitrate reduction, framerate reduction?
   - Yes. Hybridizing the transcoding architecture leads to a better video compression. With improvised algorithms in the architecture, provide better quality reconstructed video.
5. What methods are considered to validate the transcoding algorithm?
   - There are a lot of measuring tools for testing the quality and performance of a reconstructed video. Here, for the quality of video, PSNR, SSIM, MSE, Bitrate, BD-PSNR, BD-SSIM for the quality of the video. Time is calculated for the encoding and decoding process to check the performance of the video codec and the transcoder.

These questions are triggered after looking at a detailed literature review and analyzing many research papers and thesis documents in video codec domain. Some of these questions are addressed in the following objectives of this thesis.

6.3. Title and aim of the thesis

The title of the thesis is as stated below:

"*Heterogeneous Transcoding for next generation multimedia video codecs for efficient communication* "

Aim: The aim of this thesis is to implement the next generation video codecs (i.e. VVC and EVC) along with the existing video codecs (i.e. H.264 and HEVC) for seamless video communication. The analysis of each block in the video codecs is performed for a comparative study between the four codecs in terms of better compression, better output quality video, complexity of codec and time efficiency. Transcoding mechanism using cascaded pixel decoder-encoder algorithm is also implemented with heterogeneous functionalities between all four video codecs.

6.4. Objectives of the Thesis

Versatile Video Coding by ISO-ITU-T organization is the emerging video codec with new application usages and a lot of avenues in the field of research. This codec will be out in the market in October 2020 and as of now only the test version of the codec is available as open source [59]. Essential Video Coding is the next generation video coding out by the ISO-IEC-MPEG organization having royalty free software [60] and enhanced switching options. Since these video compression codec is still not out in the market, a lot of avenues in transcoding, multiplexing and 4K, 8K and 12K content compression is the hot topic of research in the coming years. The other two codecs used in the transcoding are H.264 and HEVC along with VVC and EVC. HEVC being the newest codec out in the market by ISO organization and H.264 is the most used codec for data transfer communication is the reason for its choice. Based on these reasonings, the below objectives are decided for the thesis.

1. To develop a framework for a comparative analysis between H.264, HEVC, VVC and EVC along with the implementation of the codecs.
2. To develop a hybrid transcoding architecture for format change (Heterogeneous in nature)
3. To analyze and evaluate a hybrid transcoding architecture for bitrate change using QP variation (Homogeneous and Heterogeneous) and frame rate improvisation (Homogeneous).
4. To develop an automated versatile application to hybridize the transcoding features from H.264, HEVC, VVC and EVC.

6.5. Methodology for each objective

The methodology for all the objectives is explained in detail. The methodology also focusses on the tools and techniques used for the development of the models, system, and analysis of the same. The methodologies explained in this section are also based on the previously available literature on the same topic of interests.

6.5.1. Objective 1 Methodology: To develop a framework for a comparative analysis between H.264, HEVC, VVC and EVC along with the implementation of the codecs.

VVC and EVC being the next generation video codecs with a lot of application functionalities along with H.264 and HEVC being the existing codecs in the market are the reason for its choice for transcoding. The framework for its comparative study is as shown below.

1. To implement H.264 codec using Joint Model (JM) software. To analyze and evaluate the performance of H.264 codec blocks for encoding and decoding standard definition (SD) and high definition (HD) video contents.
2. To implement HEVC codec using Hybrid Model (HM) software. To analyze and evaluate the performance of HEVC codec blocks for encoding and decoding standard definition (SD), high definition (HD) and 4K video contents.
3. To implement VVC codec using Video Test Model (VTM) software. To analyze and evaluate the performance of VVC codec blocks for encoding and decoding standard definition (SD) high definition (HD) and 4K video contents.

4. To implement EVC codec using EVC Test Model (ETM) software. To analyze and evaluate the performance of EVC codec blocks for encoding and decoding standard definition (SD) high definition (HD) and 4K video contents.
5. The performance for each codec is evaluated using metrics like PSNR, SSIM, Bitrate, compression ratio, time taken to encode and decode to provide a graphical comparative explanation for each codec.
6. H.264, HEVC, VVC and EVC codec output values to be obtained by keeping the parameters QP = 32; All filters ON, Profile = MAIN, and same GOP size value =16 and Intra period =16 (IBBBBBBBBBBBBBBBI). This is tested for 5 different contents of different resolutions for all 4 codecs.
7. Rate distortion graph obtained for PSNR and bitrate values of a content and each codec for different quantization parameter (QP) values. Here we see that all Filters are ON. The profile used for all four codecs is main profile. The Group of Picture (GOP) is 16 for all tests. The QP value tested for obtaining the curve is 10, 20, 30, 40 and 50. The intra prediction is 16 (IBBBBBBBBBBBBBBI).

6.5.2. Objective 2 Methodology: To develop a hybrid transcoding architecture for format change (Heterogeneous in nature) and bitrate change (Homogeneous and Heterogeneous in nature).

Transcoding can be of two types, Heterogeneous and Homogeneous in nature. Heterogeneous is the transcoding feature being implemented between two different video codecs. Homogeneous is the transcoding feature being implemented for the same video codec. Format change can be achieved using Heterogeneous Transcoding. Bitrate change can be achieved using both Homogeneous and Heterogenous Transcoding. Steps to achieve the same are as shown below

1. **Format Change:** The below block diagram in Figure 6.1. to 6.4. shows the design of the development of heterogeneous transcoding for format change. The cascaded pixel encoder decoder algorithm is designed and developed for this. This is for the conversion between H.264 to VVC, HEVC or EVC. The same flow is repeated for the other codec conversions.

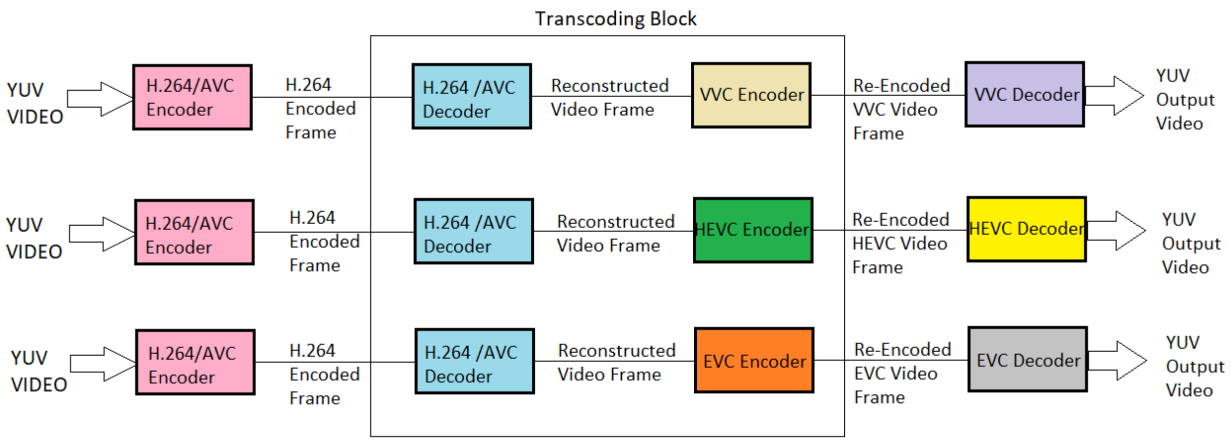


Figure 6.1. Heterogeneous Transcoding from H.264 to HEVC, VVC or EVC

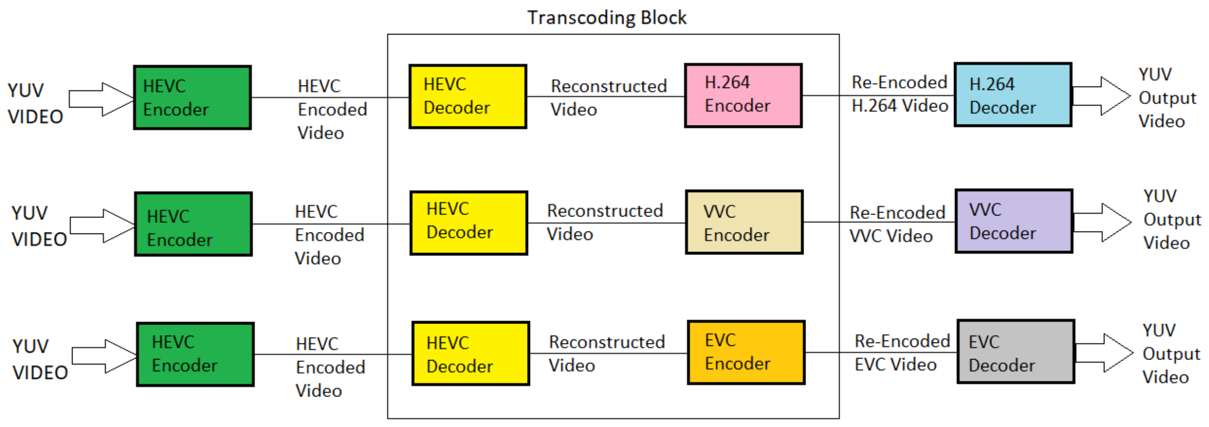


Figure 6.2. Heterogeneous Transcoding from HEVC to H.264, VVC or EVC

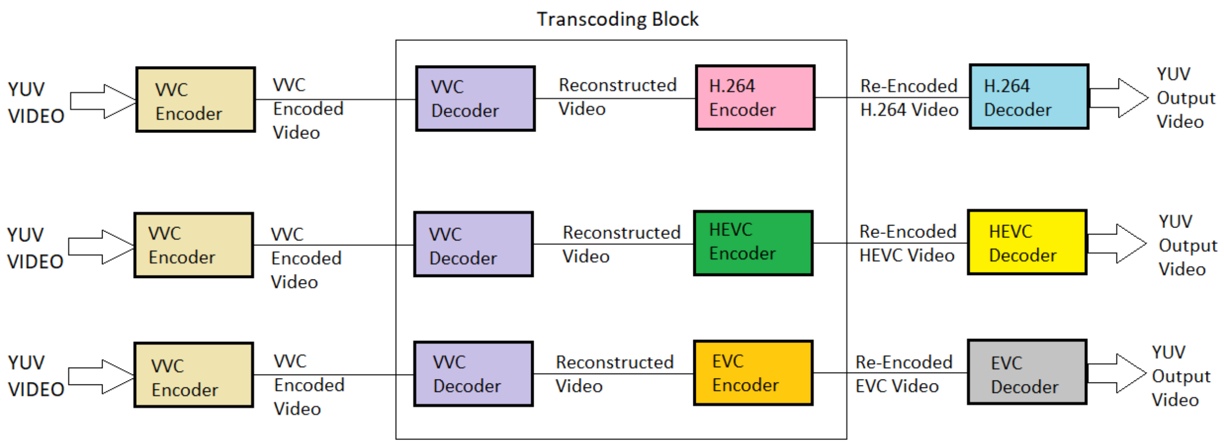


Figure 6.3. Heterogeneous Transcoding from VVC to H.264, HEVC or EVC

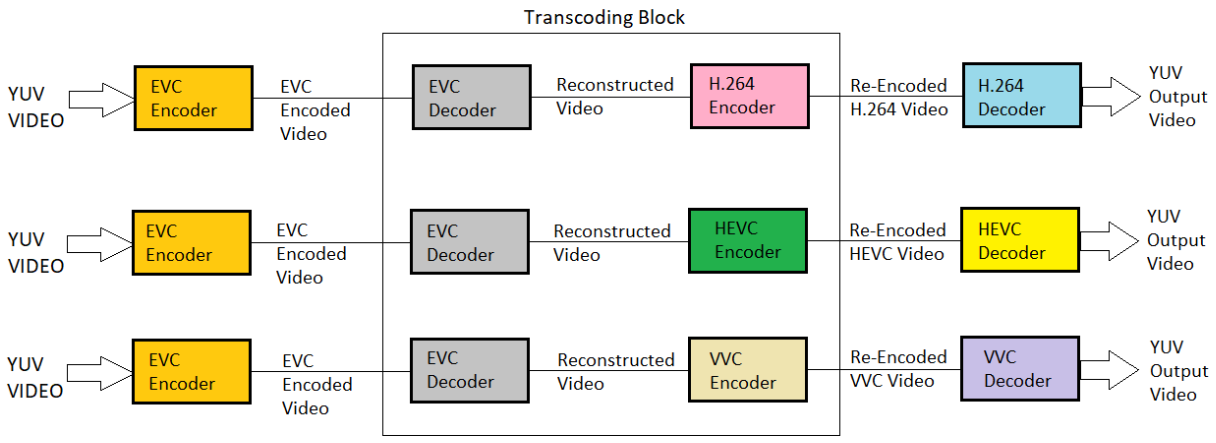


Figure 6.4. Heterogeneous Transcoding from EVC to H.264, HEVC or VVC

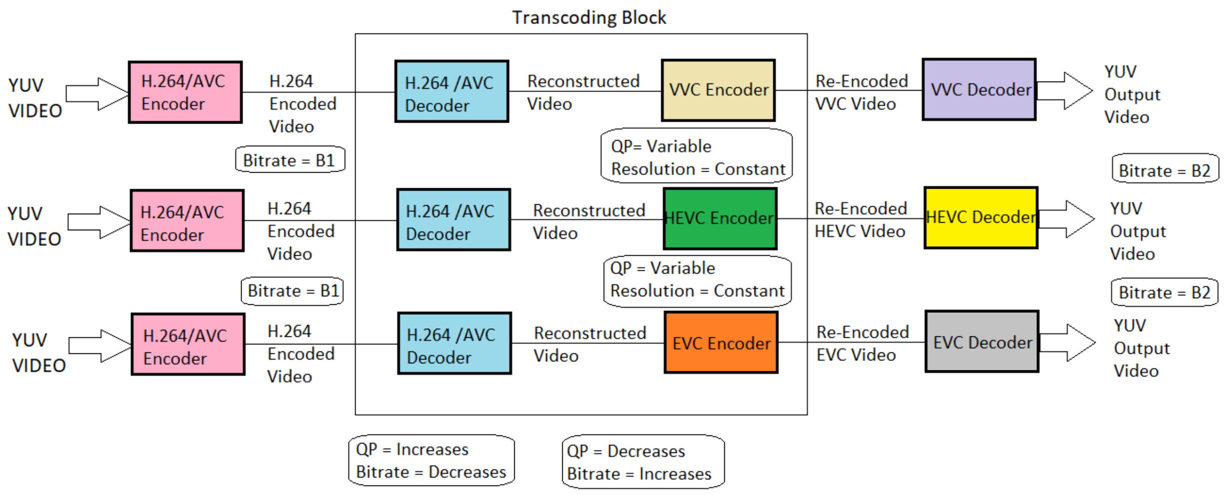


Figure 6.5. Block diagram of heterogeneous transrating of H.264 to VVC, HEVC and EVC using QP variation

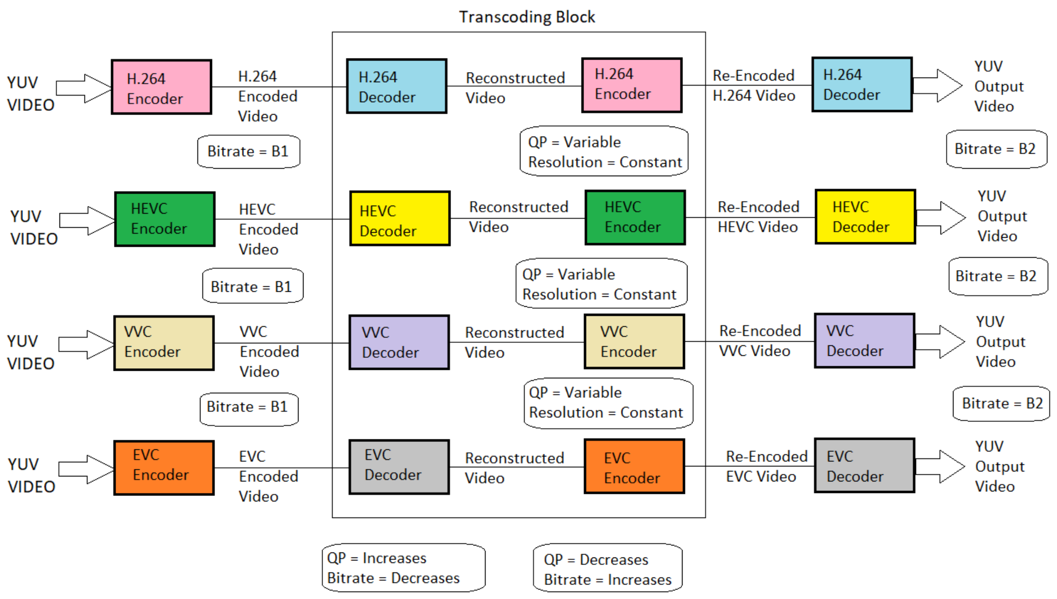


Figure 6.6. Block diagram of homogeneous transrating of H.264, HEVC, VVC and EVC using QP variation

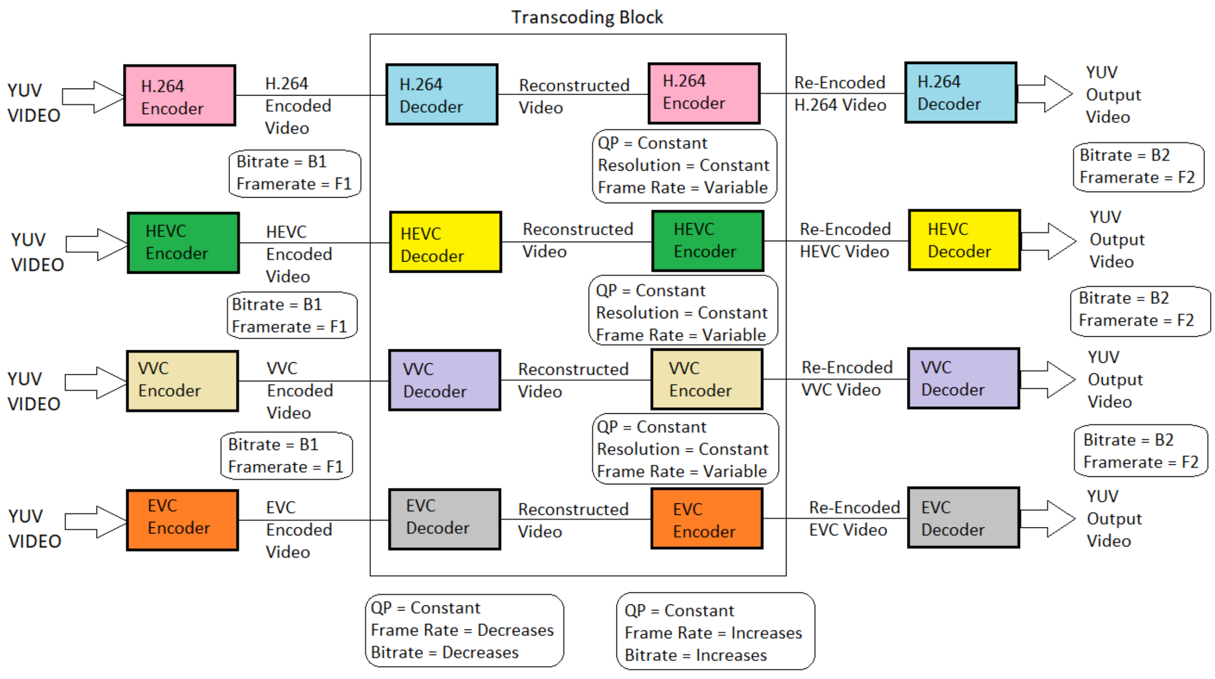


Figure 6.7. Block diagram of homogeneous framerate change of H.264, HEVC, VVC and EVC using framerate improvisation

6.5.3. Objective 3 Methodology: To analyze and evaluate a homogeneous and heterogeneous transrating architecture for QP variation and homogeneous transrating architecture for framerate change.

1. **Bitrate change with QP variation:** Changing bitrate is also known as transrating. Homogeneous and heterogeneous bitrate change (Figures 6.5. and 6.6.) can be observed by varying the quantization parameter in the transcoding block for the reconstructed video frame. The QP is changed in such a way that the transcoded output quality video is not distorted. The resolution is kept constant. If we increase the QP, the bitrate decreases. If we reduce the QP, the bitrate increases. The block diagram in Figure 6.5. and 6.6 shows for both homogeneous and heterogeneous transcoding architectures for four codecs H.264, HEVC, VVC and EVC.
2. **Bitrate change with framerate variation:** When the framerate decreases, it was observed that the bitrate also decreases. This drop in bitrate was even more than the homogeneous bitrate change using cascaded pixel decoder-encoder architecture using the same QP as the predecessor. This is mostly observed in the homogeneous environment and the block diagram for the same is shown in Figure 6.7.

6.5.4. Objective 4 Methodology: To develop an automated versatile application to hybridize the transcoding features from H.264, HEVC, VVC and EVC.

Developing an automated application to create a video transcoder and including the below features.

1. Obtain transcoding across these four codecs (H.264, HEVC, VVC and EVC) on the fly.
2. The parameter choices can be chosen on the fly.
3. Heterogeneous and homogeneous features being enabled with in the application.
4. Framerate, bitrate, format change done with just a command line.

6.6. Original Contribution:

The originality of this thesis tries to cover the research gaps found in the extensive literature review from the technical papers and previous theses. They are listed as below.

1. Implementation of VVC codec with its encoder and decoder blocks along with its overview study as this codec is still not out in the market and no overview paper available.
2. Implementation of EVC codec with its encoder and decoder blocks along with its overview study as this codec is still not out in the market and no overview paper is available.
3. Comparative study of all the 4 codecs (H.264, HEVC, VVC and EVC) in terms of encoding time, decoding time, PSNR, SSIM, BD-Rate and BD-PSNR measurements.
4. Design and development of heterogeneous format transcoding of VVC to H.264, VVC to HEVC, VVC to EVC, H.264 to VVC, HEVC to VVC and EVC to VVC using cascaded pixel domain architecture.
5. Design and development of heterogeneous format transcoding of EVC to H.264, EVC to HEVC, EVC to VVC, H.264 to EVC, HEVC to EVC, VVC to EVC using cascaded pixel domain architecture.
6. Design and development of homogeneous transcoding for VVC and EVC codecs in terms of bitrate change by QP variation and frame rate variation.

7.1. Introduction

A systematic evaluation and a comparative analysis of the codecs with respect to video compression, coding efficiency and quality of the compressed video is first examined for Advanced video coding (AVC/H.264), High efficiency video coding (HEVC), Versatile video coding (VVC), and MPEG-5 Essential Video Coding (EVC). Both subjective and objective quality analysis is obtained for each codec [61]. In the second part, a transcoder app is developed to covert one form of codec to another with soma parameters variations.

7.2. Video Quality Measures

The performance of a codec is evaluated using two methods. They are

- Objective quality measures – PSNR, MSE, Bitrate
- Structural quality measure – SSIM [61]
- Subjective quality measures – Human Visual System (HVS) [62]

Lossless and lossy compressions use different methods to evaluate compression quality. Standard criteria like compression ratio, execution time, etc. are used to evaluate the compression in lossless case, which is a simple task whereas in lossy compression, it is complex in the sense, it should evaluate both the type and amount of degradation induced in the reconstructed video .The goal of video quality assessment is to accurately measure the difference between the original and reconstructed video, the result thus obtained is used to design optimal image codecs. The objective quality measure like PSNR, measures the difference between the individual pixels of an original video frame and reconstructed video frame. It is dependent on the mean square error (MSE) of the reconstructed video.

$$MSE = \frac{1}{M * N} \sum_{m=1}^{M} \sum_{n=1}^{N} \left[x(m,n) - y(m,n) \right]^2$$

$$PSNR = 10 \log_{10} \frac{L^2}{MSE}$$

Here the x is the original video frame and y is the reconstructed video frame. M and N are the width and height of the video frame. L is the maximum pixel value in NxM pixel video frame.

The SSIM is designed to improve on traditional metrics like PSNR and MSE (which have proved to be inconsistent with human visual perception) and is highly adapted for extracting structural information.

The SSIM index is a full reference metric, in other words, the measure of image quality is based on an initial uncompressed or distortion free video as reference. The SSIM measurement system is shown in Figure 7.1.

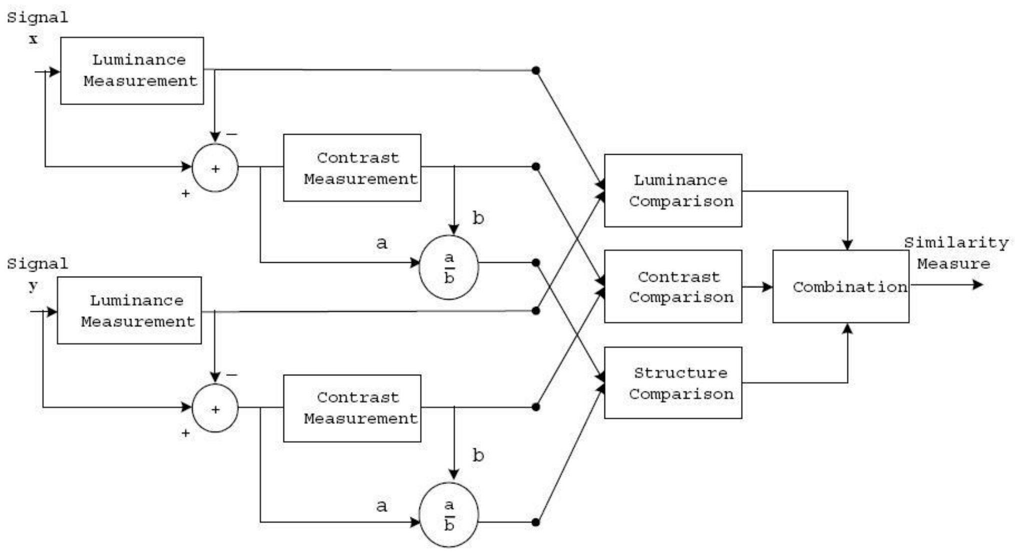


Figure 7.1. Structural Similarity Index Metric (SSIM) measurement system

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \qquad c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \qquad s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3},$$

where $\mathbf{x}$ and $\mathbf{y}$ correspond to two different signals that we would like to match, i.e. two different blocks in two separate images, μ_x, σ_x^2, and σ_{xy} the mean of $\mathbf{x}$, the variance of $\mathbf{x}$, and the covariance of $\mathbf{x}$ and $\mathbf{y}$ respectively, while C_1, C_2, and C_3 are constants given by $C_1 = (K_1L)^2$, $C_2 = (K_2L)^2$, and $C_3 = C_2/2$. L is the dynamic range for the sample data, i.e. *L*=255 for 8-bit content and K_1<<1 and K_2<<1 are two scalar constants. Given the above measures the structural similarity can be computed as

$$SSIM(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma}$$

where α, β and γ define the different importance given to each measure.

Bitrate is the number of bits per second. The symbol is bit/s. It determines the size and quality of a video or audio file. The higher the bitrate, the better the quality and the larger the file size. A human visual system model (HVS model) [63] is also termed as the subjective analysis which determines the quality of a video by the naked eye which is explained in detail in the following sections.

7.3. Test Sequence Set

The test sequences were chosen keeping in mind the different variations that we can observe between the frames. The test sequence set have wide range of resolutions (176x1444, 640x360, 720x486, 1280x720, 1920x1080 and 3840x2160) with few frames. The test contents were chosen in such a way that there were fast movement or slow movement between frames, lot of color variations in the content, lot of details like edges and sudden change in color and human face or skin. These test sequences were taken from two main sites Xiph [64] and Ultra video group [65]. Some contents were in y4m format and had to be converted to yuv since some codecs supported only yuv contents.

Test Sequence	Resolution	Number of frames	Original Size (kB)	Description	Source
Akiyo	176X144	300	11138	Color video with minimal change between frames	Xiph.org
SpeedBag	640x360	120	40500	Color video with minimal change between frames	Xiph.org
Shield2	640x360	120	40500	Color video with lot of clolor variations and movement between frames	Xiph.org
MobileCalendar	720x486	360	246038	Color video with lots of details and contains a great variation of color and a large amount of texture	Xiph.org
Stockholm	1280x720	604	815400	Color video with smaller objects moving with each frame	Xiph.org
FourPeople	1280x720	601	811350	Color video with lots of detailing.	Xiph.org
Beauty	1920x1080	600	1822500	Color video with many basic human face and skin	Ultra Video Group
SunBath	3840x2160	300	3645000	Color video with less color variation and movement	Ultra Video Group
Bhosphorus	3840x2160	600	7290000	Color video with sharp detailing and movement	Ultra Video Group

Table 7.1. Test sequences description

Akiyo

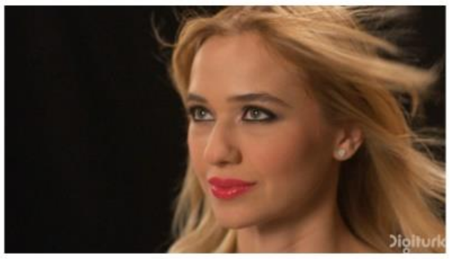
SpeedBag

Shield2

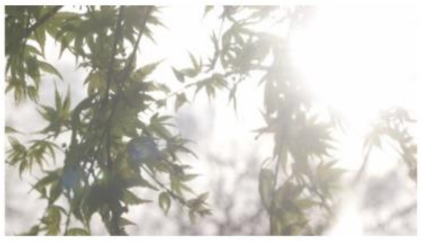
MobileCalendar

Stockholm

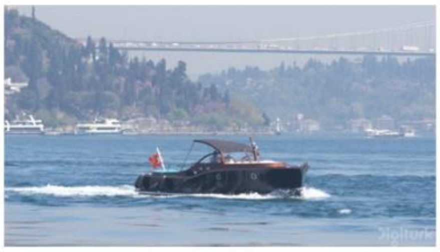
FourPeople



Beauty



SunBath



Bhosphorus

Figure 7.2. Test set used for testing [64] [65]

7.4. Results of Objective 1: Comparative analysis of the codecs

There are two distinct lines in the future video coding technology development work. VVC [16][66] driven by Joint Video Exploration team (JVET) and EVC [67] driven by Moving Picture Expert Group (MPEG). These two codecs are the future codecs and the extended versions with respect to the advances in compression technology to HEVC.

Versatile Video Coding (VVC) standard which is under development by the JVET team of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11 was out on July 6th, 2020. The latest software version of VVC available is VTM-10.2 [59]. This codec promises 40% more compression than its predecessor (HEVC) [16] with having the same perceptual quality of video. VVC supports lossless and subjectively lossless compression with resolutions varying from 4K to 16K along with 360° videos. It supports video contents of 8 to 16 bit depth with chroma formats ranging from 4:2:0 to 4:4:4. Some applications of VVC are high dynamic range (HDR) video, multiview coding, still picture coding, panoramic formats. Complexity of the codec was observed to be 10 times more than that of HEVC [16], but that is waived off with the better quality of the video output with more compression achieved.

Essential Video Coding (EVC) [67] standard which is under development by the MPEG team of ISO/IEC JTC 1/SC 29/WG 11 is due by mid-2020. The latest software version of EVC available is ETM-4.0 [60]. The coding efficiency of EVC is almost like HEVC with slightly better video quality than HEVC [56]. It is developed with some licensing conditions, i.e. royalty-free for the baseline profile and with IPR for the main profiles.

High Efficiency Video Coding (HEVC) [11] also known as H.265 standard was developed by Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP-3 and ISO/IEC JTC1/SC29/WG11 and published the first version in June 2013. The next version included applications like range extensions (RExt), Multiview extensions (MV-HEVC) and scalability extensions (SHVC) and published in 2015. The coming versions applications included extensions in 3D video and screen content coding (SCC) which were published in the 1st quarter of 2017. HEVC promised in offering 30%-50% better compression of video with respect to H.264 [68] and having the same perceptual quality of the video. It supported video contents up to 8K resolution with bit depth from 8 bit to 12 bit. The latest software version of HEVC is HM-16.0 [58].

Advanced Video coding (AVC) also known as H.264 [69] was developed by Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6) and published in 2003. H.264 codec is the first codec to support HD videos and with a very large video compression possibility with good perceptual output video quality. In today’s time, it is the most commonly used video format with 91% of the video industries using it for distribution, compression and recording of the video content [56]. With the later years of its initial development, some new features were added to the codec, i.e. Fidelity range extensions, scalable video coding, Multiview video coding, 3D-AVC and MFC stereoscopic coding. H.264 promised 50% better bitrate efficiency when compared to MPEG-2 part 2 [70]. It supported contents of bit depth 8 to 14 bits and chroma formats from 4:2:0 initially in baseline profile to 4:4:4 at the later stage for high profile. This codec is known for its broad application range like digital video compression at low-bitrate, internet streaming to broadcasting, and digital cameras for nearly lossless coding. The latest software version of AVC is JM-19.0 [57].

Three methods for comparisons are performed for H.264, HEVC, VVC and EVC:

1. The first method is an objective quality metric comparative study between the codecs in terms of PSNR, SSIM, Bitrate, encoding time, decoding time and compression size for each of the contents specified in Tables 7.3 through 7.6. keeping the QP constant to 32, GOP=16 (IBBBBBBBBBBBBBBBI)
2. The second method is the rate distortion or RD method where the rate distortion graph is obtained for a content with different QP values (10,20,30,40 and 50) and same GOP =16 (IBBBBBBBBBBBBBBBI) for all four codecs.
3. The third method is the subjective quality analysis where the comparisons of the codecs are observed by the visual analysis of the videos. The rate distortion method outputs for QP values (10, 20, 30, 40 and 50) are observed for frame by frame and the distortion noted for codecs H.264, HEVC, VVC and EVC.

Features	H.264	HEVC	VVC	EVC
Block Structure	Largest block size is 16x16.	Quad Tree CTU size upto 64x64	QTBT + Ternary Tree (TT) CTU size upto 256x256	Quad Tree + Ternary Tree CTU size upto 128x128
Intra Prediction	5 luma prediction modes, 4 chroma prediction modes (9 Intra prediction modes)	35 Intra prediction modes	81 prediction modes (in which 65 are angular)	30 prediction modes (inc. angular, DC, plane, and bilinear)
Inter Prediction	Block based motion compensation and motion vector prediction	Hierarchical weighted prediction (P, B frames) PU level motion vector prediction. Motion vector difference ¼ pel MV accuracy. Block motion compensation Translation motion prediction.	Hierarchical weighted prediction (P, B frames). Sub-CU based motion vector prediction. Adaptive motion vector precision. Affine motion prediction. Decoder-side motion vector refinement.	Hierarchical weighted prediction (P, B frames). Adaptive motion vector resolution. Affine motion prediction. Decoder-side motion vector refinement.
Transform	2D-DCT sizes 4x4 up to 8x8	DCT-II and DST-VII. Transform block size 8x8, 16x16 and 32x32	Adaptive multiple core transform. Mode dependant non-separable secondary transforms (4x4) Transform block sizes 4x4 upto 64x64	DCT-II sizes 4x4 uoto 128x128
Loop filter + Other	Adaptive in-loop deblocking filter	Deblocking filter, SAO	Deblocking filter, SAO, Adaptive loop filter	Deblocking filter, Hadamard Transform

Entropy coding	CABAC and CAVLC	CABAC	Modified CABAC (with Context modelling for transform coefficient levels)	Multiplier based context adaptive entropy coding.

Table 7.2. Architectural comparative study of H.264, HEVC, VVC and EVC codecs

7.4.1. H.264 Encoder and Decoder Codec Output

Test Configurations of H.264 [57]:

- *Software Used*: Joint Model (JM) – version 19.0
- *Encoder Commandline*: lencod.exe -d encoder_main.cfg
- *Decoder Commandline*: ldecod.exe
- *System Type*: Dell Inspiron with i7 processor/ Windows 10/ 64 bit

In the encoder commandline, -d indicated the configuration file to be added which is in the directory 'JM\bin'. In our testing, we use encoder_main.cfg file and is edited according to our preferences. In the 'InputFile' location, add the destination where the content is available. Content parameters are given in the source width/height and accordingly output width/height. QPI/P/BSlice are changed according to the QP value we need after each test run. The change in parameters required in the config file is shown below:

```
#======== File I/O =========================================================
InputFile = "C:\Desktop\akiyo_qcif.yuv"       # Input sequence
InputHeaderLength = 0      # If the inputfile has a header, state it's length in byte here
StartFrame = 0      # Start frame for encoding. (0-N)
FramesToBeEncoded = 300      # Number of frames to be coded
FrameRate = 100.0   # Frame Rate per second (0.1-100.0)
SourceWidth           = 176    # Source frame width
SourceHeight          = 144    # Source frame height
SourceResize          = 0      # Resize source size for output
OutputWidth           = 176    # Output frame width
OutputHeight    = 144    # Output frame height
```

```
TraceFile          = "trace_enc.txt"      # Trace file

ReconFile          = "test_rec.yuv"       # Recontruction YUV file

OutputFile         = "test.264"           # Bitstream

IntraPeriod        = 16   # Period of I-pictures   (0=only first)

QPISlice           = 10  # Quant. param for I Slices (0-51)

QPPSlice           = 10  # Quant. param for P Slices (0-51)

NumberBFrames = 15  # Number of B coded frames inserted (0=not used)

QPBSlice           = 10 # Quant. param for B slices (0-51)
```

After the encoder, the binary file is 'test.264'. After the decoder, the output file is 'test.yuv' which is the reconstructed video. For our testing we have used GOP = 16 and QP =32 for all the test contents. Since H.264 does not support 4K content, the testing was done only till HD videos i.e. 1280x720. The quality was verified with respect to PSNR, SSIM, MSE and Bitrate. The performance was checked with respect to the time taken to encode and decode and measured in seconds. The compression size of the encoded bit stream was also collected which was measured in kB.

S.No	INPUT VIDEO FILE	PSNR (dB)	SSIM YUV	MSE YUV	BITRATE (kbps)	ENCODING TIME (s)	DECODING TIME(s)	COMPRESSION SIZE (kB)
1	Akiyo	36.80	0.94	15.55	106.39	323.238	0.093	39
2	Speedbag	41.05	0.97	5.89	141.63	1873.631	0.451	104
3	Shields2	34.09	0.88	28.74	862.01	1292.539	0.668	632
4	Mobile calendar	34.88	0.92	24.45	2486.94	8073.856	4.539	5465
5	Stockholm	36.10	0.89	19.90	2027.63	38512.408	10.091	7475
6	Fourpeople	39.67	0.95	8.67	401.52	27572.912	8.137	1473

Table 7.3. H.264 encoder and decoder statistics

7.4.2. HEVC Encoder and Decoder Codec Output

<u>Test configurations of HEVC [58]:</u>

- *Software:* Hybrid Model (HM) – version 16.0
- *Encoder Commandline*: TAppEncoder.exe -c encoder_randomaccess_main.cfg -c akiyo.cfg
- *Decoder Commandline*: TAppDecoder.exe -b str.bin -o dec.yuv -d 8
- *System Type*: Dell Inspiron with i7 processor/ Windows 10/ 64 bit

In the encoder commandline, -c indicated the configuration file to be added which is in the directory '\cfg' and '\cfg\per-sequence'. The cfg file per-sequence needs to be created by the user under any name as the content. Since here the content to be used is akiyo, we shall name the content file as 'akiyo.cfg' as the secondary cofig file. In our testing, we use encoder_randomaccess_main.cfg as the mainfile. In the main file, we change the values of 'IntraPeriod', 'GOPSize' to 16 and 'QP' parameter as shown in VVC software. The second confile file needs to be added with parameters as below:

```
#======== File I/O ==========================================================
InputFile              : C:\Desktop\akiyo.yuv
InputBitDepth          : 8        # Input bitdepth
InputChromaFormat      : 420      # Ratio of luminance to chrominance samples
FrameRate              : 120      # Frame Rate per second
FrameSkip              : 0        # Number of frames to be skipped in input
SourceWidth            : 176      # Input  frame width
SourceHeight           : 144      # Input  frame height
FramesToBeEncoded      : 300      # Number of frames to be coded
Level                  : 4.1
```

After the encoder, the binary file is 'str.bin'. After the decoder, the output file is 'dec.yuv' which is the reconstructed video. For our testing we have used GOP = 16 and QP =32 for all the contents. Since HEVC supports even 4K contents, the testing was done for contents with resolution from qcif i.e. 176x144 till 4K i.e. 3840x2160. The quality was verified with respect to PSNR, SSIM, MSE and Bitrate. The performance was checked with respect to the time taken to encode and decode and measured in seconds. The compression size of the encoded bit stream was also collected which was measured in kB.

Sl. No.	INPUT VIDEO FILE	PSNR (dB)	SSIM YUV	MSE YUV	BITRATE (kbps)	ENCODING TIME (s)	DECODING TIME (s)	COMPRESSION SIZE (kB)
1	Akiyo	39.250	0.98945	7.728	130.922	49.35	0.454	40
2	Shields2	35.947	0.98599	16.54	347.445	231.14	0.693	255
3	Stockholm	35.812	0.98276	17.06	4222.370	5501.18	11.149	2595
4	Beauty	38.6891	0.97482	8.795	2238.458	12172.03	23.354	1367
5	SunBath	42.921	0.99132	3.319	12965.12	37967.48	54.871	3957

Table 7.4. HEVC encoder and decoder statistics

7.4.3. VVC Encoder and Decoder Codec Output

Test Configurations of VVC Codec [59]:

- *Software*: VVC Test Model (VTM) – version 10.0
- *Encoder Commandline*: EncoderApp.exe -c encoder_randomaccess_vtm.cfg -c akiyo.cfg
- *Decoder Commandline*: DecoderApp.exe -b str.bin -o dec.yuv -d 8
- *System Type*: Dell Inspiron with i7 processor/ Windows 10/ 64 bit

This software is a test version which is still under development. Here we provide two config files. Second file is the content file which is same as the one used in HEVC as ‘akiyo.cfg’. The first config file is the main file as ‘encoder_randomaccess_vtm.cfg’ which specifies the parameters to be changed in order to get the output as desired as shown below:

```
#======== File I/O =====================================================
IntraPeriod : 16  # Period of I-Frame ( -1 = only first)
GOPSize     : 16  # GOP Size (number of B slice = GOPSize-1)
QP          : 10  # Quantization parameter(0-51)
```

After the encoder, the binary file is ‘str.bin’. After the decoder, the output file is ‘dec.yuv’ which is the reconstructed video. Since VVC supports 4K and 8K contents, the testing was done for contents with resolution from qcif i.e. 176x144 till 4K i.e. 3840x2160. The quality was verified with respect to PSNR, SSIM, MSE and Bitrate. The performance was checked with respect to the time taken to encode and decode and measured in seconds. The compression size of the encoded bit stream was also collected which was measured in kB.

Sl. No.	INPUT VIDEO FILE	PSNR (dB)	SSIM YUV	MSE YUV	BITRATE (kbps)	ENCODING TIME (s)	DECODING TIME (s)	COMPRESSION SIZE (kB)
1	Akiyo	40.2677	0.9879	97.8196	83.3984	446.137	1.099	26
2	Shields2	36.7821	0.9890	218.2678	192.6293	1727.356	1.279	142
3	Stockholm	36.1117	0.9690	254.7011	2356.0132	22901.051	19.528	1448
4	Beauty	38.9595	0.9967	132.2072	1691.5568	101368.613	45.738	1033
5	SunBath	43.8083	0.9990	43.2886	9755.6192	168480.477	104.085	2978
6	Bhosphorus	42.0861	0.9830	64.3559	890.18	192677.252	202.441	3260

Table 7.5. VVC encoder and decoder statistics

7.4.4. EVC Encoder and Decoder Codec Output

Test configurations of EVC codec [60]:

- *Software*: EVC Test Model (ETM) – version 4.0
- *Encoder Commandline*: evca_encoder.exe -i akiyo.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 120 -d 8 -o str.bin –config encoder_randomaccess.cfg
- *Decoder Commandline*: evca_decoder.exe -i str.bin -o dec.yuv
- *System Type*: Dell Inspiron with i7 processor/ Windows 10/ 64 bit

EVC software is also a test model and is still under development. It is not open source and requires login details to download the software which will be provided by the MPEG team on request for research use. In the Encoder commandline the following notations denotes '-i' - the path to the input file, '-q'- the quantization parameter, '-w' - the source width, '-h' - source height, '-p' - the intra period, '-f' - the number of frames to be coded, '-z'- the frame rate, '-d'- the input bit depth, '-o'- output encoded file and '-config'- encoder configuration file.

For every test run, we need to change the '-q' value in the encoder commandline. After the encoder, the binary file is 'str.bin'. After the decoder, the output file is 'dec.yuv' which is the reconstructed video. Since EVC supports 4K and 8K contents, the testing was done for contents with resolution from qcif i.e. 176x144 till 4K i.e. 3840x2160. The quality was verified with respect to PSNR, SSIM, MSE, Bitrate. The performance was checked with respect to the time taken to encode and decode and measured in seconds. The compression size of the encoded bit stream was also collected which was measured in kB.

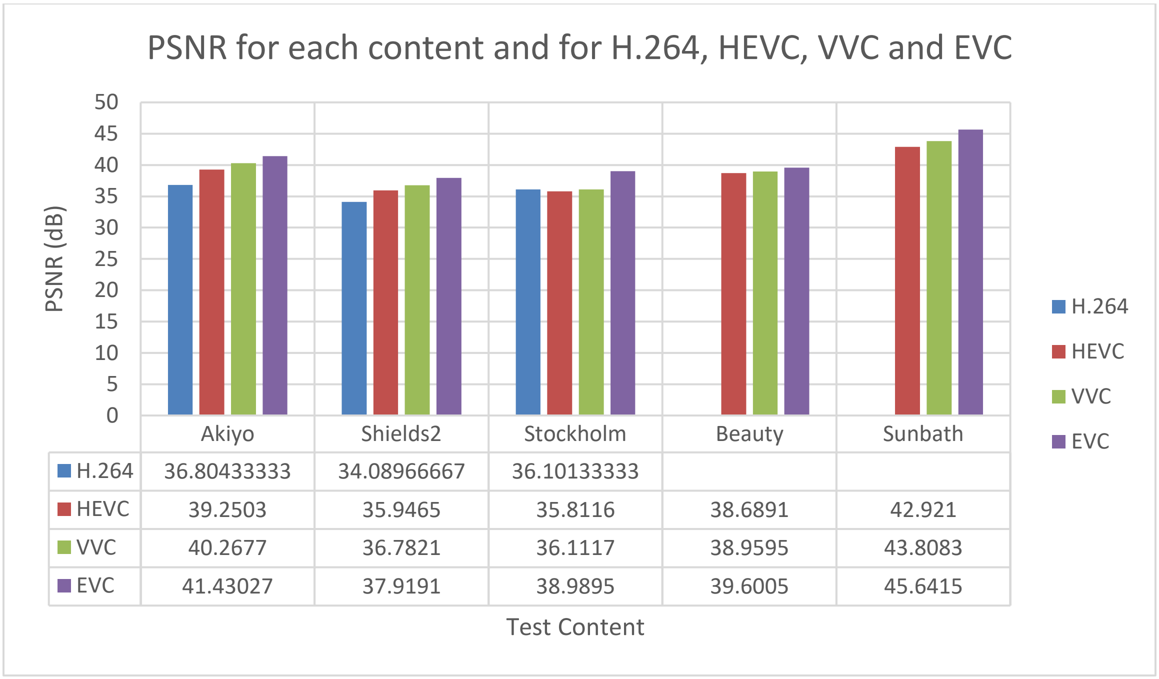
Sl. No.	INPUT VIDEO FILE	PSNR (dB)	SSIM YUV	BITRATE (kbps)	ENCODING TIME (s)	DECODING TIME (s)	COMPRESSION SIZE (kB)
1	Akiyo	41.43	0.995	21.262	154.576	1.518	39
2	Shields2	37.92	0.993	348.658	1114.427	1.581	256
3	Stockholm	38.99	0.983	709.307	17186.5	17.895	2615
4	Beauty	39.601	0.972	2079.912	40601.449	110.111	1270
5	SunBath	45.6415	0.999	11141.700	94910.641	98.6	3401

Table 7.6. EVC encoder and decoder statistics

7.4.5. Objective Quality Comparative analysis

The objective quality analysis for all the four codecs was tabulated from Tables 7.3., 7.4., 7.5. and 7.6. These graphs from Figure 7.3. to 7.9. gives us a better understanding on the quality of the encoded sequence for each codec (i.e. H.264, HEVC, VVC and EVC) using PSNR, SSIM, encoding time and compression size. Testing set was for contents ranging from qcif to 4K resolution from Table 7.1.

1. Peak Signal to Noise Ratio (PSNR) analysis:



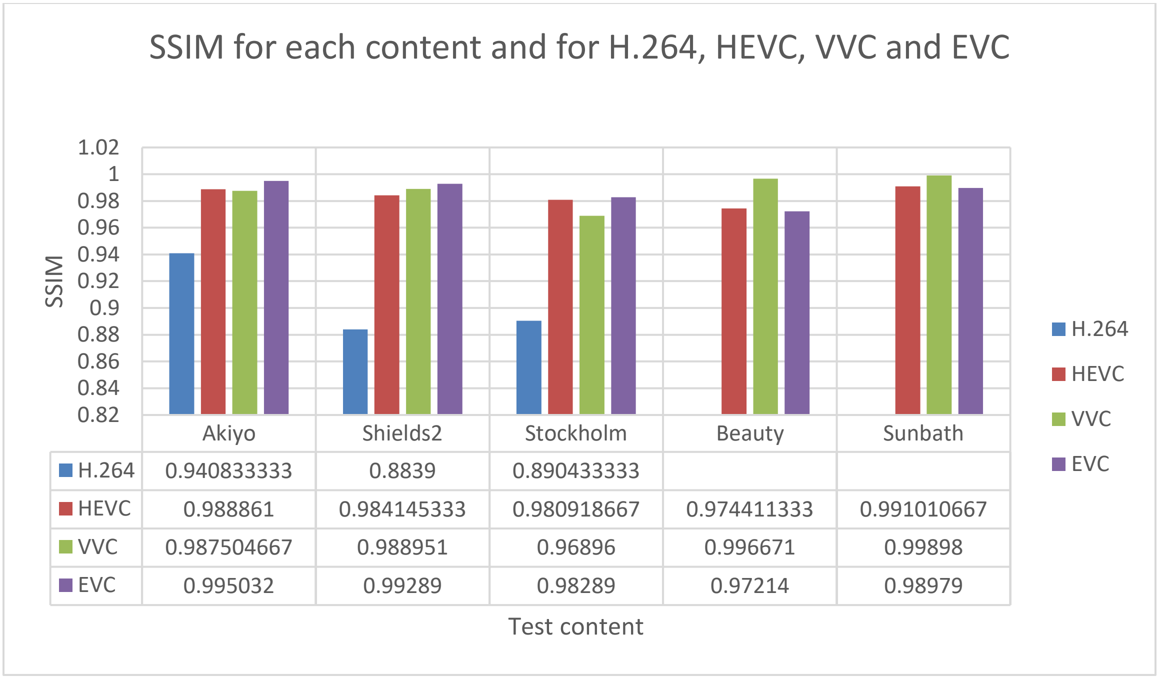
	Akiyo	Shields2	Stockholm	Beauty	Sunbath
🟦 H.264	36.80433333	34.08966667	36.10133333		
🟥 HEVC	39.2503	35.9465	35.8116	38.6891	42.921
🟩 VVC	40.2677	36.7821	36.1117	38.9595	43.8083
🟪 EVC	41.43027	37.9191	38.9895	39.6005	45.6415

Figure 7.3. PSNR value for each content and for H.264, HEVC, VVC and EVC

From Figure 7.3. we observe that the PSNR value is more for EVC followed by VVC, HEVC and H.264. Hence the reconstructed video quality was more in EVC in terms of PSNR value. The difference in PSNR between VVC and EVC is not much as most of the encoder blocks algorithms are the same between both, like motion estimation, entropy coding and block structure. The difference is due to the motion vector prediction which effects the quality of the video. More the prediction modes, more the video quality.

2. Structural Similarity Index Metric (SSIM) analysis:

From Figure 7.4. we see that SSIM value for EVC is more than the other codecs for contents from qcif to HD. For ultra HD and 4K, we see VVC preceding in the output quality when compared to other codecs. The difference in the output quality SSIM between VVC and EVC for Ultra HD and 4K resolution videos is due to higher profile being selected for VVC for higher resolution videos when compared to EVC which does not change the profile and remains constant. This higher profile gives the liberty to choose hierarchical motion estimation which is better when compared to EVC motion estimation process. Hence the output quality is better for VVC when compared to EVC. To check the output quality in objective method, SSIM is preferred compared to PSNR as it's a better metric calculation for video quality.

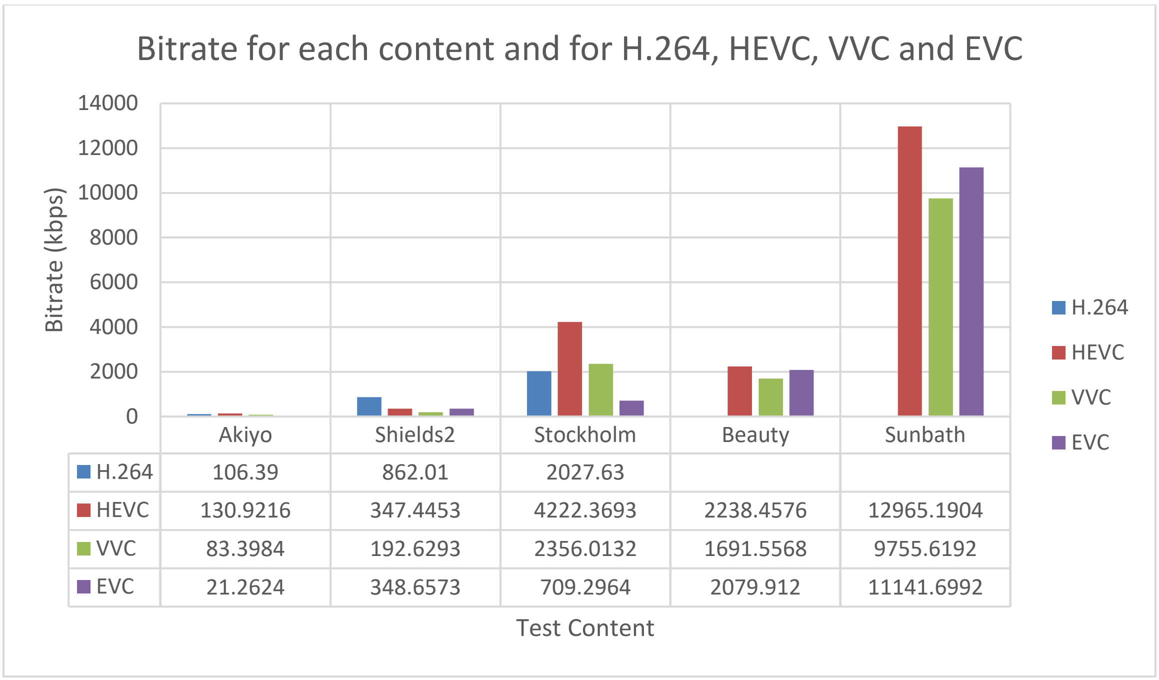


	Akiyo	Shields2	Stockholm	Beauty	Sunbath
🟦 H.264	0.940833333	0.8839	0.890433333		
🟥 HEVC	0.988861	0.984145333	0.980918667	0.974411333	0.991010667
🟩 VVC	0.987504667	0.988951	0.96896	0.996671	0.99898
🟪 EVC	0.995032	0.99289	0.98289	0.97214	0.98979

Figure 7.4. SSIM value for each content and for H.264, HEVC, VVC and EVC

3. Bitrate analysis:

Figure 7.5. shows the bitrate analysis for video codecs H.264, HEVC, VVC and EVC for contents from qcif to 4K resolution videos. It was observed from the graph that the bitrate required to encode a video for the same quality of output was more for HEVC and least for VVC. Hence proved that VVC provided more compression when compared to EVC and HEVC. This was coz of the intra prediction modes in VVC which is 81 predictions modes when compared to EVC and HEVC which is 30 and 35 modes. Hence it is expected that more compression can be achieved by using lesser bitrate in VVC when compared to EVC and HEVC for achieving the same output quality video.



	Akiyo	Shields2	Stockholm	Beauty	Sunbath
H.264	106.39	862.01	2027.63		
HEVC	130.9216	347.4453	4222.3693	2238.4576	12965.1904
VVC	83.3984	192.6293	2356.0132	1691.5568	9755.6192
EVC	21.2624	348.6573	709.2964	2079.912	11141.6992

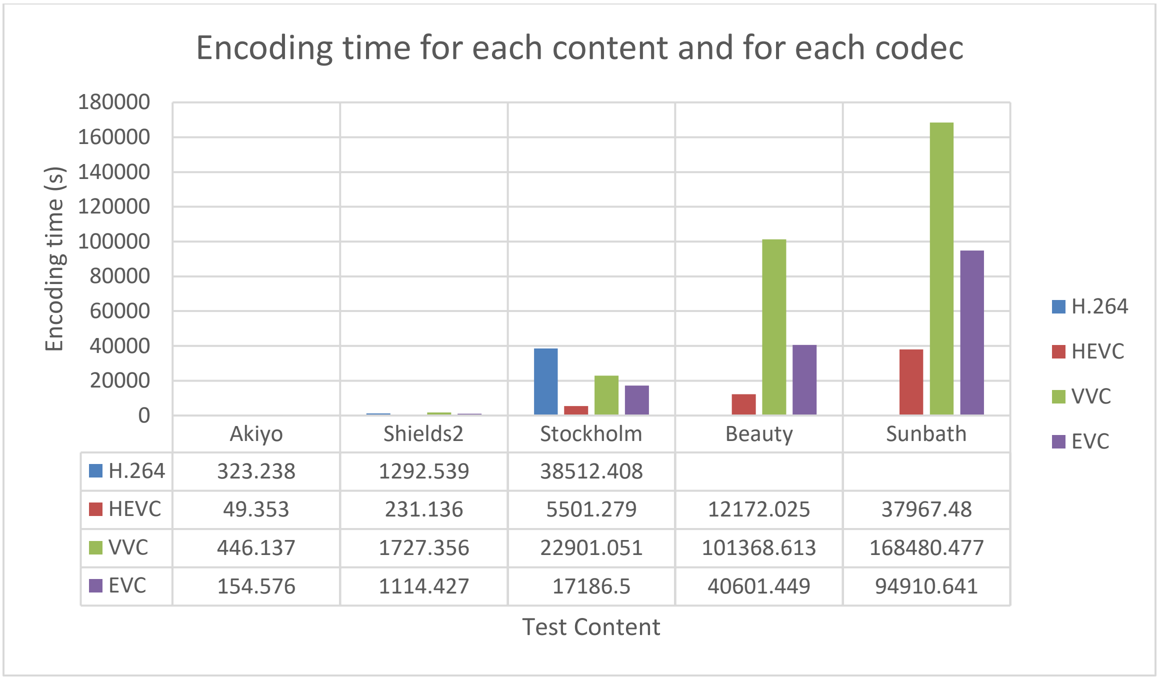
Figure 7.5. Bitrate for each content and for H.264, HEVC, VVC and EVC

4. Encoding Time analysis:

From Figure 7.6. it is evident that time taken to encode any video from qcif to 4k resolution, VVC takes maximum time (almost twice the time take by EVC). This is expected as the intra frame prediction modes are almost double the prediction modes for EVC codec (i.e. VVC prediction modes = 81 and EVC = 30). The block structure division by CTU is 256x256 for VVC when compared to 128x128 for EVC hence adding to doubling the time taken to scan the coefficients being twice in VVC when compared to EVC. The block structure division for HEVC is 64x64 which is half the structure when compared to EVC which is 128x128. Due to this, we observed that the time taken to encode the videos for same output quality for EVC is twice the time taken to encode HEVC.

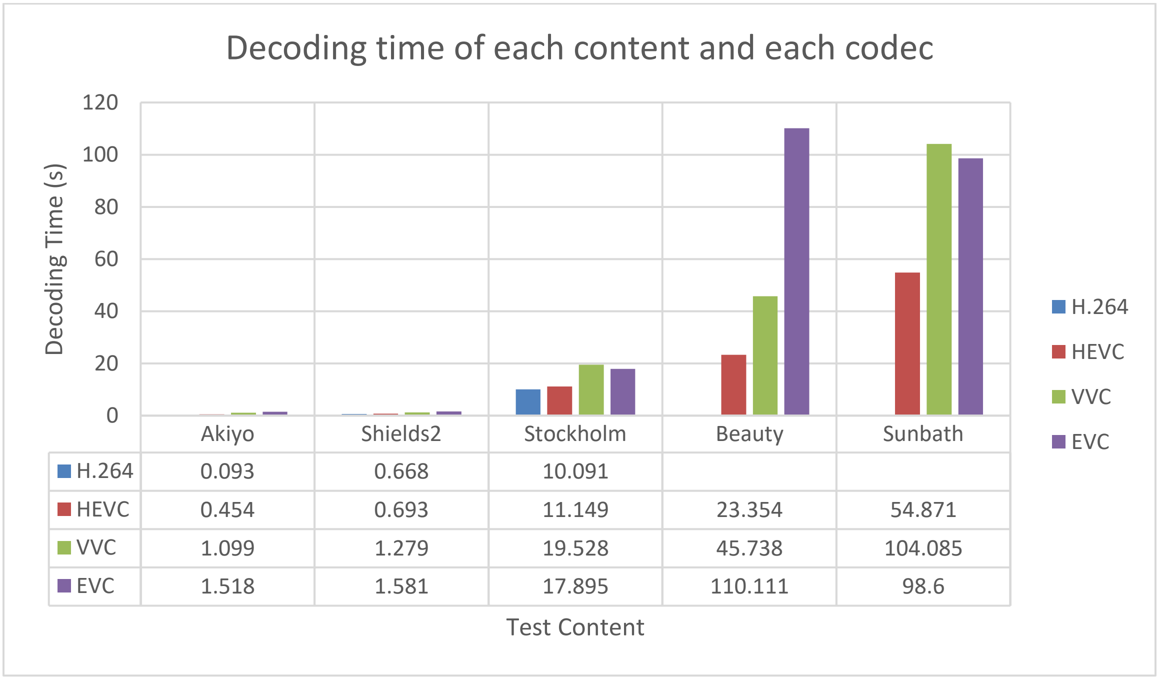
5. Decoding Time analysis:

From Figure 7.7. it is observed that the time taken to decode the encoded video for all the four codecs is almost similar with less difference between each of them. It doesn't matter how long it takes to decode as all the codecs decode within seconds from qcif to 4k resolution videos.



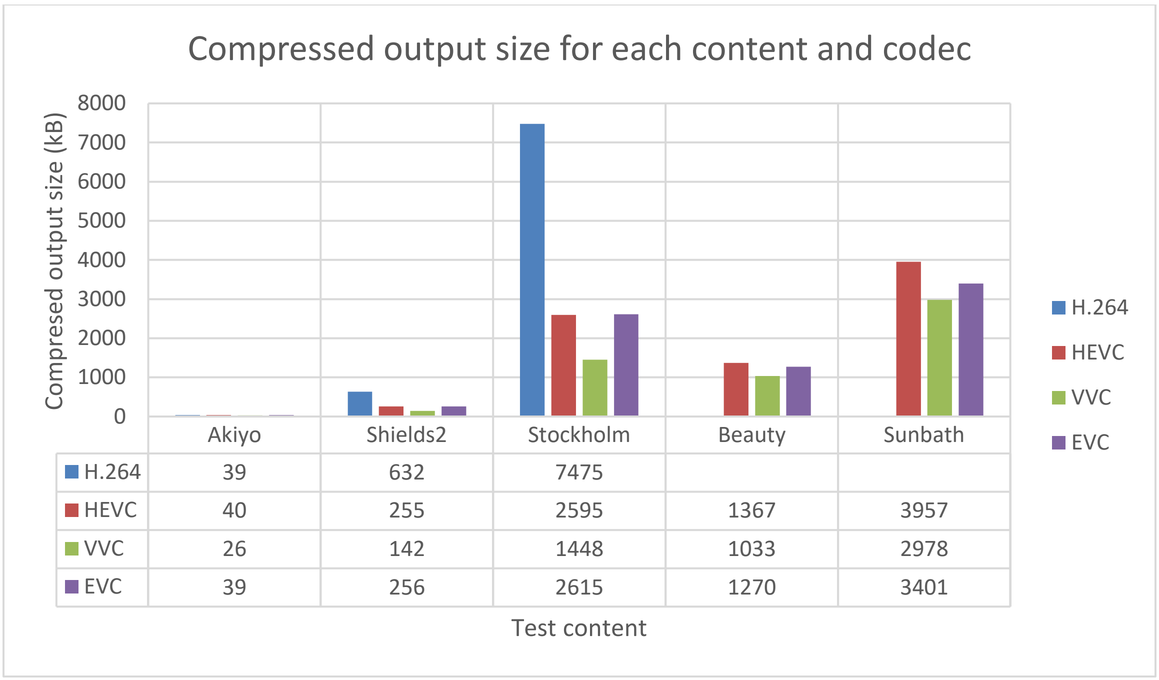
	Akiyo	Shields2	Stockholm	Beauty	Sunbath
🟦 H.264	323.238	1292.539	38512.408		
🟥 HEVC	49.353	231.136	5501.279	12172.025	37967.48
🟩 VVC	446.137	1727.356	22901.051	101368.613	168480.477
🟪 EVC	154.576	1114.427	17186.5	40601.449	94910.641

Figure 7.6. Encoding time for each content and for each codec



	Akiyo	Shields2	Stockholm	Beauty	Sunbath
🟦 H.264	0.093	0.668	10.091		
🟥 HEVC	0.454	0.693	11.149	23.354	54.871
🟩 VVC	1.099	1.279	19.528	45.738	104.085
🟪 EVC	1.518	1.581	17.895	110.111	98.6

Figure 7.7. Decoding time of each content and each codec



	Akiyo	Shields2	Stockholm	Beauty	Sunbath
🟦 H.264	39	632	7475		
🟥 HEVC	40	255	2595	1367	3957
🟩 VVC	26	142	1448	1033	2978
🟪 EVC	39	256	2615	1270	3401

Figure 7.8. Compressed output for each content and codec

6. Compression ratio analysis:

The bitrate analysis shown in Figure 7.5. does also portray the amount of compression achieved by each codec. More the bitrate, less the compressed encoded video and the same was observed in Figure 7.8. H.264 compressed size was maximum when compared to other codecs, hence it achieved least compression. VVC compression size was least when compared to other codecs, hence it achieved maximum compression. The reason for this maximum compression is similar to that of the bitrate reduction of more intra prediction modes in VVC when compared to other codecs.

7.4.6. Rate Distortion method comparative analysis

Rate distortion method is the graph obtained for PSNR to bitrate for a particular content tested at different QP values. In Table 7.7. the testing for content ‘shield2’ with resolution 640x360 was tested for codecs H.264, HEVC, VVC and EVC for QP values 10, 20, 30 and 40. The analysis of this graph is to achieve maximum output quality (PSNR) for lesser bitrate used. This also points at the codec that offers more compression with good quality encoded video.

Sl.No.	QP	RESOLUTION	No. Frames	PSNR (dB)				Bitrate (kbps)			
				H.264	HEVC	VVC	EVC	H.264	HEVC	VVC	EVC
1	10	640x360	120	51.40233	47.6618	48.3477	48	14951	6955.77	6427	6704
2	20	640x360	120	42.8583	41.9283	42.8424	43.9	3073.8	1224.51	1142.8	1204
3	30	640x360	120	36.91933	37.1581	38.5625	39.1	548.52	429.427	405.73	429
4	40	640x360	120	31.851333	35.3014	32.4476	33.6	144.47	205.173	133.15	130.6
5	50	640x360	120	28.80833	25.6372	26.442	29.5	41.39	30.6813	30.344	28.8

Table 7.7. Rate Distortion graph statistics for shield2 content

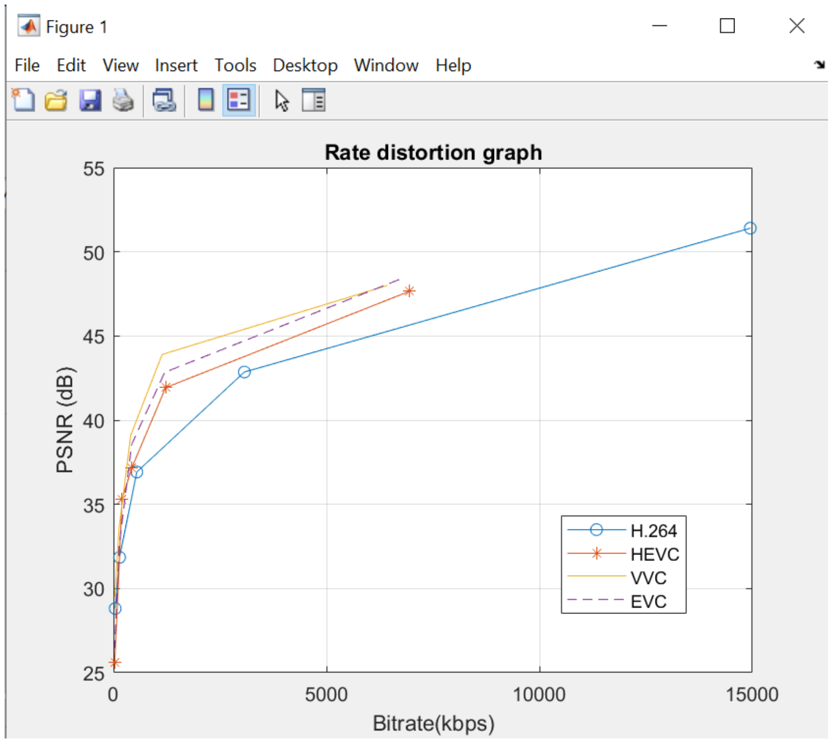


Figure 7.9. Rate distortion graph for shield2 content

Rate distortion graph for ‘shield2’ content is as shown in Figure 7.9. This graph shows that the quality of encoded video is more for VVC followed by EVC and then HEVC and H.264. This also depicts that the compression is achieved maximum for VVC since the bitrate required for VVC is less to obtain the same output quality video when compared to the other 3 codecs. The reason for this output is similar to the reason analysed for the objective quality analysis of the codecs obtained from Figure 7.3 to Figure 7.8.

7.4.7. Subjective Quality Comparative analysis:

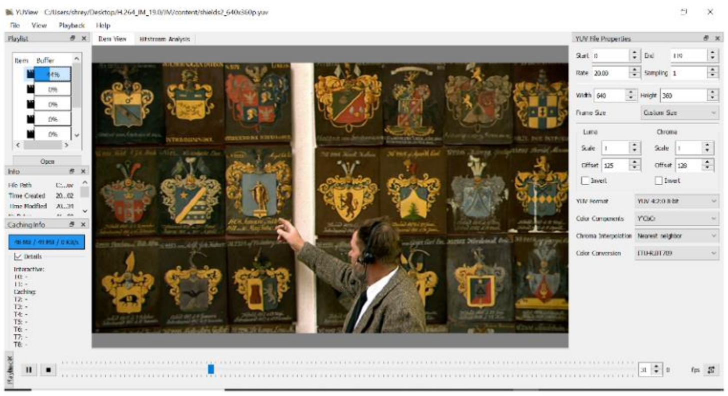
Subjective video quality is a video quality assessment as experienced by humans. It is concerned with how video is perceived by a viewer (also called “observer” or “subject”) and designates their opinion on a particular video sequence and therefore related to field of quality of experience. The measurement of subjective video quality is necessary as the objective quality assessment algorithms such as PSNR have been shown to correlate badly with ratings. Subjective ratings may also be used as ground truth to develop new algorithms.

Subjective video quality tests are psychophysical experiments in which a number of viewers rate a given set of videos. These tests are quite expensive in terms of time (preparation and running) and human resources and must therefore be carefully designed.

In our subjective quality test. It is verified by one viewer and the “sources”, i.e. original video sequences are run through encoder and decoder of their respective codec softwares having the same parameter changes across all the codecs to generate the processed video sequences.

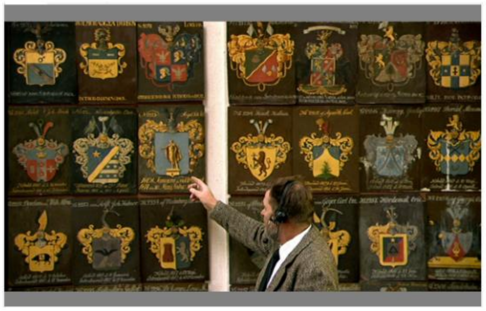
Figure 7.13 to 7.16 shows the subjective quality analysis using rate distortion statistics outputs obtained from Table 7.7. The subjective quality analysis is seen for shield2 content for QP’s 10, 20, 30, 40 and 50 for codec H.264, HEVC, VVC and EVC. It was observed that there was no difference observed in the output video quality for QP = 10 between the codecs but there was a drastic video quality degradation observed for the four codecs for QP=50. VVC proved to provide a better video quality which was viewed by the naked eye when compared other codecs. EVC followed VVC in video quality improvements. H.264 and HEVC showed more video degradation for QP = 50.

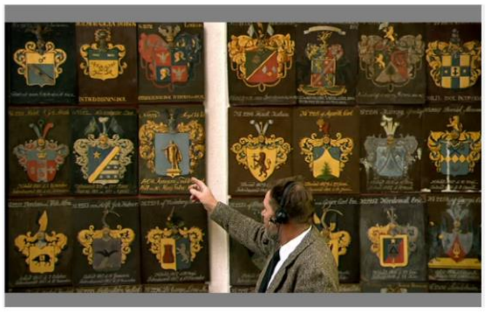
The objective quality and subjective quality analysis matched their results with VVC providing better quality output followed by EVC, HEVC and then H.264. It was also proved that VVC provided more compression when compared to EVC, followed by HEVC and then H.264 due to more intra prediction modes and block structure size in VVC.

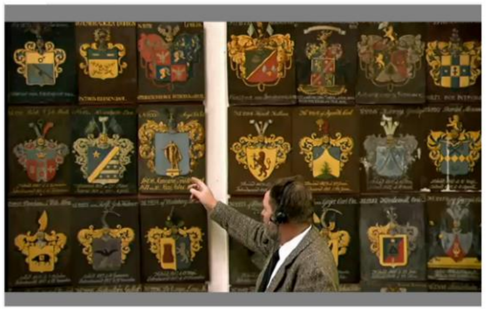

YUView C:/Users/shrey/Desktop/H.264_JM_19.0/JM/content/shields2_640x360p.yuv

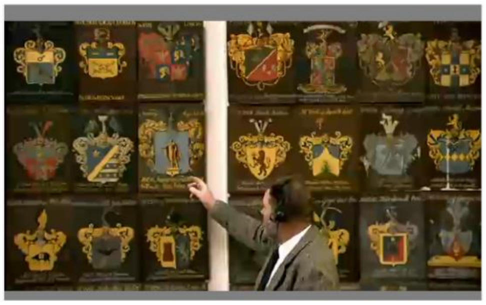
Start 0
End 119
Rate 20.00
Sampling 1
Width 640
Height 360
Frame Size Custom Size
Luma Scale 1 Offset 125 Invert
Chroma Scale 1 Offset 128 Invert
YUV Format YUV 4:2:0 8-bit
Color Components Y'CbCr
Chroma Interpolation Nearest neighbor
Color Conversion ITU-R.BT709

Original Shield2 content at frame number=31


Shield2 content for QP = 10 and frame number = 31


Shield2 content for QP = 20 and frame number = 31


Shield2 content for QP = 30 and frame number = 31


Shield2 content for QP = 40 and frame number = 31

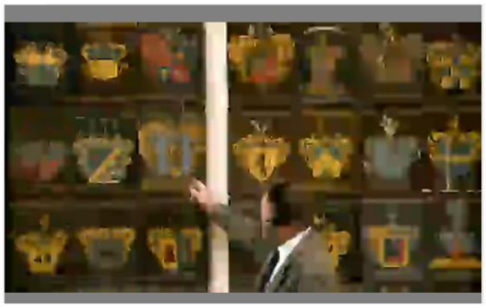
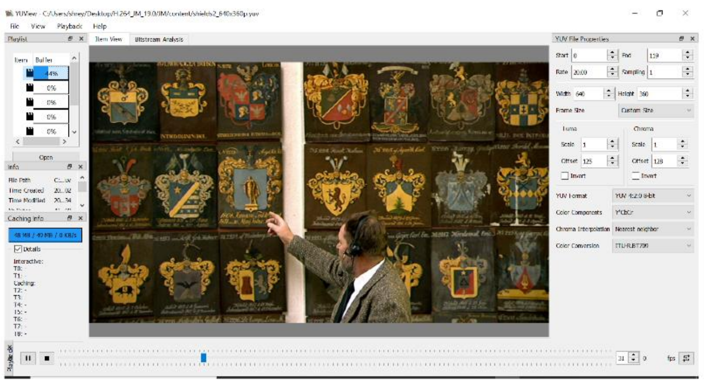

Shield2 content for QP = 50 and frame number = 31

Figure 7.10. Rate distortion graph statistics for shield2 content and QP = 10, 20, 30, 40 and 50 for H.264 codec


YUView - C:/Users/shrey/Desktop/H.264_JM_19.0/JM/content/shields2_640x360p.yuv

File View Playback Help

Playlist | Item View | Bitstream Analysis

Item | Buffer
44%
0%
0%
0%
0%

Open

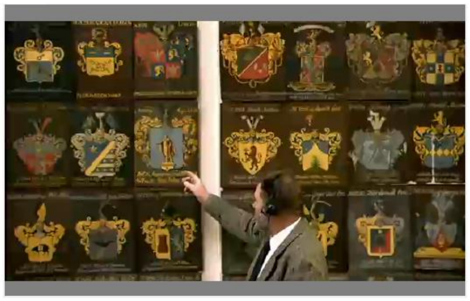
Info
File Path C:...uv
Time Created 20...02
Time Modified 20...34

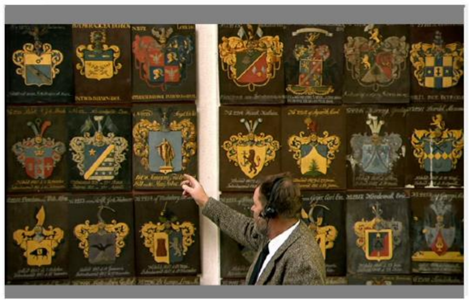
Caching Info
48 MB / 40 MB / 0 KB/s
Details
Interactive:
T0:
T1:
Caching:
T2: -
T3:
T4: -
T5: -
T6:
T7:
T8: -

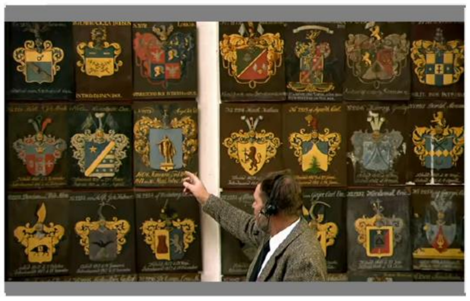
YUV File Properties
Start 0 End 119
Rate 20.00 Sampling 1
Width 640 Height 360
Frame Size Custom Size
Luma Scale 1 Offset 125 Invert
Chroma Scale 1 Offset 128 Invert
YUV Format YUV 4:2:0 8-bit
Color Components Y'CbCr
Chroma Interpolation Nearest neighbor
Color Conversion ITU-R.BT709

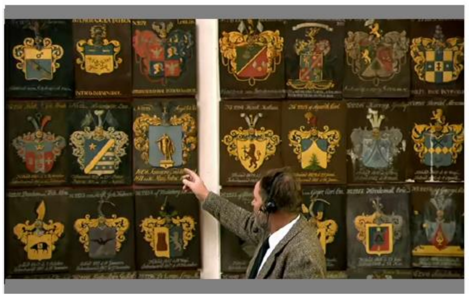
31 0 fps

Original Shield2 content at frame number=31


Shield2 content for QP = 10 and frame number = 31


Shield2 content for QP = 20 and frame number = 31


Shield2 content for QP = 30 and frame number = 31


Shield2 content for QP = 40 and frame number = 31

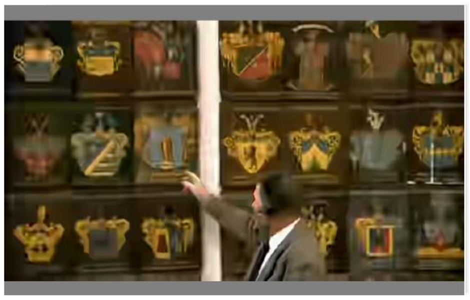
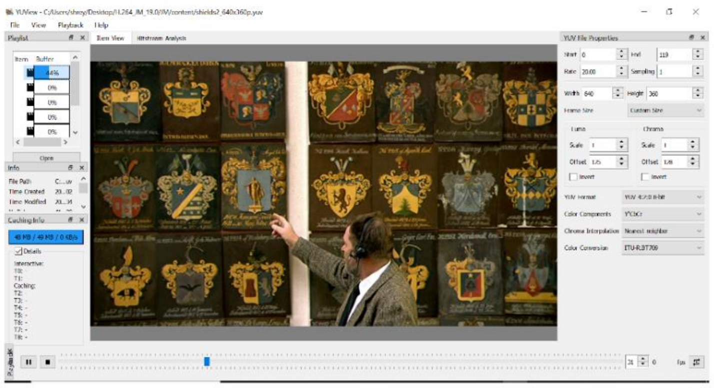

Shield2 content for QP = 50 and frame number = 31

Figure 7.11. Rate distortion graph statistics for shield2 content and QP = 10, 20, 30, 40 and 50 for HEVC codec



YUView - C:/Users/shrey/Desktop/H.264_JM_19.0/JM/content/shields2_640x360p.yuv

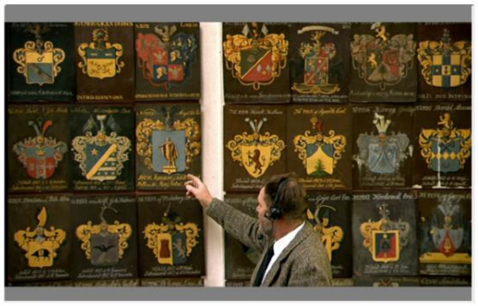
YUV File Properties

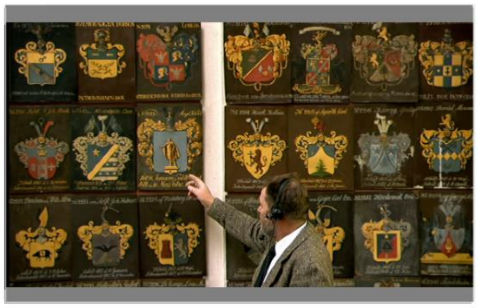
Start 0 End 119
Rate 20.00 Sampling 1
Width 640 Height 360
Frame Size Custom Size

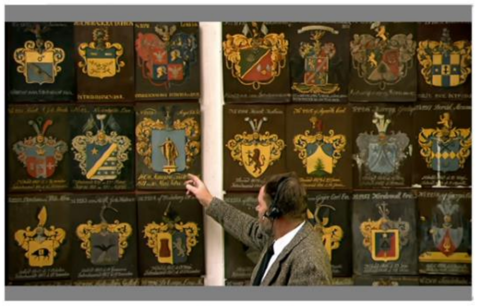
Luma: Scale 1, Offset 1/5, Invert
Chroma: Scale 1, Offset 1/8, Invert

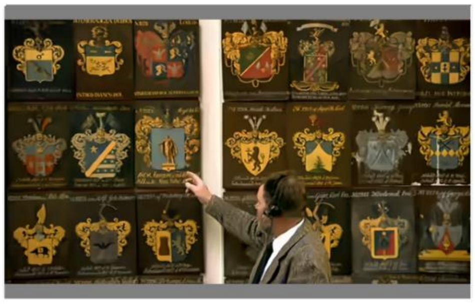
YUV Format YUV 4:2:0 8-bit
Color Components Y'CbCr
Chroma Interpolation Nearest neighbor
Color Conversion ITU-R.BT709

Original Shield2 content at frame number=31


Shield2 content for QP = 10 and frame number = 31


Shield2 content for QP = 20 and frame number = 31


Shield2 content for QP = 30 and frame number = 31


Shield2 content for QP = 40 and frame number = 31

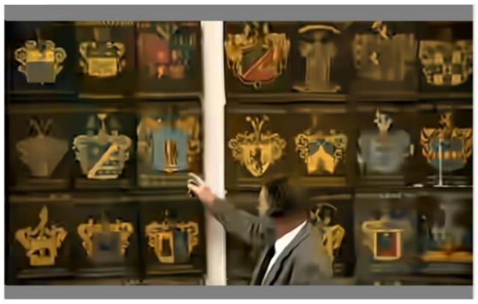
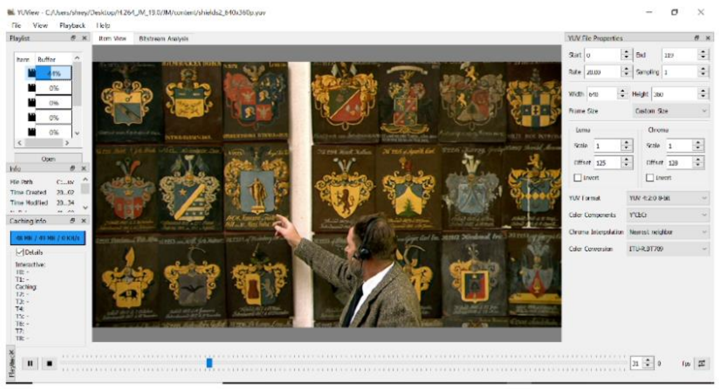
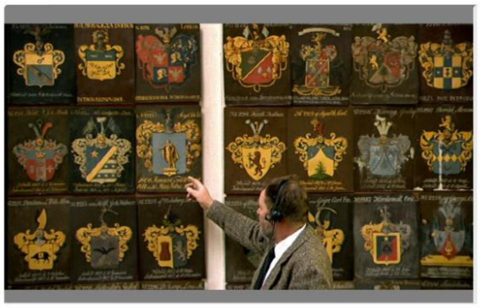

Shield2 content for QP = 50 and frame number = 31

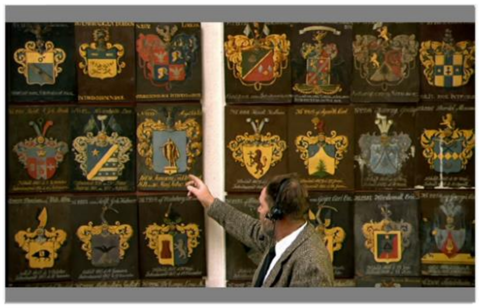
Figure 7.12. Rate distortion graph statistics for shield2 content and QP = 10, 20, 30, 40 and 50 for VVC codec

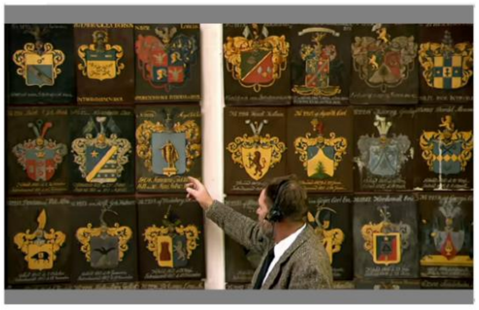


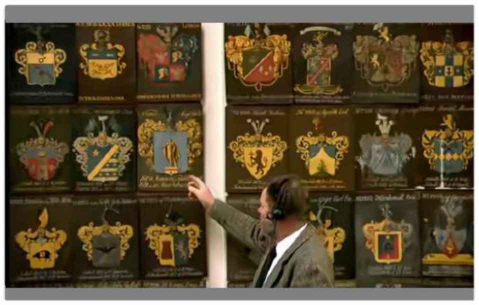
Start 0 End 119
Rate 20.00 Sampling 1
Width 640 Height 360
Frame Size Custom Size
Luma Scale 1 Offset 125 Invert
Chroma Scale 1 Offset 128 Invert
YUV Format YUV 4:2:0 8-bit
Color Components Y'CbCr
Chroma Interpolation Nearest neighbor
Color Conversion ITU-R.BT709

Original Shield2 content at frame number=31


Shield2 content for QP = 10 and frame number = 31


Shield2 content for QP = 20 and frame number = 31


Shield2 content for QP = 30 and frame number = 31


Shield2 content for QP = 40 and frame number = 31

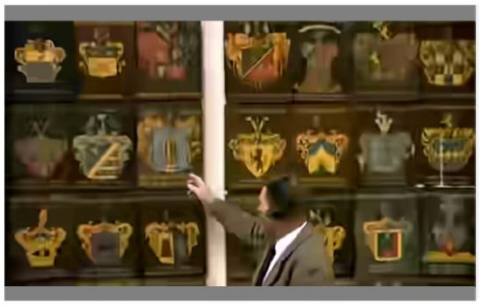

Shield2 content for QP = 50 and frame number = 31

Figure 7.13. Rate distortion graph statistics for shield2 content and QP = 10, 20, 30, 40 and 50 for EVC codec

7.5. Results of Objective 2: Heterogeneous Transcoding architecture for format change

Heterogeneous Transcoding of H.264, HEVC, VVC and EVC video codecs for format changes using cascaded decoder-encoder pixel transcoding architecture provides very good PSNR and SSIM output values for QP=10. Both PSNR and SSIM graphs in Figure 7.17. and Figure 7.18. proves that the transcoding from HEVC to VVC gives a better PSNR value which is almost equal to 50.5dB and SSIM value which is almost equal to 0.999863. If we increase the QP from 10 to 20, 30 or 40, we see a degradation in the quality of output video.

7.5.1. Objective Quality analysis for Heterogeneous Transcoding

Figure 6.1, 6.2, 6.3 and 6.4 shows the architectural format change block diagram for H.264, HEVC, VVC and EVC. Metrics like PSNR, SSIM and Time taken to encode was tabulated for each format conversion as shown in Table 7.8 for akiyo_qcif content.

1. Peak Signal to Noise Ratio (PSNR) analysis

Figure 7.17. shows the PSNR graph with respect to Heterogeneous transcoding for format change using akiyo_qcif content. All the testing scenarios proved to provide very good output quality with PSNR value being very high for QP =10. In Objective 1, it was proved that VVC and EVC codecs provided better video quality output due to improvised algorithm in encoder block when compared to HEVC and H.264. Hence the codec change from HEVC to VVC and H264 to EVC showed best PSNR value which is almost equal to 50 dB and EVC to H264 least value, almost equal to 49dB.

2. Structural Similarity Index Metric (SSIM) analysis

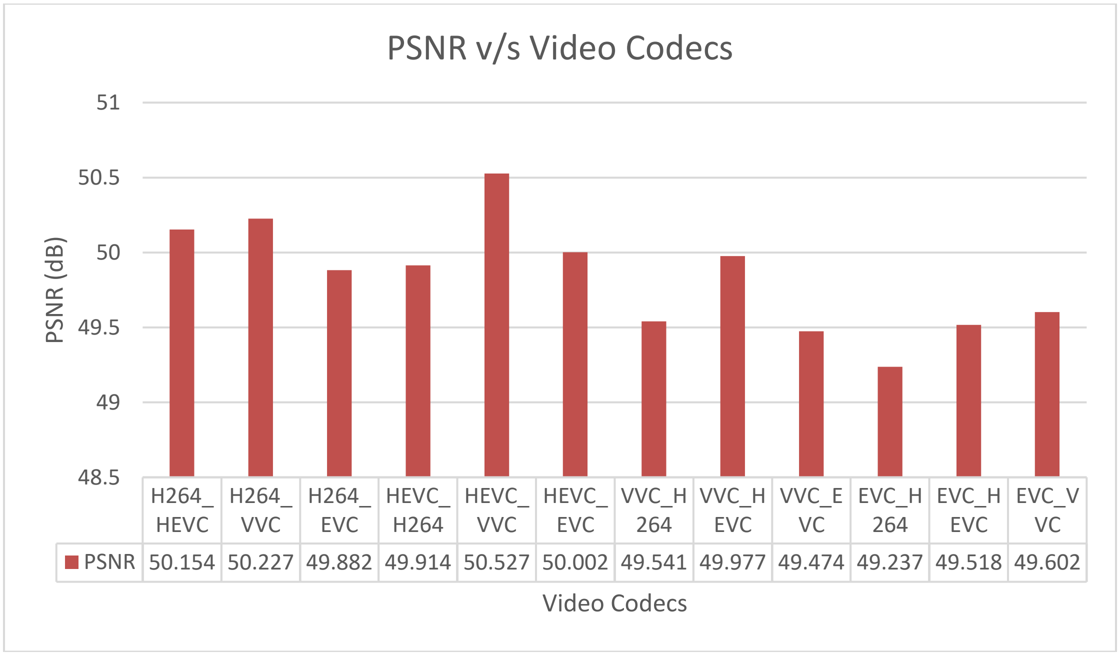
Figure 7.18. shows the SSIM graph with respect to Heterogeneous transcoding for format change using akiyo_qcif content. All the testing scenarios proved to provide very good output quality with SSIM value being very high with a value of 0.999 for QP=10. The output result was similar to PSNR analysis as HEVC to VVC showed the highest SSIM value when compared to other format changes.

3. Time taken to transcode analysis

Figure 7.19. shows the time taken to transcode from one codec to another in heterogeneous form. It is obvious that any codec conversion involving VVC takes a lot of time as observed in Objective 1, as the time taken to encode VVC is double the time taken to encode EVC. Due to this analysis, we see maximum time taken to transcode for QP=10 is VVC to EVC and EVC to VVC.

Video Transcoding	PSNR (dB)	SSIM	Time Taken(s)
H264_HEVC	50.15354	0.999624	411.977
H264_VVC	50.22658	0.999639	2109.25
H264_EVC	49.88168	0.999608	1102.07
HEVC_H264	49.91432	0.999613	401.235
HEVC_VVC	50.52744	0.999681	1614.09
HEVC_EVC	50.00184	0.999628	759.391
VVC_H264	49.54123	0.999595	2110.5
VVC_HEVC	49.97654	0.999644	2105.07
VVC_EVC	49.47421	0.999612	2449.75
EVC_H264	49.23676	0.999571	1099.86
EVC_HEVC	49.51782	0.999604	857.551
EVC_VVC	49.60185	0.999621	2377.56

Table 7.8. Video transcoding for akiyo_qcif content



	H264_HEVC	H264_VVC	H264_EVC	HEVC_H264	HEVC_VVC	HEVC_EVC	VVC_H264	VVC_HEVC	VVC_EVC	EVC_H264	EVC_HEVC	EVC_VVC
🟥 PSNR	50.154	50.227	49.882	49.914	50.527	50.002	49.541	49.977	49.474	49.237	49.518	49.602

Figure 7.14. PSNR vs Video Codecs for Heterogeneous transcoding

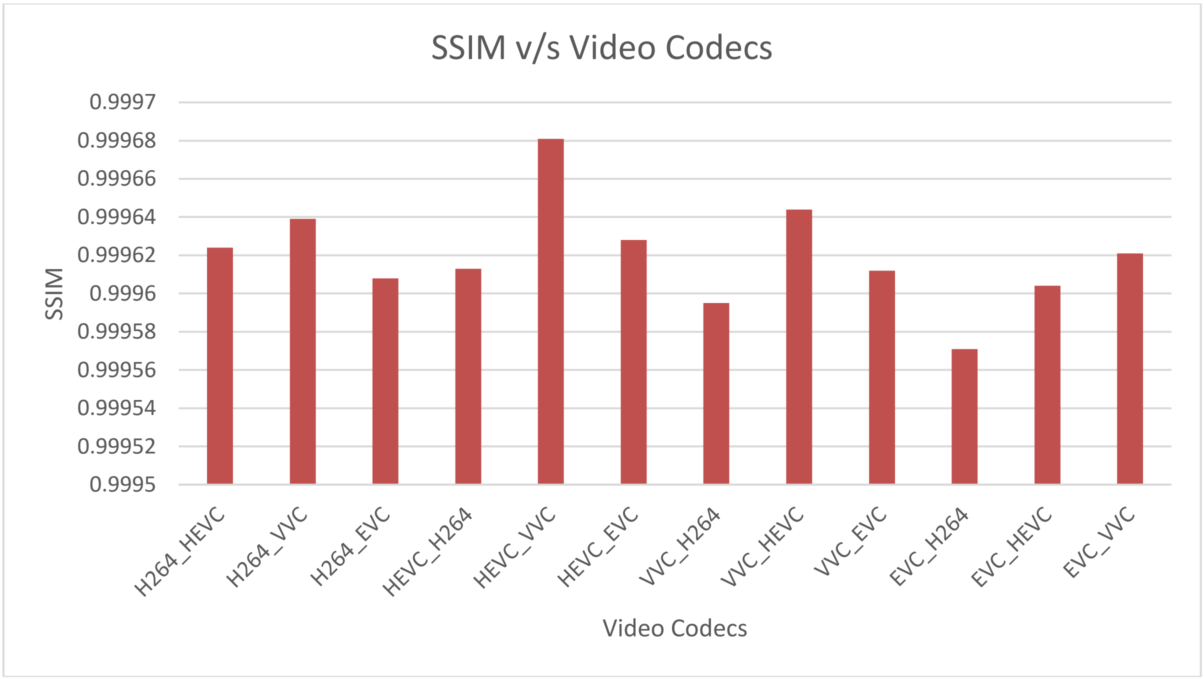
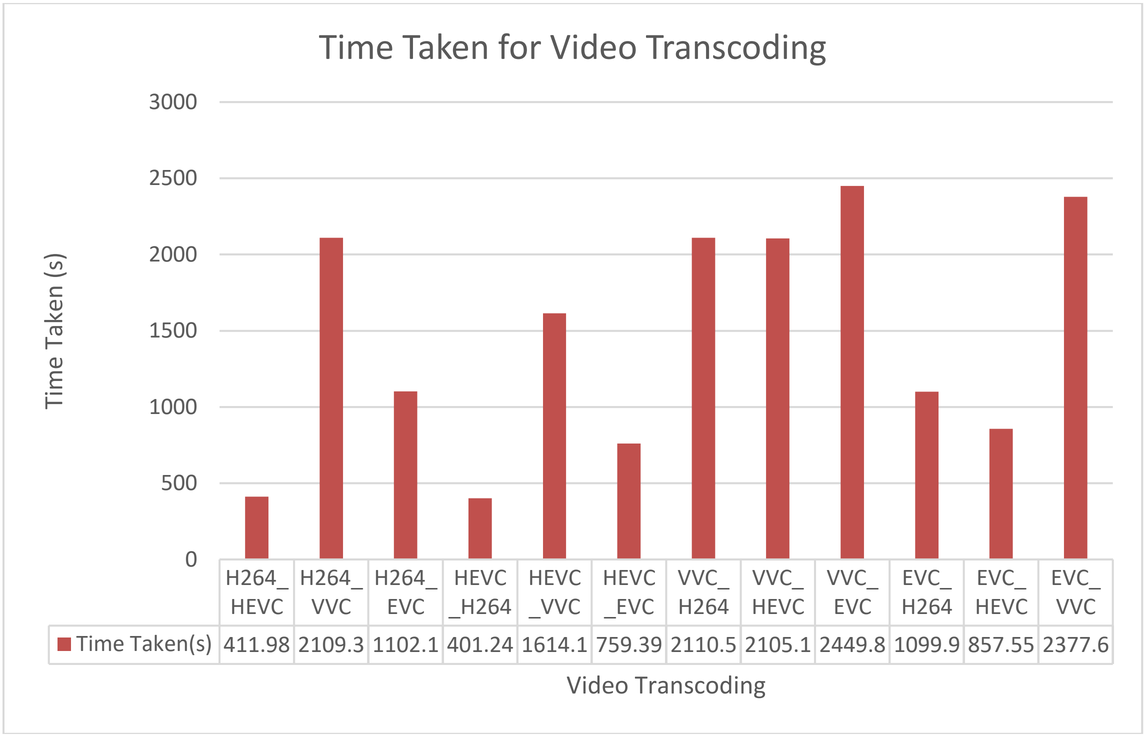


Figure 7.15. SSIM vs Video Codecs for Heterogeneous transcoding

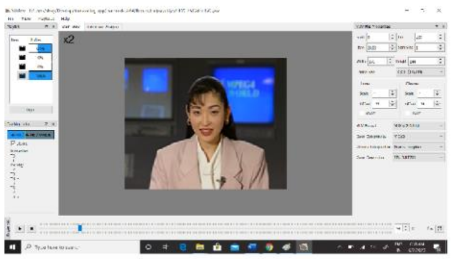


	H264_ HEVC	H264_ VVC	H264_ EVC	HEVC _H264	HEVC _VVC	HEVC _EVC	VVC_ H264	VVC_ HEVC	VVC_ EVC	EVC_ H264	EVC_ HEVC	EVC_ VVC
🟥 Time Taken(s)	411.98	2109.3	1102.1	401.24	1614.1	759.39	2110.5	2105.1	2449.8	1099.9	857.55	2377.6

Figure 7.16. Time taken for Heterogeneous transcoding

7.5.2. Subjective Quality analysis for Heterogeneous Transcoding

Subjective quality analysis for the Heterogeneous Transcoding for codecs H.264, HEVC, VVC and EVC also proves that the quality for video is the same for QP = 10 as observed in Figure 7.17 and 7.18.


Akiyo_qcif original content (176x144) for 300 frames

Output of H.264 to HEVC transcoding (dec_HEVC.yuv)
QP = 10, PSNR = 50.15dB, SSIM = 0.9996

Output of H.264 to VVC transcoding (dec_VVC.yuv)
QP = 10, PSNR = 50.22dB, SSIM = 0.9996

Output of H.264 to EVC transcoding (dec_EVC.yuv)
QP = 10, PSNR = 49.881dB, SSIM = 0.9996

Output of HEVC to H.264 transcoding (test_dec.yuv)
QP = 10, PSNR = 49.91dB, SSIM = 0.9996

Output of HEVC to VVC transcoding (dec_VVC.yuv)
QP = 10, PSNR = 50.52dB, SSIM = 0.9996

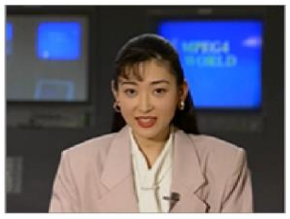
Output of HEVC to EVC transcoding (dec_EVC.yuv)
QP = 10, PSNR = 50.00 dB, SSIM = 0.9996

Output of VVC to H.264 transcoding (test_dec.yuv)
QP = 10, PSNR = 49.54 dB, SSIM = 0.9995

Output of VVC to HEVC transcoding (dec_HEVC.yuv)
QP = 10, PSNR = 49.97 dB, SSIM = 0.9996

Output of VVC to EVC transcoding (dec_EVC.yuv)
QP = 10, PSNR = 49.47 dB, SSIM = 0.9996

Output of EVC to H.264 transcoding (test_dec.yuv)
QP = 10, PSNR = 49.23 dB, SSIM = 0.9995)


Output of EVC to HEVC transcoding (dec_HEVC.yuv)
QP = 10, PSNR = 49.51 dB, SSIM = 0.9996)

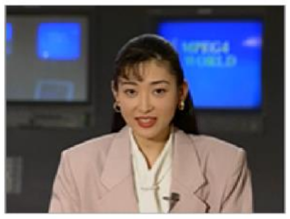

Output of EVC to VVC transcoding (dec_VVC.yuv)
QP = 10, PSNR = 49.60 dB, SSIM = 0.9996)

Figure 7.17. Subjective quality analysis for Heterogeneous Transcoding

7.6. Results of Objective 3: Heterogeneous and Homogeneous Transcoding architecture for bitrate change by varying quantization parameter and frame rate

Heterogeneous / Homogeneous Transcoding of H.264, HEVC, VVC and EVC video codecs for bitrate changes using cascaded decoder-encoder pixel transcoding architecture provides very good PSNR and SSIM output values for QP=10. Both PSNR and SSIM graphs in Figure 3.23 and Figure 3.24 proves that the transcoding from HEVC to VVC gives a better PSNR value which is almost equal to 50.5dB and SSIM value which is almost equal to 0.999863.

7.6.1. Objective Quality analysis for Heterogeneous Bitrate Transcoding – QP variation

From Figure 6.5 and Figure 6.6, for a video sequence akiyo_qcif.yuv (raw video), quantization parameter is kept constant (QP1= 10) at the first codec level. The bitrate is noted as Bitrate1 at first encoded stage. Heterogeneous transcoding at second stage for different QP values (QP2 = 10, 15, 20, 25, 30, 35, 40, 45 and 50) are tested and corresponding Bitrate2 values are noted. I was observed that for a constant Bitrate1 at stage 1, the Bitrate2 reduced with increase in QP value and Bitrate2 increased with decrease in QP value at stage 2. Values of Encoding time, Bitrate2, PSNR, MSSIM and Transcoding time are tabulated for different heterogeneous transcoding combinations which can be seen from Table 7.9 to 7.24. The graph for Bitrate variation with respect to QP2 and Time taken for transcoding is observed from Figure 7.21. to 7.28.

It was also observed from the graphs (Figure 7.22, 7.24, 7.26. and 7.28.) that the Time taken for bitrate transcoding was more for codecs pertaining to VVC and EVC and less for H.264 and HEVC. This is due to the fact that for Objective1 implementation, it was proved that VVC and EVC takes more time to encode when compared to other 2 codecs. Transcoding time was directly proportional to the Bitrate2 and inversely proportional to QP2 value. It was also observed from the graphs (Figure 7.23, 7.25, 7.27 and 7.29) that the Bitrate2 at second stage was also observed to be inversely proportional to QP2.

	H.264			HEVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	318.516	2053.05	10	98.675	1095.216	52.5558	0.9995	419.55
2				15	84	601.776	49.5037	0.9991	431.73
3				20	67.516	365.5744	46.7981	0.9982	389.96
4				25	62.47	234.976	43.897	0.9964	395.06
5				30	57.491	154.14	40.6667	0.9925	385.35
6				35	53.883	102.49	37.3533	0.9859	376
7				40	53.735	68.53	34.052	0.9751	380.722
8				45	52.86	48.1952	31.0577	0.9596	413.656
9				50	53.29	35.3248	28.2751	0.9247	381.218

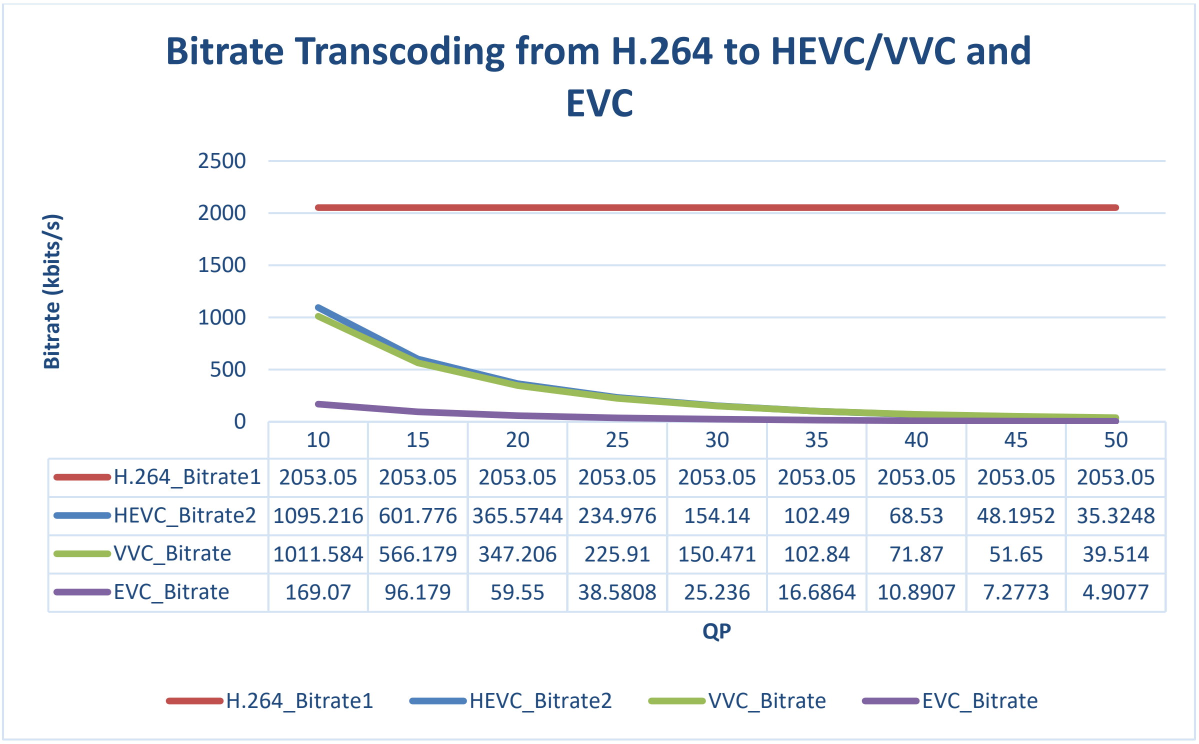
Table 7.9. H.264 to HEVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

	H.264			VVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSE (YUV)	Transcoding Time (s)
1	10	318.516	2053.05	10	1871.004	1011.584	53.0226	5.1873	2196.73
2				15	1388.58	566.179	50.359	9.579	1711.84
3				20	998.31	347.206	47.763	17.415	1335.98
4				25	740.033	225.91	45.0186	32.76	1079.62
5				30	549.995	150.471	41.961	66.24	865.251
6				35	450.73	102.84	38.807	136.92	770.861
7				40	368.221	71.87	35.669	282.045	697.679
8				45	302.965	51.65	32.374	602.26	625.385
9				50	304.515	39.514	29.35	1208.33	632.617

Table 7.10. H.264 to VVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

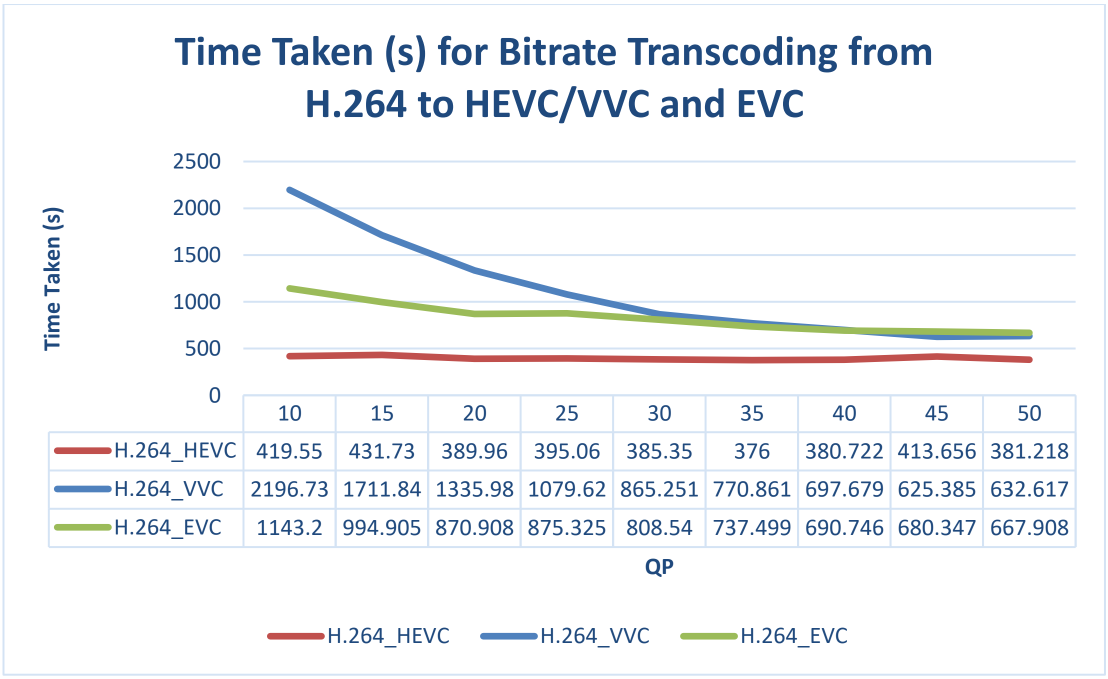
	H.264			EVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	318.516	2053.05	10	819.88	169.07	52.95	0.99941	1143.2
2				15	657.548	96.179	50.685	0.99914	994.905
3				20	514.925	59.55	48.3051	0.99877	870.908
4				25	538.785	38.5808	45.722	0.99805	875.325
5				30	450.015	25.236	42.8017	0.99637	808.54
6				35	321.856	16.6864	40.107	0.99224	737.499
7				40	267.882	10.8907	37.1163	0.98282	690.746
8				45	238.039	7.2773	34.1269	0.96044	680.347
9				50	157.036	4.9077	31.451	0.9161	667.908

Table 7.11. H.264 to EVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence



QP	10	15	20	25	30	35	40	45	50
H.264_Bitrate1	2053.05	2053.05	2053.05	2053.05	2053.05	2053.05	2053.05	2053.05	2053.05
HEVC_Bitrate2	1095.216	601.776	365.5744	234.976	154.14	102.49	68.53	48.1952	35.3248
VVC_Bitrate	1011.584	566.179	347.206	225.91	150.471	102.84	71.87	51.65	39.514
EVC_Bitrate	169.07	96.179	59.55	38.5808	25.236	16.6864	10.8907	7.2773	4.9077

Figure 7.18. Heterogeneous Bitrate Transcoding from H.264 to HEVC/VVC and EVC using akiyo_qcif sequence



QP	10	15	20	25	30	35	40	45	50
H.264_HEVC	419.55	431.73	389.96	395.06	385.35	376	380.722	413.656	381.218
H.264_VVC	2196.73	1711.84	1335.98	1079.62	865.251	770.861	697.679	625.385	632.617
H.264_EVC	1143.2	994.905	870.908	875.325	808.54	737.499	690.746	680.347	667.908

Figure 7.19. Time taken for Heterogeneous Bitrate Transcoding from H.264 to HEVC/VVC and EVC using akiyo_qcif sequence

	HEVC			H.264					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	88.562	1051.2256	10	341.65	348.18	53.271	0.9992	432.437
2				15	353.458	195.91	49.94	0.9988	446.621
3				20	339.424	109.25	46.66	0.9982	432.806
4				25	339.299	59.77	43.381	0.9968	430.8
5				30	340.024	31.47	40.066	0.9936	430.196
6				35	330.914	18.36	37.252	0.9883	416.826
7				40	357.275	11.45	34.802	0.9772	444.402
8				45	388.229	7.29	33.037	0.9652	475.437
9				50	386.13	5.17	31.576	0.9398	470.889

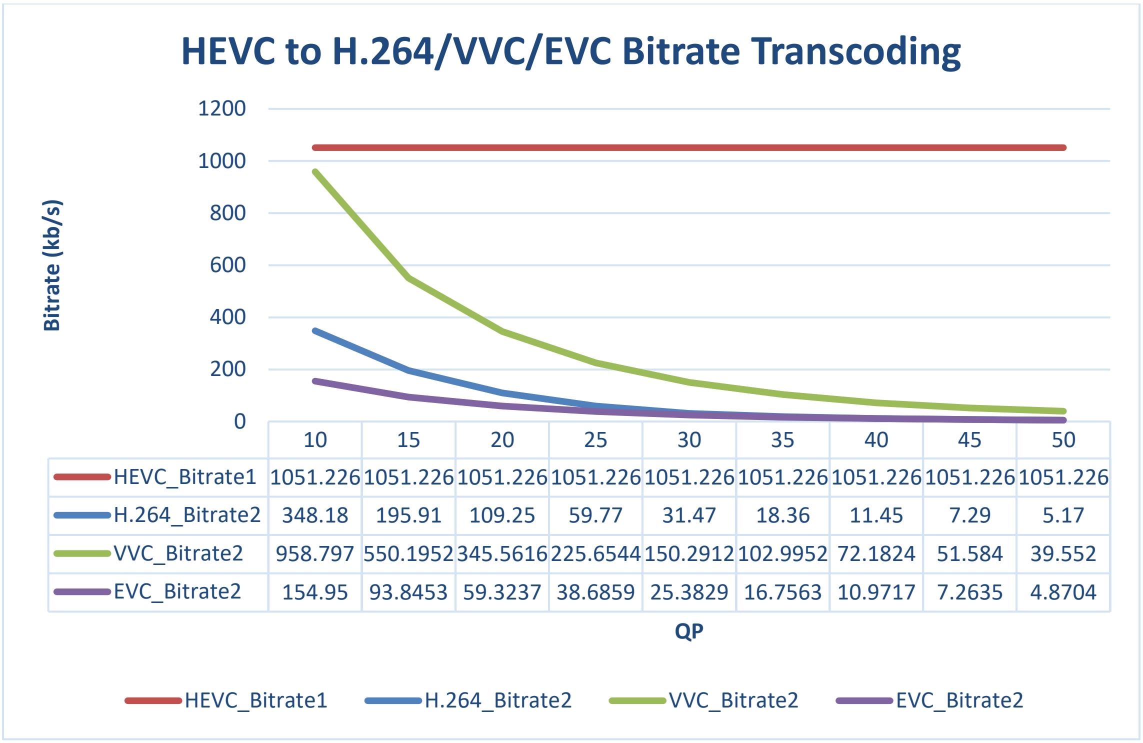
Table 7.12. HEVC to H.264 Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

	HEVC			VVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSE (YUV)	Transcoding Time (s)
1	10	88.562	1051.2256	10	1545.137	958.797	55.2698	3.0919	1636.41
2				15	2586.943	550.1952	51.2629	7.7788	1489.59
3				20	1625.48	345.5616	48.2874	15.4335	1075.39
4				25	731.523	225.6544	45.3038	30.6776	822.918
5				30	567.03	150.2912	42.0815	64.4249	657.081
6				35	460.478	102.9952	38.8749	134.8068	552.959
7				40	458.745	72.1824	35.6452	283.5856	458.745
8				45	320.137	51.584	32.3876	600.3978	408.594
9				50	306.003	39.552	29.3599	1205.6127	398.895

Table 7.13. HEVC to VVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

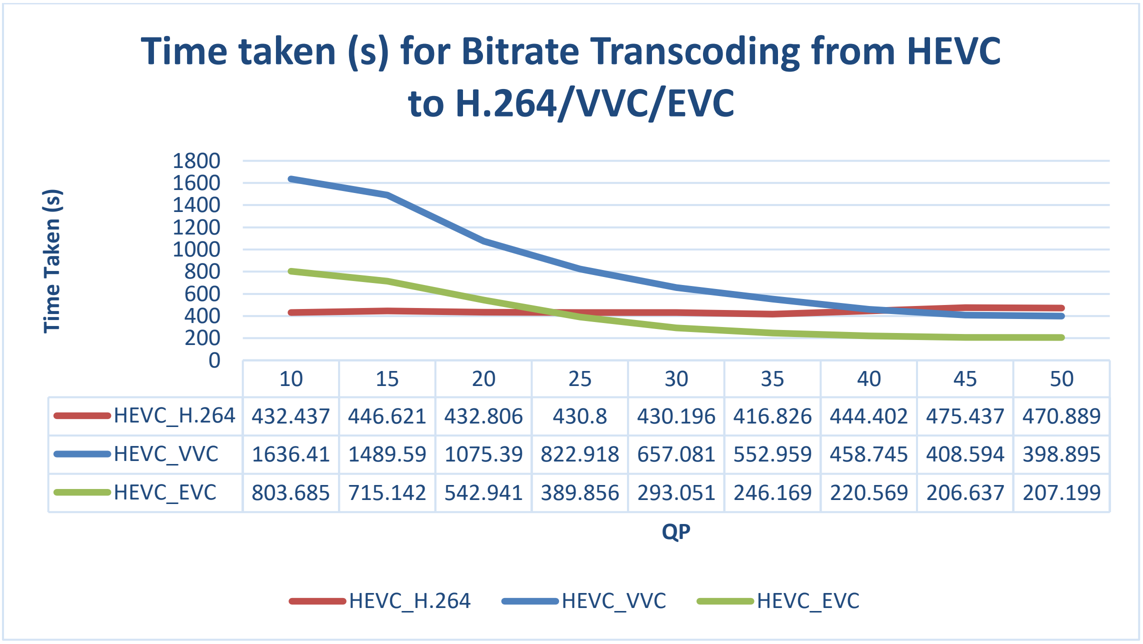
	HEVC			EVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	88.562	1051.2256	10	710.448	154.95	53.82	0.999479	803.685
2				15	615.384	93.8453	51.1042	0.9992196	715.142
3				20	445.314	59.3237	48.672	0.9988611	542.941
4				25	294.824	38.6859	45.9395	0.9981421	389.856
5				30	202.483	25.3829	42.8806	0.9965052	293.051
6				35	152.446	16.7563	40.1382	0.9924018	246.169
7				40	126.703	10.9717	37.0582	0.983137	220.569
8				45	111.609	7.2635	34.1254	0.9608622	206.637
9				50	114.873	4.8704	31.4754	0.9156842	207.199

Table 7.14. HEVC to EVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
HEVC_Bitrate1	1051.226	1051.226	1051.226	1051.226	1051.226	1051.226	1051.226	1051.226	1051.226
H.264_Bitrate2	348.18	195.91	109.25	59.77	31.47	18.36	11.45	7.29	5.17
VVC_Bitrate2	958.797	550.1952	345.5616	225.6544	150.2912	102.9952	72.1824	51.584	39.552
EVC_Bitrate2	154.95	93.8453	59.3237	38.6859	25.3829	16.7563	10.9717	7.2635	4.8704

Figure 7.20. Heterogeneous Bitrate Transcoding from HEVC to H.264/VVC and EVC using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
HEVC_H.264	432.437	446.621	432.806	430.8	430.196	416.826	444.402	475.437	470.889
HEVC_VVC	1636.41	1489.59	1075.39	822.918	657.081	552.959	458.745	408.594	398.895
HEVC_EVC	803.685	715.142	542.941	389.856	293.051	246.169	220.569	206.637	207.199

Figure 7.21. Time taken for Heterogeneous Bitrate Transcoding from HEVC to H.264/VVC and EVC using akiyo_qcif sequence

	VVC			H.264					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	1936.459	939.0048	10	387.597	362.25	52.725	0.9992	2326.28
2				15	349.96	200.45	49.845	0.9988	2192.77
3				20	328.039	111.22	46.663	0.9982	2052.32
4				25	341.26	60.61	43.365	0.9969	2054.69
5				30	318.798	31.47	40.065	0.9936	2025.83
6				35	330.696	18.45	37.313	0.98847	1988.27
7				40	350.143	11.46	34.781	0.9772	2031.91
8				45	407.88	7.28	33.072	0.9653	2308.83
9				50	382.229	5.15	43.584	0.9399	2074.24

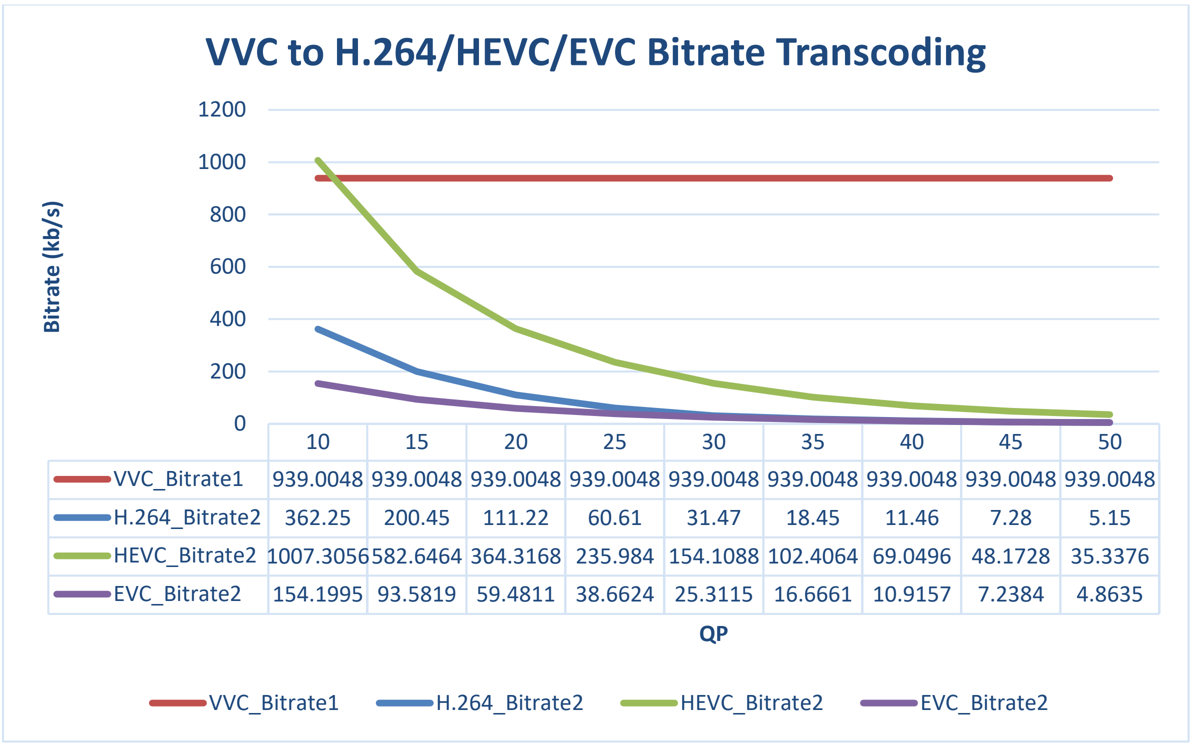
Table 7.15. VVC to H.264 Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

	VVC			HEVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	1936.459	939.0048	10	89.423	1007.3056	53.489	0.999613	1789.17
2				15	73.637	582.6464	50.0427	0.9992043	1772.79
3				20	65.271	364.3168	47.1767	0.998388	1752.46
4				25	59.795	235.984	44.1106	0.996669	1762.66
5				30	53.313	154.1088	40.7168	0.992673	1740.58
6				35	53.022	102.4064	37.3881	0.98617	1730.01
7				40	52.173	69.0496	34.1142	0.975577	1744.83
8				45	51.521	48.1728	31.0671	0.959009	1752.32
9				50	55.023	35.3376	28.2841	0.924517	1795.05

Table 7.16. VVC to HEVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

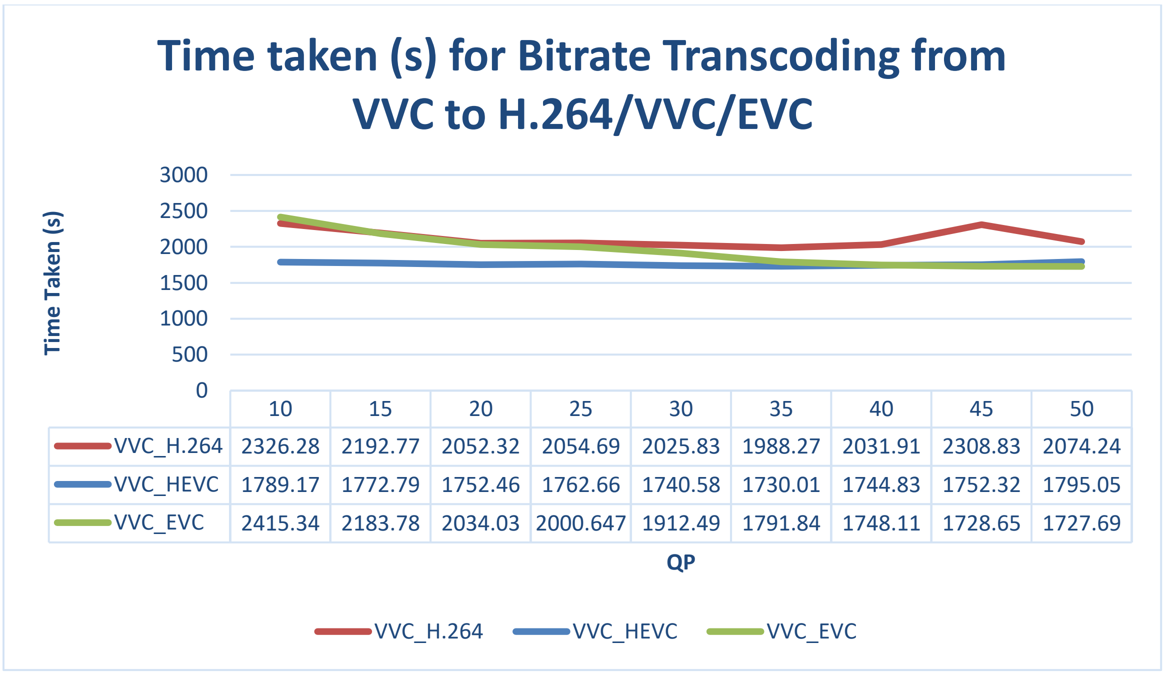
	VVC			EVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	1936.459	939.0048	10	731.388	154.1995	53.4656	0.999462	2415.34
2				15	530.338	93.5819	51.05207	0.99922	2183.78
3				20	378.178	59.4811	48.6654	0.99888	2034.03
4				25	377.845	38.6624	45.93787	0.998151	2000.647
5				30	182.813	25.3115	42.89023	0.99649	1912.49
6				35	140.33	16.6661	40.10603	0.992418	1791.84
7				40	112.493	10.9157	37.02683	0.982868	1748.11
8				45	104.261	7.2384	34.17297	0.96048	1728.65
9				50	105.44	4.8635	31.3204	0.914966	1727.69

Table 7.17. VVC to EVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
VVC_Bitrate1	939.0048	939.0048	939.0048	939.0048	939.0048	939.0048	939.0048	939.0048	939.0048
H.264_Bitrate2	362.25	200.45	111.22	60.61	31.47	18.45	11.46	7.28	5.15
HEVC_Bitrate2	1007.3056	582.6464	364.3168	235.984	154.1088	102.4064	69.0496	48.1728	35.3376
EVC_Bitrate2	154.1995	93.5819	59.4811	38.6624	25.3115	16.6661	10.9157	7.2384	4.8635

Figure 7.22. Heterogeneous Bitrate Transcoding from VVC to H.264/HEVC and EVC using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
VVC_H.264	2326.28	2192.77	2052.32	2054.69	2025.83	1988.27	2031.91	2308.83	2074.24
VVC_HEVC	1789.17	1772.79	1752.46	1762.66	1740.58	1730.01	1744.83	1752.32	1795.05
VVC_EVC	2415.34	2183.78	2034.03	2000.647	1912.49	1791.84	1748.11	1728.65	1727.69

Figure 7.23. Time taken for Heterogeneous Bitrate Transcoding from VVC to H.264/HEVC and EVC using akiyo_qcif sequence

	EVC			H.264					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	715.152	157.9253	10	309.116	351.2	53.1797	0.99923	1027.28
2				15	309.384	196.34	50.091	0.9989	1034.17
3				20	305.765	109.43	46.7753	0.9982	1022.64
4				25	308.913	59.66	43.4373	0.9968	1024.58
5				30	309.585	31.37	40.0933	0.9935	1026.53
6				35	317.415	18.42	37.255	0.9882	1028.36
7				40	334.878	11.44	34.7977	0.977133	1050.23
8				45	384.146	7.26	33.0757	0.966166	1094.6
9				50	382.997	5.16	31.6463	0.94037	1129.24

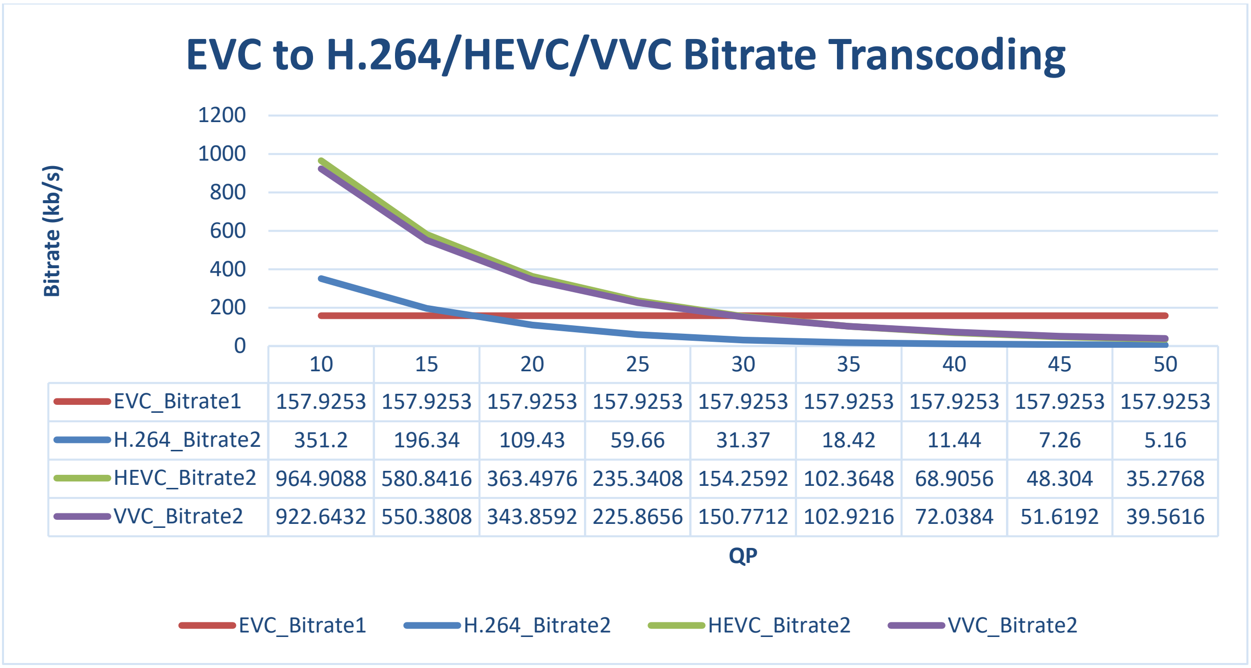
Table 7.18. EVC to H.264 Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

	EVC			HEVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSSIM (YUV)	Transcoding Time (s)
1	10	715.152	157.9253	10	85.744	964.9088	53.7351	0.999643	819.79
2				15	72.567	580.8416	50.3782	0.999259	810.668
3				20	59.158	363.4976	47.3547	0.998433	815.84
4				25	59.501	235.3408	44.2211	0.996703	820.294
5				30	54.989	154.2592	40.8138	0.992761	812.536
6				35	51.527	102.3648	37.422	0.98645	937.475
7				40	50.188	68.9056	34.105	0.97498	778.597
8				45	49.362	48.304	31.0905	0.959838	772.678
9				50	48.678	35.2768	28.2904	0.923841	762.15

Table 7.19. EVC to HEVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence

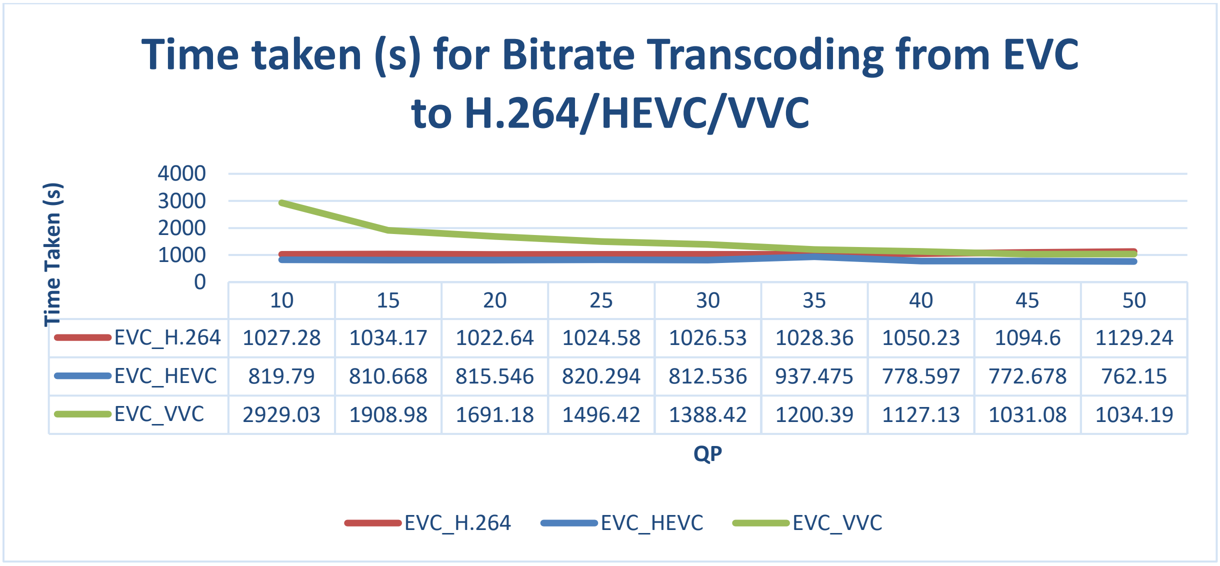
	EVC			VVC					
Sl.No	QP1	Encoding time (s)	Bitrate1 (kbits/sec)	QP2	Encoding time (s)	Bitrate2 (kbits/sec)	PSNR (YUV)	MSE (YUV)	Transcoding Time (s)
1	10	1936.459	939.0048	10	2120.023	922.6432	55.2343	3.1172	2929.03
2				15	1185.33	550.3808	51.651	7.1138	1908.98
3				20	943.138	343.8592	48.5156	14.6434	1691.18
4				25	710.216	225.8656	45.4637	29.5684	1496.42
5				30	552.37	150.7712	42.1973	62.7299	1388.42
6				35	438.974	102.9216	38.9434	132.6956	1200.39
7				40	360.072	72.0384	35.7039	279.7759	1127.13
8				45	281.243	51.6192	32.4376	593.5231	1031.08
9				50	296.761	39.5616	29.3703	1202.7429	1034.19

Table 7.20. EVC to VVC Heterogeneous Bitrate Transcoding using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
EVC_Bitrate1	157.9253	157.9253	157.9253	157.9253	157.9253	157.9253	157.9253	157.9253	157.9253
H.264_Bitrate2	351.2	196.34	109.43	59.66	31.37	18.42	11.44	7.26	5.16
HEVC_Bitrate2	964.9088	580.8416	363.4976	235.3408	154.2592	102.3648	68.9056	48.304	35.2768
VVC_Bitrate2	922.6432	550.3808	343.8592	225.8656	150.7712	102.9216	72.0384	51.6192	39.5616

Figure 7.24. Heterogeneous Bitrate Transcoding from EVC to H.264/HEVC and VVC using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
EVC_H.264	1027.28	1034.17	1022.64	1024.58	1026.53	1028.36	1050.23	1094.6	1129.24
EVC_HEVC	819.79	810.668	815.546	820.294	812.536	937.475	778.597	772.678	762.15
EVC_VVC	2929.03	1908.98	1691.18	1496.42	1388.42	1200.39	1127.13	1031.08	1034.19

Figure 7.25. Time taken for Heterogeneous Bitrate Transcoding from EVC to H.264/HEVC and VVC using akiyo_qcif sequence

7.6.2. Objective Quality analysis for Homogeneous Bitrate Transcoding – QP variation

Homogeneous bitrate change with QP variation is coded as in Figure 6.6. and results tabulated from Table 7.21. to Table 7.24 and analyzed in Figure 7.26 and Figure 7.27. for a video sequence akiyo_qcif.yuv (raw video), quantization parameter is kept constant (QP1= 10) at the first codec level. Homogeneous transcoding at second stage for different QP values (QP2 = 10, 15, 20, 25, 30, 35, 40, 45 and 50) are tested and corresponding Bitrate2 values are noted. I was observed that for a constant Bitrate1 at stage 1, the Bitrate2 decreased with increase in QP value and Bitrate2 increased with decrease in QP value at stage 2.

H.264		H.264			
QP1	Bitrate1 (kbits/sec)	QP2	Bitrate2 (kbits/sec)	PSNR (YUV)	Transcoding Time (s)
10	2053.05	10	404	61.053	1091.87
		15	213	49.791	1052.56
		20	111.57	46.549	993.15
		25	59.93	43.333	729.31
		30	31.38	40.067	650.492
		35	18.37	37.251	672.188
		40	11.48	34.831	707.719
		45	7.25	32.992	668.476
		50	5.18	31.638	688.549

Table 7.21. H.264 to H.264 Homogeneous Bitrate Transcoding using akiyo_qcif sequence

HEVC		HEVC			
QP1	Bitrate1 (kbits/sec)	QP2	Bitrate2 (kbits/sec)	PSNR (YUV)	Transcoding Time (s)
10	1051.2256	10	1032.95	61.43	179.703
		15	583.39	50.3749	162.037
		20	364.2752	47.2429	150.237
		25	235.1072	44.1169	145.669
		30	154.6912	40.7603	141.493
		35	102.672	37.4108	140.926
		40	69.248	34.1173	140.361
		45	48.179	31.0407	139.3
		50	35.408	28.3067	138.546

Table 7.22. HEVC to HEVC Homogeneous Bitrate Transcoding using akiyo_qcif sequence

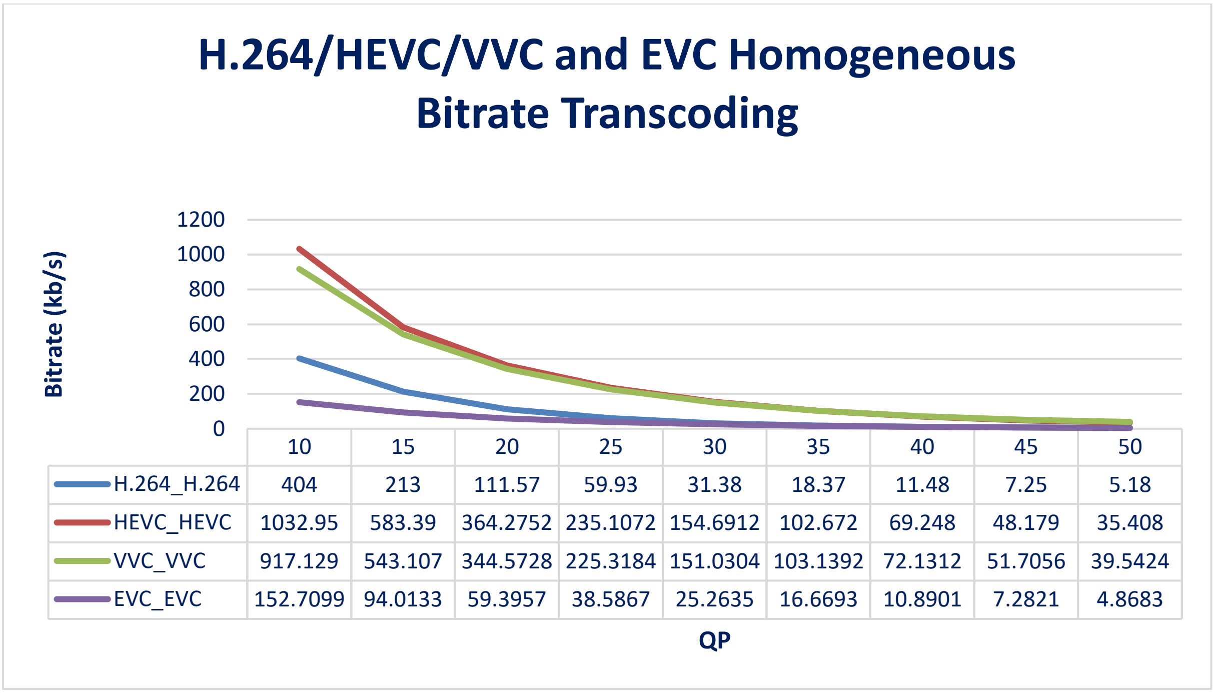
VVC		VVC			
QP1	Bitrate1 (kbits/sec)	QP2	Bitrate2 (kbits/sec)	PSNR (YUV)	Transcoding Time (s)
10	939.005	10	917.129	56.4621	3223.4
		15	543.107	51.6116	2901.31
		20	344.5728	48.4937	2572.79
		25	225.3184	45.4283	2373.23
		30	151.0304	42.1947	2201.96
		35	103.1392	38.9641	2096.95
		40	72.1312	35.732	2181.26
		45	51.7056	32.4129	1988.83
		50	39.5424	29.3488	2012.41

Table 7.23. VVC to VVC Homogeneous Bitrate Transcoding using akiyo_qcif sequence

EVC		EVC			
QP1	Bitrate1 (kbits/sec)	QP2	Bitrate2 (kbits/sec)	PSNR (YUV)	Transcoding Time (s)
10	157.9253	10	152.7099	55.2274	1398.73
		15	94.0133	51.6547	1308.9
		20	59.3957	48.91743	1159.42
		25	38.5867	46.04373	1058.91
		30	25.2635	42.94913	959.242
		35	16.6693	40.1439	916.423
		40	10.8901	37.124466	887.878
		45	7.2821	34.249833	881.735
		50	4.8683	31.438	861.95

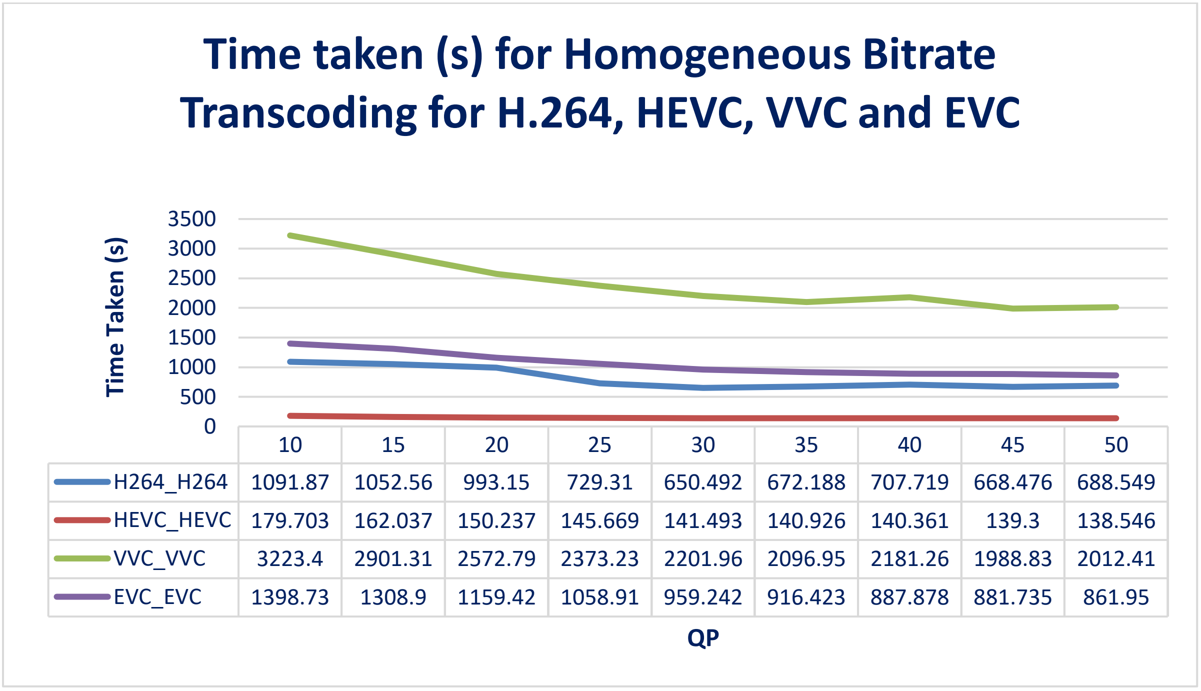
Table 7.24. EVC to EVC Homogeneous Bitrate Transcoding using akiyo_qcif sequence

From Figure 7.26, it was observed that the bitrate2 at second stage of the transcoding pipeline reduced with increase in QP value from 10 to 50. This characteristic was observed for all the four codec (H.264, HEVC, VVC and EVC) homogeneous bitrate change for QP variation using akiyo_qcif content. Figure 7.27 showed the time taken for homogeneous bitrate transcoding using QP variation. It was evident from the graph that VVC took more time to transcode followed by EVC, H.264 and then HEVC.



	10	15	20	25	30	35	40	45	50
H.264_H.264	404	213	111.57	59.93	31.38	18.37	11.48	7.25	5.18
HEVC_HEVC	1032.95	583.39	364.2752	235.1072	154.6912	102.672	69.248	48.179	35.408
VVC_VVC	917.129	543.107	344.5728	225.3184	151.0304	103.1392	72.1312	51.7056	39.5424
EVC_EVC	152.7099	94.0133	59.3957	38.5867	25.2635	16.6693	10.8901	7.2821	4.8683

Figure 7.26. Homogeneous Bitrate Transcoding for H.264/HEVC/VVC and EVC using akiyo_qcif sequence



	10	15	20	25	30	35	40	45	50
H264_H264	1091.87	1052.56	993.15	729.31	650.492	672.188	707.719	668.476	688.549
HEVC_HEVC	179.703	162.037	150.237	145.669	141.493	140.926	140.361	139.3	138.546
VVC_VVC	3223.4	2901.31	2572.79	2373.23	2201.96	2096.95	2181.26	1988.83	2012.41
EVC_EVC	1398.73	1308.9	1159.42	1058.91	959.242	916.423	887.878	881.735	861.95

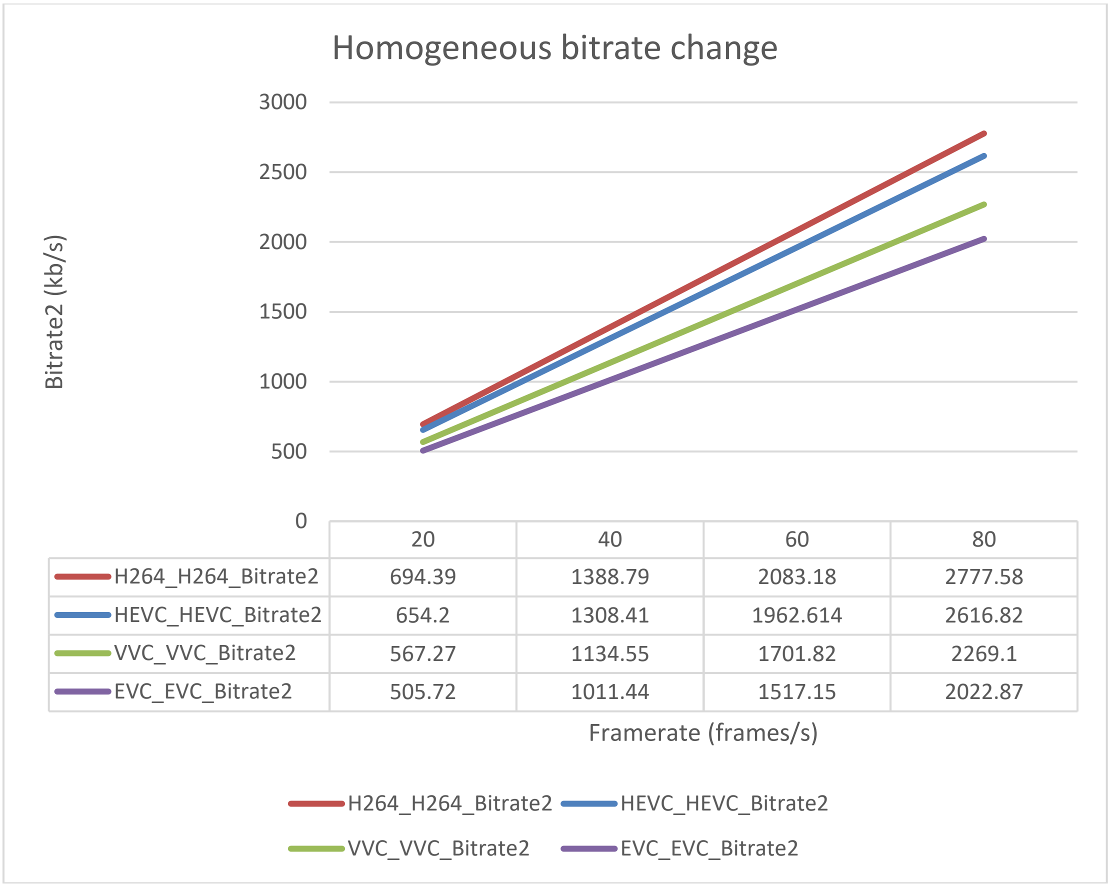
Figure 7.27. Time taken for Homogeneous Bitrate Transcoding for H.264/HEVC/VVC and EVC using akiyo_qcif sequence

7.6.3. Objective Quality analysis for Homogeneous Bitrate Transcoding – Framerate variation

Homogeneous bitrate change with frame rate variation is coded as in Figure 6.7. and results tabulated in Table 7.25. Here the framerate at the first block is Framerate1 and second block is Framerate2. Simultaneously, the Bitrate at the first block of transcoding is Bitrate1 and second block is Bitrate2. The compressed output at the first block is Compressed output1 and second block is Compressed output2.

Video Transcoding	Frame Rate1	Frame Rate2	Bitrate 1 (kb/s)	Bitrate 2 (kb/s)	PSNR (dB)	SSIM	Time taken to encode (s)	Compressed Output1 (kB)	Compressed Output2 (kB)
H264_H264	20	20	418.61	694.39	72.591	1	1748.86	752	1272
		40	418.61	1388.79	72.591	1	1682.81	752	1272
		60	418.61	2083.18	72.591	1	634.699	752	1272
		80	418.61	2777.58	72.591	1	730.182	752	1272
	100	20	2053.05	694.39	72.591	1	665.928	752	1272
		40	2053.05	1388.79	72.591	1	914.408	752	1272
		60	2053.05	2083.18	72.591	1	1013.93	752	1272
		80	2053.05	2777.58	72.591	1	720.419	752	1272
HEVC_HEVC	20	20	658.1024	654.2	75.03	1	381.034	1206	1198
		40	658.1024	1308.41	75.03	1	359.705	1206	1198
		60	658.1024	1962.614	75.03	1	356.496	1206	1198
		80	658.1024	2616.82	75.03	1	391.468	1206	1198
	100	20	3290.512	654.2	75.03	1	350.726	1206	1198
		40	3290.512	1308.41	75.03	1	352.791	1206	1198
		60	3290.512	1962.614	75.03	1	352.572	1206	1198
		80	3290.512	2616.82	75.03	1	356.176	1206	1198
VVC_VVC	20	20	576.54	567.27	61.34	1	7045.8	1056	1039
		40	576.54	1134.55	61.34	1	6805.6	1056	1039
		60	576.54	1701.82	61.34	1	6764.47	1056	1039
		80	576.54	2269.1	61.34	1	6885.53	1056	1039
	100	20	2882.704	567.2736	61.34	1	6899.39	1056	1039
		40	2882.704	1134.55	61.34	1	6999.85	1056	1039
		60	2882.704	1701.82	61.34	1	6933.82	1056	1039
		80	2882.704	2269.09	61.34	1	6933.36	1056	1039
EVC_EVC	20	20	499.0133	505.72	57.34	1	3625.72	914	926
		40	499.0133	1011.44	57.34	1	3548.55	914	926
		60	499.0133	1517.15	57.34	1	3611.6	914	926
		80	499.0133	2022.87	57.34	1	3600.76	914	926
	100	20	2495.07	505.72	57.34	1	3621.16	914	926
		40	2495.07	1011.44	57.34	1	3598.93	914	926
		60	2495.07	1517.15	57.34	1	3830.2	914	926
		80	2495.07	2022.87	57.34	1	3736.53	914	926

Table 7.25. Homogeneous Bitrate Transcoding using framerate variation



	20	40	60	80
🟥 H264_H264_Bitrate2	694.39	1388.79	2083.18	2777.58
🟦 HEVC_HEVC_Bitrate2	654.2	1308.41	1962.614	2616.82
🟩 VVC_VVC_Bitrate2	567.27	1134.55	1701.82	2269.1
🟪 EVC_EVC_Bitrate2	505.72	1011.44	1517.15	2022.87

Figure 7.28. Homogeneous bitrate change with framerate variation.

It is very evident from the graph in Figure 7.28. that the bitrate2 increases with increase in framerate. The compression size of the output video remains the same with increase in framerate but the time taken to encode reduces with increase in framerate. This is one of the main characteristics of framerate variation. We also observe that the PSNR and SSIM remains same and is maximum as quantization parameter here is kept to zero. This same characteristic is observed for all four codec (H.264, HEVC, VVC and EVC) change.

7.7. Results of Objective 4: Develop an automated versatile application of Transcoding H.264, HEVC, VVC and EVC for format change and bitrate variation (QP and framerate change)

The automated application was coded and tested accordingly as seen in Appendix 1 and Appendix 2.

CHAPTER 8: Conclusion and Future Work

Transcoding is one of the main applications in wireless and video communication. This application is always in demand untill there is code developments of new video codecs ongoing. The VVC (Versatile Video Coding) and EVC (Essential Video Coding) being the next generation video codecs was the main focus point as the codecs brings in a lot of application uses along with its additional algorithmic development. Since VVC and EVC are still not out in the market, there is a lot of scope in transcoding other codecs to VVC/EVC and vice versa. All these video codecs are open source other than EVC, and used by the industry standards, it is advantages to choose them as the main codec. The complexity of VVC and EVC were also proved to be high, hence opens up avenues for research in reducing the complexity of the codec algorithms. Other applications like multiplexing, video on demand, internet video can also be developed using these new codecs. Along with heterogeneous functionality, homogeneous properties like bitrate change and frame rate changes were also tested which are the main features for wireless communication. Each of these functionalities were divided into 3 different objectives and the conclusion for the same is provided in the next session. The objective 4 gives the automated application which is presented during the demonstration of the thesis.

8.1. Conclusion of Objective 1: Codec comparative analysis

The codec analysis using the three methods of verification (Objective analysis, RD method and Subjective analysis) concluded by providing the same results in terms of codec performance with respect to four codec software implementations. The testing verified that VVC showed better bitrate savings for the encoded bit sequence, that is 30% – 50% better when compared to HEVC codecs keeping the video quality constant. This is due to the VVC having progressive block implementations when compared to the other codecs, like having twice the block structure when compared to EVC which increases the compression rate, hence reduces the bitrate for the same QP. EVC showed 10% – 30% better bitrate savings when compared to HEVC keeping the video quality constant. This is because of EVC having twice the maximum block size when compared to HEVC. HEVC providing 50% better bitrate savings than H.264 for same output video quality was already proved in the research papers [68] and hence was also verified in this thesis.

Complexity of the codec is another domain which opens for research to improve the codec time performance. VVC as the codec with software version 10.0 showed more time to encode the sequence when compared to EVC, HEVC and H.264. The time taken was almost double the time taken for EVC to encode a 4K resolution video. This is also due to the intra prediction modes in VVC being maximum which is 89 when compared to EVC which is 30, HEVC which is 35 and H.264 which is 9. This opens for avenues in research in reducing the complexity of the codec.

8.2. Conclusion of Objective 2: Heterogeneous Transcoding

Format change between H.264, HEVC, VVC and EVC is known as heterogeneous transcoding which was developed and tested for QP = 10 and GOP=15. The transcoded bitstream of the transcoder for VVC to HEVC and HEVC to VVC showed better PSNR and SSIM value, as it was proved in objective 1, as VVC was a better codec with better output results when compared to EVC, HEVC and H.264. So, any codec shift which involved VVC showed better output results and more compression size. It was also noted in objective 1 that time taken to encode VVC and EVC was maximum, like VVC taking two times more time than EVC and EVC 1/3 time more than HEVC for its encoding. The same was observed in heterogeneous transcoding with time taken for VVC and EVC was maximum, hence its combination in transcoding showed maximum time. Any transcoding combination with respect to H.264 showed least time to transcode. This is due to the complexity increment in codec in the ascending order as H.264, HEVC, EVC and VVC. Hence when the codec is more complex, more time is taken to encode a sequence using that codec.

Hence can be concluded that any video communication application required to transmit any data with more compression having less bandwidth availability can use heterogeneous transcoding from any codec to VVC. The time taken for transcoding this combination is also more but can be ignored if the output video quality is important along with more compression rate. This is more effective for higher data videos like 4K and 8K videos.

8.3. Conclusion of Objective 3: Bitrate Homogeneous/Heterogeneous Transcoding

Heterogeneous / Homogeneous Transcoding of H.264, HEVC, VVC and EVC video codecs for bitrate changes using cascaded decoder-encoder pixel transcoding architecture for QP variation and frame rate variation, results showed very good output quality video. Heterogeneous transcoding for bitrate change using QP variation in the second codec in the transcoding pipeline, showed very good PSNR and SSIM output values for the first codec QP =10 in the transcoding pipeline. The PSNR value was almost equal to 50.5dB and SSIM value equal to 0.999863 which is almost lossless output values. It was also observed that when QP increases in the second codec in the pipeline, the bitrate decreases, hence achieving better compression rate with good PSNR and SSIM value. It was also observed that when QP decreases in the second codec in the pipeline, the bitrate increases, hence reducing the compression rate but the PSNR and SSIM value remained good.

Homogeneous transcoding for bitrate change using framerate variation in the second codec in the transcoding pipeline, showed stable PSNR and SSIM value with variation in the bitrate for the first and second codec QP = 0 in the transcoding pipeline. It was observed that when frame rate increases in the second codec in the pipeline, bitrate decreases and when framerate decreases in the second codec in the pipeline, bitrate increases. Here compression rate remained constant for framerate variation. Bitrate variable is directly proportional to time taken to transcode, hence it was observed that when bitrate reduces, time taken for bitrate transcoding decreases and when bitrate increases, time taken for bitrate transcoding increases.

Hence can be concluded that any video communication application required to transmit any data quick with less bitrate, can use heterogeneous transcoding with QP variation, for higher QP value, as time taken is low and homogeneous transcoding with framerate variation, for lower framerate value, as time taken for transcoding is less.

REFERENCES

[1] Sayood, K., 2002. *Lossless compression handbook*. Elsevier.

[2] Goyal, V.K., Fletcher, A.K. and Rangan, S., 2008. Compressive sampling and lossy compression. *IEEE Signal Processing Magazine*, *25*(2), pp.48-56.

[3] Pan, Z., Qin, H., Yi, X., Zheng, Y. and Khan, A., 2019. Low complexity versatile video coding for traffic surveillance system. *International Journal of Sensor Networks*, *30*(2), pp.116-125.

[4] Mayer, R.E., 2002. Multimedia learning. In *Psychology of learning and motivation* (Vol. 41, pp. 85-139). Academic Press.

[5] Ebrahimi, T. and Kunt, M., 1998. Visual data compression for multimedia applications. *Proceedings of the IEEE*, *86*(6), pp.1109-1125.

[6] Wu, T.B. and Rao, K.R., 1982. Digital TV Receiver for NTSC Color TV Signals with Dual Word-Length DPCM Coding. *IEEE Transactions on Broadcasting*, (1), pp.20-24.

[7] Wu, T.B. and Rao, K.R., 1982. Digital TV Receiver for NTSC Color TV Signals with Dual Word-Length DPCM Coding. *IEEE Transactions on Broadcasting*, (1), pp.20-24.

[8] Bankoski, J., Wilkins, P. and Xu, Y., 2011, July. Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*(pp. 1-6). IEEE.

[9] Ghanbari, M., 1992. An adapted H. 261 two-layer video codec for ATM networks. *IEEE Transactions on Communications*, *40*(9), pp.1481-1490.

[10] Hanzo, L., Cherriman, P. and Streit, J., 2007. *Video Compression and Communications.: From Basics to H. 261, H. 263, H. 264, MPEG4 for DVB and HSDPA-Style Adaptive Turbo-Transceivers*. John Wiley & Sons.

[11] Sullivan, G.J., Ohm, J.R., Han, W.J. and Wiegand, T., 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, *22*(12), pp.1649-1668.

[12] Chang, H.C., Chen, L.G., Hsu, M.Y. and Chang, Y.C., 2000, May. Performance analysis and architecture evaluation of MPEG-4 video codec system. In *2000 IEEE International Symposium on*

Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No. 00CH36353)(Vol. 2, pp. 449-452). IEEE.

[13] Shi, C. and Bhargava, B., 1998, October. An efficient MPEG video encryption algorithm. In *Proceedings Seventeenth IEEE Symposium on Reliable Distributed Systems (Cat. No. 98CB36281)* (pp. 381-386). IEEE.

[14] Mukherjee, D., Han, J., Bankoski, J., Bultje, R., Grange, A., Koleszar, J., Wilkins, P. and Xu, Y., 2013, October. A technical overview of vp9–the latest open-source video codec. In *SMPTE 2013 Annual Technical Conference & Exhibition* (pp. 1-17). SMPTE.

[15] Kufa, J. and Kratochvil, T., 2015, April. Comparison of H. 265 and VP9 coding efficiency for full HDTV and ultra HDTV applications. In *2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA)* (pp. 168-171). IEEE.

[16] Topiwala, P., Krishnan, M. and Dai, W., 2018, September. Performance comparison of VVC, AV1, and HEVC on 8-bit and 10-bit content. In *Applications of Digital Image Processing XLI* (Vol. 10752, p. 107520V). International Society for Optics and Photonics.

[17] Abdoli, M., Henry, F., Brault, P., Duhamel, P. and Dufaux, F., 2018. Short-distance intra prediction of screen content in versatile video coding (VVC). *IEEE Signal Processing Letters*, *25*(11), pp.1690-1694.

[18] Chen, Y., Murherjee, D., Han, J., Grange, A., Xu, Y., Liu, Z., Parker, S., Chen, C., Su, H., Joshi, U. and Chiang, C.H., 2018, June. An overview of core coding tools in the AV1 video codec. In *2018 Picture Coding Symposium (PCS)* (pp. 41-45). IEEE.

[19] Grois, D., Nguyen, T. and Marpe, D., 2018, February. Performance comparison of AV1, JEM, VP9, and HEVC encoders. In *Applications of Digital Image Processing XL* (Vol. 10396, p. 103960L). International Society for Optics and Photonics.

[20] D.N. Kim and K.R. Rao, "Current Video Coding Standards: H.264/AVC, Dirac, AVS China and VC-1', Current developments in theory and applications of computer science, engineering and technology, vol. 1, pp. 67-94, 2010.

[21] Advanced video coding for generic audiovisual services, ITU-T Rec. H.264 / ISO / IEC 14496-10, Nov. 2009.

[22] H.263 basics Website link : http://en.wikipedia.org/wiki/H.263

[23] G. Sullivan, P. Topiwala and A. Luthra, ―The H.264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions‖, SPIE conference on Applications of Digital Image Processing XXVII, vol. 5558, pp. 53-74, Aug. 2004.

[24] Open source article, ―H.264/MPEG-4 AVC,‖ Wikipedia Foundation, http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

[25] T. Wiegand and G. J. Sullivan, ―The H.264 video coding standard‖, IEEE Signal Processing Magazine, vol. 24, pp. 148-153, March 2007.

[26] Ramzan, N., Pervez, Z. and Amira, A., 2014, December. Quality of experience evaluation of H. 265/MPEG-HEVC and VP9 comparison efficiency. In *2014 26th International Conference on Microelectronics (ICM)* (pp. 220-223). IEEE.

[27] Bankoski, J., Bultje, R.S., Grange, A., Gu, Q., Han, J., Koleszar, J., Mukherjee, D., Wilkins, P. and Xu, Y., 2013, February. Towards a next generation open-source video codec. In *Visual Information Processing and Communication IV* (Vol. 8666, p. 866606). International Society for Optics and Photonics.

[28] Kim, I.K., Lee, S., Piao, Y. and Chen, J., 2014, July. Coding efficiency comparison of new video coding standards: HEVC vs VP9 vs AVS2 video. In *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)* (pp. 1-6). IEEE.

[29] Chen, Y., Murherjee, D., Han, J., Grange, A., Xu, Y., Liu, Z., Parker, S., Chen, C., Su, H., Joshi, U. and Chiang, C.H., 2018, June. An overview of core coding tools in the AV1 video codec. In *2018 Picture Coding Symposium (PCS)* (pp. 41-45). IEEE.

[30] T. Wiegand et al, ―Overview of the H.264/AVC video coding standard‖, IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, pp. 560-576, Jul. 2003.

[31] C. Deng et al, ―Performance analysis, parameter selection and extensions to H.264/AVC FRExt for high resolution video coding‖, J. Vis. Commun. Image R., vol. 22 (In Press), Available on line, Feb. 2011.

[32] Z.Wang, E.P.Simoncelli and A.C.Bovik, ―Multi-scale structural similarity for image quality assessment‖, Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, vol. 2, Nov. 2003.

[33] I. E.G. Richardson, ―H.264 and MPEG-4 video compression: video coding for next-generation multimedia‖, Wiley, 2003.

[34] D. Marpe, T. Wiegand and G. J. Sullivan, ―The H.264/MPEG-4 AVC standard and its applications‖, IEEE Communications Magazine, vol. 44, pp. 134-143, Aug. 2006.

[35] J. Boyce et al. "Draft high efficiency video coding (HEVC) version 2, combined format range extensions (RExt), scalability (SHVC), and multi-view (MV-HEVC) extensions", JCT-VC, July 11, 2014.

[36] HEVC white paper: http://www.ateme.com/an-introduction-to-uhdtv-and-hevc

[37] G.J. Sullivan et al, ―Standardized Extensions of High Efficiency Video Coding (HEVC)‖, IEEE Journal of selected topics in Signal Processing, vol. 7, no. 6, pp. 1001-1016, Dec. 2013.

[38] K. Iguchi et al, ―HEVC Encoder for Super Hi-Vision‖, 2014 IEEE International conference on Consumer Electronics (ICCE), pp. 61-62, 2014.

[39] K.R. Rao and J.J. Hwang, ―Techniques and standards for image/video/audio coding,‖ Prentice-Hall, 1996.

[40] HEVC tutorial by I.E.G. Richardson: http://www.vcodex.com/h265.html

[41] Moto, ―HEVC - What are CTU, CU, CTB, CB, PB and TB?‖ CODE: Sequoia, worgpress.com site, blog. [online]. Available: http://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/ (accessed on August 9th 2020).

[42] S. Riabstev, ― Detailed overview of HEVC/H.265‖, [online]. Available: https://app.box.com/s/rxxxzr1a1lnh7709yvih (accessed on August 9th 2020).

[43] I.K. Kim et al, ―Block partitioning structure in the HEVC standard, IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, pp. 1697-1706, Dec. 2012.

[44] M.T. Pourazad et al, ―HEVC: The new gold standard for video compression,‖ IEEE CE Magazine, pp. 36-46, vol. 1, issue 3, July 2012.

[45] F. Bossen et al, ―HEVC Complexity and Implementation Analysis‖, IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1685-1696, Dec. 2012.

[46] M.T. Pourazad et al, ―HEVC: The new gold standard for video compression,‖ IEEE CE Magazine, pp. 36-46, vol. 1, issue 3, July 2012.

[47] S. Vasudevan and K.R. Rao, ―Combination method of fast HEVC encoding,‖ IEEE ECTICON 2014, Korat, Thailand, May 2014.

[48] S. Lui et al, ―Video Prediction Block Structure and the Emerging High Efficiency Video Coding Standard‖, IEEE proceedings on Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific, pp. 1-4, 2012.

[49] I.G. Kim et al., High Efficiency Video Coding (HEVC) Test Model 15 (HM15) Encoder Description, JCTVC-Q1002, April 2014.

[50] K. McCann et al, ―HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description‖, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1002v2, Oct. 2012.

[51] J. Chen, J. Boyce, Y. Ye, and M. M. Hannuksela, ―Scalable high efficiency video coding draft 3,‖ in Joint Collaborative Team on Video Coding (JCT-VC) Document JCTVC-N1008, 14th Meeting: Vienna, Austria, July 25–Aug. 2 2013.

[52] Vetro, A., Christopoulos, C. and Sun, H., 2003. Video transcoding architectures and techniques: an overview. *IEEE Signal processing magazine*, *20*(2), pp.18-29.

[53] Xin, J., Lin, C.W. and Sun, M.T., 2005. Digital video transcoding. *Proceedings of the IEEE*, *93*(1), pp.84-97.

[54] Sharma, S. and Rao, K.R., 2007, October. Transcoding of H. 264 bitstream to MPEG-2 bitstream. In *2007 Asia-Pacific Conference on Communications* (pp. 391-396). IEEE.

[55] Ahmad, I., Wei, X., Sun, Y. and Zhang, Y.Q., 2005. Video transcoding: an overview of various techniques and research issues. *IEEE Transactions on multimedia*, *7*(5), pp.793-804.

[56] Topiwala, P., Krishnan, M. and Dai, W., 2019, September. Performance comparison of VVC, AV1 and EVC. In *Applications of Digital Image Processing XLII* (Vol. 11137, p. 1113715). International Society for Optics and Photonics.

[57] JM Software: https://github.com/shihuade/JM.git

[58] HM Software: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/trunk/

[59] VTM Software: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM.git

[60] ETM Software: https://gitlab.com/MPEG-5/ETM

[61] Hore, A. and Ziou, D., 2010, August. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition* (pp. 2366-2369). IEEE.

[62] Thorpe, S., Fize, D. and Marlot, C., 1996. Speed of processing in the human visual system. *nature*, *381*(6582), pp.520-522.

[63] Panetta, K.A., Wharton, E.J. and Agaian, S.S., 2008. Human visual system-based image enhancement and logarithmic contrast measure. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *38*(1), pp.174-188.

[64] Test videos found at: https://media.xiph.org/video/derf/

[65] Test videos found at: http://ultravideo.cs.tut.fi/#testsequences

[66] Abdoli, M., Henry, F., Brault, P., Dufaux, F. and Duhamel, P., 2019, May. Transform Coefficient Coding for Screen Content in Versatile Video Coding (VVC). In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1792-1796). IEEE.

[67] Choi, K., Park, M.W., Choi, K.P., Park, J., Chen, J., Wang, Y.K., Chernyak, R., Ikonin, S., Rusanovskyy, D., Chien, W.J. and Seregin, V., 2019, September. MPEG-5: essential video coding standard. In *Applications of Digital Image Processing XLII* (Vol. 11137, p. 1113710). International Society for Optics and Photonics.

[68] Ohm, J.R., Sullivan, G.J., Schwarz, H., Tan, T.K. and Wiegand, T., 2012. Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC). *IEEE Transactions on circuits and systems for video technology*, *22*(12), pp.1669-1684.

[69] Wiegand, T., Sullivan, G.J., Bjontegaard, G. and Luthra, A., 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, *13*(7), pp.560-576.

[70] H.264/MPEG-4 AVC: https://en.wikipedia.org/wiki/H.264%2FMPEG-4_AVC

[71] Wang, S., Zhang, X., Wang, S., Ma, S. and Gao, W., 2019, March. Adaptive Wavelet Domain Filter for Versatile Video Coding (VVC). In *2019 Data Compression Conference (DCC)* (pp. 73-82). IEEE.

[72] Fu, T., Zhang, H., Mu, F. and Chen, H., 2019, July. Fast CU Partitioning Algorithm for H. 266/VVC Intra-Frame Coding. In *2019 IEEE International Conference on Multimedia and Expo (ICME)* (pp. 55-60). IEEE.

[73] Do, J., Park, D., Kim, J.G. and Jeong, D.G., 2018, October. Block-shape adaptive inter prediction candidate list in Versatile Video Coding. In *TENCON 2018-2018 IEEE Region 10 Conference* (pp. 0215-0218). IEEE.

[74] Sun, Y.C., Lou, J., Chao, Y.H., Wang, H., Seregin, V. and Karczewicz, M., 2019, March. Analysis of Palette Mode on Versatile Video Coding. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)* (pp. 455-458). IEEE.

[75] Zhang, F., Feng, C. and Bull, D.R., 2020, July. Enhancing VVC through CNN-based Post-Processing. In *2020 IEEE International Conference on Multimedia and Expo (ICME)* (pp. 1-6). IEEE.

[76] Cho, S., Kim, D.W. and Jung, S.W., 2020. Quality enhancement of VVC intra-frame coding for multimedia services over the Internet. *International Journal of Distributed Sensor Networks*, *16*(5), p.1550147720917647.

[77] Li, T., Xu, M. and Tang, R., 2020. DeepQTMT: A Deep Learning Approach for Fast QTMT-based CU Partition of Intra-mode VVC. *arXiv preprint arXiv:2006.13125*.

[78] Tang, M., Chen, X., Wen, J. and Han, Y., 2018. Hadamard Transform-Based Optimized HEVC Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, *29*(3), pp.827-839.

[79] Nakamura, K., Omori, Y., Kobayashi, D., Osawa, T., Onishi, T., Nitta, K., Iwasaki, H. and Shimizu, A., 2019, April. Low Delay 4K 120fps HEVC Decoder with Parallel Processing Architecture. In *2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)* (pp. 1-3). IEEE.

[80] Abolfathi, R., Roodaki, H. and Shirmohammadi, S., 2019, February. A Novel Rate Control Method for Free-viewpoint Video in MV-HEVC. In *2019 International Conference on Computing, Networking and Communications (ICNC)* (pp. 582-587). IEEE.

[81] Chen, Y., Yu, L., Li, T., Wang, H. and Wang, S., 2019, March. Fast CU Size Decision Based on AQ-CNN for Depth Intra Coding in 3D-HEVC. In *2019 Data Compression Conference (DCC)* (pp. 561-561). IEEE.

[82] Sakamoto, Y., Yokoyama, R., Takeuchi, M., Matsuo, Y. and Katto, J., 2019, January. Improvement of H. 265/HEVC Encoding for 8K UHDTV by GOP Size and Prediction Mode Selection. In *2019 IEEE International Conference on Consumer Electronics (ICCE)* (pp. 1-2).

[83] Guan, X., Dong, X., Zhang, M. and Liu, Z., 2019, March. Fast Early Termination of CU Partition and Mode Selection Algorithm for Virtual Reality Video in HEVC. In *2019 Data Compression Conference (DCC)* (pp. 576-576). IEEE.

[84] Xu, J., Xu, M., Wei, Y., Wang, Z. and Guan, Z., 2018. Fast H. 264 to HEVC Transcoding: A Deep Learning Method. (pp. 1633 – 1645), Vol.21, *IEEE Transactions on Multimedia.*

[85] Ochoa-Dominguez H. and K. R. Rao, 2019: Versatile video coding, River publishers.

[86] Azgin, H., Mert, A.C., Kalali, E. and Hamzaoglu, I., 2018, August. A Reconfigurable Fractional Interpolation Hardware for VVC Motion Compensation. In *2018 21st Euromicro Conference on Digital System Design (DSD)* (pp. 99-103). IEEE.

[87] CanMert, A., Kalali, E. and Hamzaoglu, I., 2018, October. A Low Power Versatile Video Coding (VVC) Fractional Interpolation Hardware. In *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)* (pp. 43-47). IEEE.

[88] Fu, T., Zhang, H., Mu, F. and Chen, H., 2019, July. Two-Stage Fast Multiple Transform Selection Algorithm for VVC Intra Coding. In *2019 IEEE International Conference on Multimedia and Expo (ICME)* (pp. 61-66). IEEE.

[89] Venugopal, G., Helle, P., Mueller, K., Marpe, D. and Wiegand, T., 2019, March. Hardware-Friendly Intra Region-Based Template Matching for VVC. In *2019 Data Compression Conference (DCC)* (pp. 606-606). IEEE.

[90] Fan, K., Wang, R., Lin, W., Hou, J.U., Duan, L., Li, G. and Gao, W., 2019, March. Separable KLT for Intra Coding in Versatile Video Coding (VVC). In *2019 Data Compression Conference (DCC)* (pp. 571-571). IEEE.

[91] Abdoli, M., Henry, F., Brault, P., Dufaux, F. and Duhamel, P., 2019, May. Transform Coefficient Coding for Screen Content in Versatile Video Coding (VVC). In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1792-1796). IEEE.

[92] Azgin, H., Mert, A.C., Kalali, E. and Hamzaoglu, I., 2018, August. A Reconfigurable Fractional Interpolation Hardware for VVC Motion Compensation. In *2018 21st Euromicro Conference on Digital System Design (DSD)* (pp. 99-103). IEEE.

[93] S. Park and K.R. Rao, "Bit-Depth Scalable Video Coding Based on H.264/AVC", IEICE Trans. vol. E91-A, pp.1541-1544, June 2008.

[94] S. Park and K.R. Rao, "Hybrid scalable video codec for bit-depth scalability", J. of Optical Engineering, vol. 48, No.1, Jan. 2009.

[95] Rao, K.R., 2007. Sequence mirroring properties of orthogonal transforms having even and odd symmetric vectors.

[96] Kim, D.N. and Rao, K.R., 2008. Two-dimensional discrete sine transform scheme for image mirroring and rotation. *Journal of Electronic Imaging*, *17*(1), p.013011.

[97] Yadav, H. and Rao, K.R., 2006, October. Optimization of the deblocking filter in H. 264 codec for real time implementation. In *2006 International Symposium on Communications and Information Technologies* (pp. 932-936). IEEE.

[98] R. Pereira, K.R. Rao and A. Kruafak, "Efficient transcoding of an MPEG-2 bit stream to an H.264 bit stream", University Scientific Journal series, Telecommunications and Electronics, Poland, pp. 5-38, 2008.

[99] FFmpeg software and official website: www.ffmpeg.org

[100] Zeng, B. and Fu, J., 2008. Directional discrete cosine transforms—A new framework for image coding. *IEEE transactions on circuits and systems for video technology*, *18*(3), pp.305-313.

[101] Li, R., Zeng, B. and Liou, M.L., 1994. A new three-step search algorithm for block motion estimation. *IEEE transactions on circuits and systems for video technology*, *4*(4), pp.438-442.

[102] Said, A. and Pearlman, W.A., 1996. An image multiresolution representation for lossless and lossy compression. *methods*, *1*, p.20.

[103] Richardson, I.E., 2004. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons.

[104] ISO Website: https://www.iso.org/home.html

[105] ITU-T Website: https://www.itu.int/en/ITU-T/Pages/default.aspx

[106] IEC Website: https://www.iec.ch/

BIOGRAPHICAL STATEMENT

Ms. Shreyanka Subbarayappa is currently working as an Assistant Professor in the Department of Electronics and Communication Engineering at one of the reputed universities in India, Ramaiah University of Applied Sciences, Bangalore from 2018. Prior to this, she has had 9+ years of research and industry experience, working for Intel Corporation in US, UK and India as a Team Lead in the graphics driver development for Windows, Linux and Android operating systems. She was appointed as an ‘expert consultant’ for patent litigation cases by Google LLC team, U.S.A. from September 2019 to December 2019.

Shreyanka Subbarayappa received B.E. degree in Telecommunication Engineering from Ramaiah Institute of Technology, Bangalore, India in 2009. She received M.S. degree in Electrical Engineering from The University of Texas at Arlington, Texas, U.S.A. in 2012. She is currently working towards her Ph.D. degree from University of Texas at Arlington, Texas, U.S.A. and Ramaiah University of Applied Sciences, Bangalore, India. Her passion and research interests are in the field of Multimedia Processing, Video Pre-Post Processing, Video Codecs, Image Pre-Post Processing, Image Codecs and Audio Codecs.

```
/*Heterogeneous Transcoding for format change for
H.264, HEVC, VVC and EVC codecs*/

#include <iostream>
#include <stdio.h>
#include <time.h>
#include <ctime>

void H264_HEVC(void) {
         /*H264 Encoder Commandline*/
         std::string cmd = "<path_to_executable>lencod.exe -
         d "<path_to_config_file>encoder_main_H264_H264toHEVC.cfg";
         std::system(cmd.c_str());
         /*H264 Decoder Commandline*/
         cmd = "<path_to_executable>ldecod.exe";
         std::system(cmd.c_str());
         /*HEVC Encoder Commandline*/
         cmd = "<path_to_executable>TAppEncoder.exe -
         c <path_to_config_file>encoder_randomaccess_main_HEVC_H264toHEVC.cfg -
         c <path_to_config_file>akiyo_H264toHEVCnVVC.cfg";
         std::system(cmd.c_str());
         /*HEVC Decoder Commandline*/
         cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8";
         std::system(cmd.c_str());
}

void H264_VVC(void) {
         /*H264 Encoder Commandline*/
         std::string cmd = "<path_to_executable>lencod.exe -
         d <path_to_config_file>encoder_main_H264_H264toHEVC.cfg";
         std::system(cmd.c_str());
         /*H264 Decoder Commandline*/
         cmd = "<path_to_executable>ldecod.exe";
         std::system(cmd.c_str());
         /*VVC Encoder Commandline*/
         cmd = "<path_to_executable>EncoderApp.exe -
         c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_H264toVVC.cfg -
         c <path_to_config_file>akiyo_H264toHEVCnVVC.cfg";
         std::system(cmd.c_str());
         /*VVC Decoder Commandline*/
         cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VCC.yuv -d 8";
         std::system(cmd.c_str());
}

void H264_EVC(void) {
         /*H264 Encoder Commandline*/
         std::string cmd = "<path_to_executable>lencod.exe -
         d <path_to_config_file>encoder_main_H264_H264toHEVC.cfg";
         std::system(cmd.c_str());
         /*H264 Decoder Commandline*/
         cmd = "<path_to_executable>ldecod.exe";
         std::system(cmd.c_str());
         /*EVC Encoder Commandline*/
         cmd = "<path_to_executable>evca_encoder.exe -i <path_to_input_file>test_dec.yuv -
         q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
         config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg";
         std::system(cmd.c_str());
         /*EVC Decoder Commandline*/
```

```
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
}

void HEVC_H264(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8 ";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264_HEVCtoH264.cfg";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe";
        std::system(cmd.c_str());
}

void HEVC_VVC(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8 ";
        std::system(cmd.c_str());
        /*VVC Encoder Commandline*/
        cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_HEVCtoVVC.cfg -
        c <path_to_config_file>akiyo_VVC_HEVCtoVVC.cfg";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VCC.yuv -d 8";
        std::system(cmd.c_str());
}

void HEVC_EVC(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8 ";
        std::system(cmd.c_str());
        /*EVC Encoder Commandline*/
        cmd = "<path_to_executable>evca_encoder.exe -i dec_HEVC.yuv -q 10 -w 176 -h 144 -
        p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_HEVCtoEVC.cfg";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
}

void VVC_H264(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -d 8";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264_VVCtoH264.cfg";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe";
        std::system(cmd.c_str());
}

void VVC_HEVC(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -d 8";
        std::system(cmd.c_str());
        /*HEVC Encoder Commandline*/
        cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_VVCtoHEVC.cfg -
        c <path_to_config_file>akiyo_HEVC_VVCtoHEVC.cfg";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8";
        std::system(cmd.c_str());
}

void VVC_EVC(void) {
        /*VVC Encoder Commandline:*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -d 8";
        std::system(cmd.c_str());
        /*EVC Encoder Commandline*/
        cmd = "<path_to_executable>evca_encoder.exe -i dec_VVC.yuv -q 10 -w 176 -h 144 -
        p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_VVCtoEVC.cfg";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
}

void EVC_H264(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_config_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
      d <path_to_config_file>encoder_main_H264_EVCtoH264.cfg";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe";
        std::system(cmd.c_str());
}

void EVC_HEVC(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_config_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
        /*HEVC Encoder Commandline*/
        cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_EVCtoHEVC.cfg -
        c <path_to_config_file>akiyo_EVC_EVCtoHEVC.cfg";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -d 8";
        std::system(cmd.c_str());
}

void EVC_VVC(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_config_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -o dec_EVC.yuv";
        std::system(cmd.c_str());
        /*VVC Encoder Commandline:*/
        cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_EVCtoVVC.cfg -
        c <path_to_config_file>akiyo_EVC_EVCtoVVC.cfg";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -d 8";
        std::system(cmd.c_str());
}


/*Transcoder.exe application main() function*/
int main(int argc, const int* argv[])
{
        clock_t t;
        /*Choosing the transcoding option on the fly from 12 format change
        possibilities between H.264, HEVC, VVC and EVC*/
        std::cout << "Available options:" << std::endl;
        std::cout << "1. Transcoding from H.264 to HEVC" << std::endl;
        std::cout << "2. Transcoding from H.264 to VVC" << std::endl;
        std::cout << "3. Transcoding from H.264 to EVC" << std::endl;
```

```
std::cout << "4. Transcoding from HEVC to H.264" << std::endl;
std::cout << "5. Transcoding from HEVC to VVC" << std::endl;
std::cout << "6. Transcoding from HEVC to EVC" << std::endl;
std::cout << "7. Transcoding from VVC to H.264" << std::endl;
std::cout << "8. Transcoding from VVC to HEVC" << std::endl;
std::cout << "9. Transcoding from VVC to EVC" << std::endl;
std::cout << "10. Transcoding from EVC to H.264" << std::endl;
std::cout << "11. Transcoding from EVC to HEVC" << std::endl;
std::cout << "12. Transcoding from EVC to VVC" << std::endl;
std::cout << "Type your options(1-12):" << std::endl;
int opt;
std::cin >> opt;

switch (opt) {
case 1:
         t = clock();
     std::cout << "Testing Program for Transcoding from H.264 to HEVC" << std::endl;
         H264_HEVC();
         t = clock() - t;
         break;
case 2:
         t = clock();
      std::cout << "Testing Program for Transcoding from H.264 to VVC" << std::endl;
         H264_VVC();
         t = clock() - t;
         break;
case 3:
         t = clock();
      std::cout << "Testing Program for Transcoding from H.264 to EVC" << std::endl;
         H264_EVC();
         t = clock() - t;
         break;
case 4:
         t = clock();
     std::cout << "Testing Program for Transcoding from HEVC to H.264" << std::endl;
         HEVC_H264();
         t = clock() - t;
         break;
case 5:
         t = clock();
       std::cout << "Testing Program for Transcoding from HEVC to VVC" << std::endl;
         HEVC_VVC();
         t = clock() - t;
         break;
case 6:
         t = clock();
       std::cout << "Testing Program for Transcoding from HEVC to EVC" << std::endl;
         HEVC_EVC();
         t = clock() - t;
         break;
case 7:
         t = clock();
      std::cout << "Testing Program for Transcoding from VVC to H.264" << std::endl;
         VVC_H264();
         t = clock() - t;
         break;
case 8:
         t = clock();
       std::cout << "Testing Program for Transcoding from VVC to HEVC" << std::endl;
         VVC_HEVC();
         t = clock() - t;
```

```
                    break;
            case 9:
                    t = clock();
                   std::cout << "Testing Program for Transcoding from VVC to EVC" << std::endl;
                    VVC_EVC();
                    t = clock() - t;
                    break;
            case 10:
                    t = clock();
                 std::cout << "Testing Program for Transcoding from EVC to H.264" << std::endl;
                    EVC_H264();
                    t = clock() - t;
                    break;
            case 11:
                    t = clock();
                  std::cout << "Testing Program for Transcoding from EVC to HEVC" << std::endl;
                    EVC_HEVC();
                    t = clock() - t;
                    break;
            case 12:
                    t = clock();
                   std::cout << "Testing Program for Transcoding from EVC to VVC" << std::endl;
                    EVC_VVC();
                    t = clock() - t;
                    break;
            }
            /*Time taken to Transcode is calculated in each ‘case’
            statement and printed on the console*/
            double time_taken = ((double)t) / CLOCKS_PER_SEC;
            std::cout << "Time taken to finish transcoding is :" << time_taken << " sec" << std::
    endl;
    }
```

```
/*Heterogeneous and Homogeneous Transcoding for
Bitrate and Frame-rate change for
H.264, HEVC, VVC and EVC codecs*/

#include <iostream>
#include <stdio.h>
#include <time.h>
#include <ctime>

void H264_HEVC(void) {
	/*H264 Encoder Commandline*/
	std::string cmd = "<path_to_executable>lencod.exe -
	d <path_to_config_file>encoder_main_H264_H264toHEVC.cfg > <path_to_output_file>output
	1.txt";
	std::system(cmd.c_str());
	/*H264 Decoder Commandline*/
	cmd = "<path_to_executable>ldecod.exe > <path_to_output_file>output2.txt";
	std::system(cmd.c_str());
	/*HEVC Encoder Commandline*/
	cmd = "<path_to_executable>TAppEncoder.exe -
	c <path_to_config_file>encoder_randomaccess_main_HEVC_H264toHEVC.cfg -
	c <path_to_config_file>/akiyo_H264toHEVCnVVC.cfg > <path_to_output_file>output3.txt";
	std::system(cmd.c_str());
	/*HEVC Decoder Commandline*/
	cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -
	d 8 > <path_to_output_file>output4.txt";
	std::system(cmd.c_str());
}

void H264_VVC(void) {
	/*H264 Encoder Commandline*/
	std::string cmd = "<path_to_executable>lencod.exe -
	d <path_to_config_file>encoder_main_H264_H264toHEVC.cfg > <path_to_output_file>output
	1.txt";
	std::system(cmd.c_str());
	/*H264 Decoder Commandline*/
	cmd = "<path_to_executable>ldecod.exe > <path_to_output_file>output2.txt";
	std::system(cmd.c_str());
	/*VVC Encoder Commandline*/
	cmd = "<path_to_executable>EncoderApp.exe -
	c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_H264toVVC.cfg -
	c <path_to_config_file>akiyo_H264toHEVCnVVC.cfg > <path_to_output_file>output3.txt";
	std::system(cmd.c_str());
	/*VVC Decoder Commandline*/
	cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VCC.yuv -
	d 8 > <path_to_output_file>output4.txt";
	std::system(cmd.c_str());
}

void H264_EVC(void) {
	/*H264 Encoder Commandline*/
	std::string cmd = "<path_to_executable>lencod.exe -
	d <path_to_config_file>encoder_main_H264_H264toHEVC.cfg > <path_to_output_file>output
	1.txt";
	std::system(cmd.c_str());
	/*H264 Decoder Commandline*/
	cmd = "<path_to_executable>ldecod.exe > <path_to_output_file>output2.txt";
	std::system(cmd.c_str());
	/*EVC Encoder Commandline*/
```

```
        cmd = "<path_to_executable>evca_encoder.exe -i <path_to_input_file>test_dec.yuv -
        q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg > <path_to_output_
        file>output3.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void HEVC_H264(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b <path_to_input_file>str_HEVC.bin -
        o dec_HEVC.yuv -d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264_HEVCtoH264.cfg > <path_to_output_file>output
        3.txt";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void HEVC_VVC(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b <path_to_input_file>str_HEVC.bin -
        o dec_HEVC.yuv -d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*VVC Encoder Commandline*/
        cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_HEVCtoVVC.cfg -
        c <path_to_config_file>akiyo_VVC_HEVCtoVVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VCC.yuv -
        d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void HEVC_EVC(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_HEVCtoH264_VVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b <path_to_input_file>str_HEVC.bin -
        o dec_HEVC.yuv -d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*EVC Encoder Commandline*/
        cmd = "<path_to_executable>evca_encoder.exe -i <path_to_input_file>dec_HEVC.yuv -
q 50 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
config <path_to_config_file>encoder_randomaccess_EVC_HEVCtoEVC.cfg > <path_to_output_
file>output3.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
o dec_EVC.yuv > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());

}

void VVC_H264(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
c <path_to_executable>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
c C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\/x64/\Release/\cfg/\/aki
yo_qcif.cfg > C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\output1.txt"
;
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\/x64/\Release/\Deco
derApp.exe -b str_VVC.bin -o dec_VVC.yuv -
d 8 > C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\output2.txt";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\/x64/\Release/\lenc
od.exe -
d C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\/x64/\Release/\cfg/encod
er_main_H264_VVCtoH264.cfg > C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcode
r/\output3.txt";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\/x64/\Release/\ldec
od.exe > C:/\Users/\shrey/\Desktop/\TranscodingBitrate/\Transcoder/\output4.txt";
        std::system(cmd.c_str());
}

void VVC_HEVC(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -
d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*HEVC Encoder Commandline*/
        cmd = "<path_to_executable>TAppEncoder.exe -
c <path_to_config_file>encoder_randomaccess_main_HEVC_VVCtoHEVC.cfg -
c <path_to_config_file>akiyo_HEVC_VVCtoHEVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -
d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}
```

```
void VVC_EVC(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -
        d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*EVC Encoder Commandline*/
        cmd = "<path_to_executable>evca_encoder.exe -i <path_to_input_file>dec_VVC.yuv -
        q 50 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_VVCtoEVC.cfg > <path_to_output_f
        ile>output3.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void EVC_H264(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_input_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg <path_to_output_fi
        le>output1.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_executable>output2.txt";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264_EVCtoH264.cfg > <path_to_output_file>output3
        .txt";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void EVC_HEVC(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_input_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg > <path_to_output_
        file>output1.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*HEVC Encoder Commandline*/
        cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC_EVCtoHEVC.cfg -
        c <path_to_config_file>akiyo_EVC_EVCtoHEVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>TAppDecoder.exe -b str_HEVC.bin -o dec_HEVC.yuv -
        d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void EVC_VVC(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_input_file>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --
        config <path_to_config_file>encoder_randomaccess_EVC_H264toEVC.cfg > <path_to_output_
        file>output1.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*VVC Encoder Commandline:*/
        cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_EVCtoVVC.cfg -
        c <path_to_config_file>akiyo_EVC_EVCtoVVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -
        d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void H264_H264(void) {
        /*H264 Encoder Commandline*/
        std::string cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*H264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe > <path_to_executable>output2.txt";
        std::system(cmd.c_str());
        /*H.264 Encoder Commandline*/
        cmd = "<path_to_executable>lencod.exe -
        d <path_to_config_file>encoder_main_H264toH264.cfg > <path_to_output_file>output3.txt
        ";
        std::system(cmd.c_str());
        /*H.264 Decoder Commandline*/
        cmd = "<path_to_executable>ldecod.exe -
        o test_dec1.yuv > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void HEVC_HEVC(void) {
        /*HEVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>Decoder.exe -b <path_to_input_file>str_HEVC.bin -
        o dec_HEVC.yuv -d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*HEVC Encoder Commandline*/
        cmd = "<path_to_executable>TAppEncoder.exe -
        c <path_to_config_file>encoder_randomaccess_main_HEVCtoHEVC.cfg -
        c <path_to_config_file>akiyo_HEVCtoHEVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*HEVC Decoder Commandline*/
        cmd = "<path_to_executable>Decoder.exe -b <path_to_input_file>str_HEVC1.bin -
        o dec_HEVC1.yuv -d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void VVC_VVC(void) {
        /*VVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVC_VVCtoH264_HEVC_EVC.cfg -
        c <path_to_config_file>akiyo_qcif.cfg > <path_to_output_file>output1.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC.bin -o dec_VVC.yuv -
        d 8 > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*VVC Encoder Commandline:*/
        cmd = "<path_to_executable>EncoderApp.exe -
        c <path_to_config_file>encoder_randomaccess_vtm_RD_VVCtoVVC.cfg -
        c <path_to_config_file>akiyo_VVCtoVVC.cfg > <path_to_output_file>output3.txt";
        std::system(cmd.c_str());
        /*VVC Decoder Commandline*/
        cmd = "<path_to_executable>DecoderApp.exe -b str_VVC1.bin -o dec_VVC1.yuv -
d 8 > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

void EVC_EVC(void) {
        /*EVC Encoder Commandline*/
        std::string cmd = "<path_to_executable>evca_encoder.exe -
        i <path_to_executable>akiyo_qcif.yuv -q 10 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -
        o str_EVC.bin --
        config <path_to_executable>encoder_randomaccess_EVC.cfg > <path_to_output_file>output
        1.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC.bin -
        o dec_EVC.yuv > <path_to_output_file>output2.txt";
        std::system(cmd.c_str());
        /*EVC Encoder Commandline*/
        cmd = "<path_to_executable>evca_encoder.exe -i <path_to_input_file>dec_EVC.yuv -
        q 50 -w 176 -h 144 -p 16 -f 300 -z 20 -d 8 -o str_EVC1.bin --
        config <path_to_config_file>encoder_randomaccess_EVCtoEVC.cfg > <path_to_output_file>
        output3.txt";
        std::system(cmd.c_str());
        /*EVC Decoder Commandline*/
        cmd = "<path_to_executable>evca_decoder.exe -i str_EVC1.bin -
        o dec_EVC1.yuv > <path_to_output_file>output4.txt";
        std::system(cmd.c_str());
}

int main(int argc, const int* argv[])
{
        clock_t t;
        std::cout << "Available options:" << std::endl;
        std::cout << "1. Heterogeneous Bitrate change" << std::endl;
        std::cout << "2. Homogeneous Bitrate change" << std::endl;
```

```
std::cout << "Type your options as 1 or 2: " << std::endl;
int choice;
int opt;
std::cin >> choice;
if (choice == 1)
{

std::cout << "1. Transcoding from H.264 to HEVC for bitrate change" << std::endl;

std::cout << "2. Transcoding from H.264 to VVC for bitrate change" << std::endl;

std::cout << "3. Transcoding from H.264 to EVC for bitrate change" << std::endl;

std::cout << "4. Transcoding from HEVC to H.264 for bitrate change" << std::endl;

std::cout << "5. Transcoding from HEVC to VVC for bitrate change" << std::endl;

std::cout << "6. Transcoding from HEVC to EVC for bitrate change" << std::endl;

std::cout << "7. Transcoding from VVC to H.264 for bitrate change" << std::endl;

std::cout << "8. Transcoding from VVC to HEVC for bitrate change" << std::endl;

std::cout << "9. Transcoding from VVC to EVC for bitrate change" << std::endl;

std::cout << "10. Transcoding from EVC to H.264 for bitrate change" << std::endl;

std::cout << "11. Transcoding from EVC to HEVC for bitrate change" << std::endl;

std::cout << "12. Transcoding from EVC to VVC for bitrate change" << std::endl;
         std::cout << "Type your options(1-12):" << std::endl;
         std::cin >> opt;
}
else if (choice == 2)
{

std::cout << "13. Transcoding from H.264 to H.264 for bitrate change " << std::endl;

std::cout << "14. Transcoding from HEVC to HEVC for bitrate change" << std::endl;

std::cout << "15. Transcoding from VVC to VVC for bitrate change" << std::endl;

std::cout << "16. Transcoding from EVC to EVC for bitrate change" << std::endl;
         std::cout << "Type your options(13-16):" << std::endl;
         std::cin >> opt;
}
else
{
         std::cout << "Check your option!!! " << std::endl;
}
switch (opt) {
case 1:
         t = clock();
         std::cout << "Testing Program for Transcoding from H.264 to HEVC for bitrate
         change" << std::endl;
         H264_HEVC();
         t = clock() - t;
         break;

case 2:
         t = clock();
```

```
        std::cout << "Testing Program for Transcoding from H.264 to VVC for bitrate
        change" << std::endl;
        H264_VVC();
        t = clock() - t;
        break;

case 3:
        t = clock();
        std::cout << "Testing Program for Transcoding from H.264 to EVC for bitrate
        change" << std::endl;
        H264_EVC();
        t = clock() - t;
        break;

case 4:
        t = clock();
        std::cout << "Testing Program for Transcoding from HEVC to H.264 for bitrate
        change" << std::endl;
        HEVC_H264();
        t = clock() - t;
        break;

case 5:
        t = clock();
        std::cout << "Testing Program for Transcoding from HEVC to VVC for bitrate c
        hange" <<std::endl;
        HEVC_VVC();
        t = clock() - t;
        break;

case 6:
        t = clock();
        std::cout << "Testing Program for Transcoding from HEVC to EVC for bitrate c
        hange" <<std::endl;
        HEVC_EVC();
        t = clock() - t;
        break;

case 7:
        t = clock();
        std::cout << "Testing Program for Transcoding from VVC to H.264 for bitrate
        change" << std::endl;
        VVC_H264();
        t = clock() - t;
        break;

case 8:
        t = clock();
        std::cout << "Testing Program for Transcoding from VVC to HEVC for bitrate c
        hange" <<std::endl;
        VVC_HEVC();
        t = clock() - t;
        break;

case 9:
        t = clock();
        std::cout << "Testing Program for Transcoding from VVC to EVC for bitrate ch
        ange" << std::endl;
        VVC_EVC();
        t = clock() - t;
        break;
        case 10:
                t = clock();
                std::cout << "Testing Program for Transcoding from EVC to H.264 for bitrate
                change" << std::endl;
                EVC_H264();
                t = clock() - t;
                break;

        case 11:
                t = clock();
                std::cout << "Testing Program for Transcoding from EVC to HEVC for bitrate c
                hange" <<std::endl;
                EVC_HEVC();
                t = clock() - t;
                break;

        case 12:
                t = clock();
                std::cout << "Testing Program for Transcoding from EVC to VVC for bitrate ch
                ange" << std::endl;
                EVC_VVC();
                t = clock() - t;
                break;

        case 13:
                t = clock();
                std::cout << "Testing Program for Transcoding from H.264 to H.264 for bitrat
                e change:" << std::endl;
                H264_H264();
                t = clock() - t;
                break;

        case 14:
                t = clock();
                std::cout << "Testing Program for Transcoding from HEVC to HEVC for bitrate
                change:" << std::endl;
                HEVC_HEVC();
                t = clock() - t;
                break;

        case 15:
                t = clock();
                std::count << "Testing Program for Transcoding from VVC to VVC for bitrate
                change: " << std::endl;
                VVC_VVC();
                t = clock() - t;
                break;

        case 16:
                t = clock();
                std::cout << "Testing Program for Transcoding from EVC to EVC for bitrate
                change:" << std::endl;
                EVC_EVC();
                t = clock() - t;
                break;
        }
        double time_taken = ((double)t) / CLOCKS_PER_SEC;
        std::cout << "Time taken to finish transcoding is :" << time_taken << " sec" << std::
endl;
}
```