İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

COMPRESSED DOMAIN H.264/AVC VIDEO EDITING

M.Sc. Thesis by

Ertuğrul DOĞAN, B.Sc.

Department : Electronics and Telecommunications Engineering

Programme: Telecommunications Engineering

JUNE 2008

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

COMPRESSED DOMAIN H.264/AVC VIDEO EDITING

M.Sc. Thesis by
Ertuğrul DOĞAN, B.Sc.
(504051308)

Date of submission : 5 May 2008

Date of defence examination: 11 June 2008

Supervisor (Chairman): Prof.Dr. Melih PAZARCI

Members of the Examining Committee Prof.Dr. Bilge GÜNSEL

Assoc. Prof.Dr. Uluğ BAYAZIT

JUNE 2008

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

H.264/AVC SIKIŞMIŞ UZAYINDA GÖRÜNTÜ DÜZENLEME

YÜKSEK LİSANS TEZİ
Müh. Ertuğrul DOĞAN
(504051308)

Tezin Enstitüye Verildiği Tarih : 5 Mayıs 2008
Tezin Savunulduğu Tarih : 11 Haziran 2008

Tez Danışmanı : Prof.Dr. Melih PAZARCI

Diğer Jüri Üyeleri Prof.Dr. Bilge GÜNSEL

Doç.Dr. Uluğ BAYAZIT

HAZİRAN 2008

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Prof. Dr. Melih Pazarcı, for his valuable advices, comments and guidance throughout this thesis.

Also I would like to give special thanks to my family for their generous understanding and moral support.

June, 2008 Ertuğrul DOĞAN

İstanbul

CONTENTS

ABBREVIATIONS

AVC	: Advanced Video Coding
ITU	: International Telecommunication Union
ISO	: International Standardization for Standardization
JVT	: Joint Video Team
VCEG	: Video Coding Experts Group
MPEG	: Moving Pictures Experts Group
NAL	: Network Abstraction Layer
VCL	: Video Coding Layer
DCT	: Discrete Cosine Transform
CABAC	: Context Adaptive Binary Arithmetic Coding
CAVLC	: Context Adaptive Variable Length Coding)
QCIF	: Quarter Common Intermediate Format
MMS	: Multimedia Messaging Service
RBSP	: Raw Byte Sequence Payload
RTP	: Real Time Transport Protocol
IDR	: Instantaneous Decoding Refresh
MB	: Macro Block
MV	: Motion Vector
GOP	: Group of Pictures
CDP	: Compressed Domain Processing
QP	: Quantization Parameter
CBR	: Constant Bit Rate
VBR	: Variable Bit Rate

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As compression becomes the heart of digital video applications, H.264/AVC has become widely used in various applications and services because of its high coding efficiency. With the spread of H.264/AVC contents, the need for editing video has substantially increased. It has been strongly required to edit the H.264/AVC coded contents with less computational complexity since the compression of H.264/AVC requires far more processing power than other existing formats such as MPEG-4, and MPEG-2. So, compressed domain editing that has benefits like savings for memory, processing power and delay (due to decoding and then re-encoding the edited video back to the compressed domain), and the preservation of picture quality by avoiding lossy decode/re-encode chain is used.

In this thesis, two methods for compressed domain editing of H.264/AVC are proposed: there is a fast editing method provided that the editing is performed in GOP-based. Since there is no dependency between consecutive GOPs when closed GOP, cut and splice operations are done easily without decoding the originally coded stream.

Secondly, frame-accurate editing is proposed. In frame accurate editing, cut points can be seleceted inside a GOP and it is required that every frame contained in the resultant stream. It consists of re-encoding method with tandem connection of frame based decoder and encoder. When cutting out a segment of H.264/AVC video at arbitrary location, it is needed to decode and re-encode only the frames that are out of the GOP boundary at the beginning or ending part of the segments. The newly created GOPs at the two ends may have a different size, but the segment is still conformable to the standard format.

These two methods are used on various contents and output streams produced. The streams are verified with H.264/AVC reference decoder and the compliances of the two methods to the standard are proven.

ÖZET

Görüntü sıkıştırma, sayısal görüntü uygulamalarında önemini arttırdıkça, H264/AVC sayısal kodlama standartı da yüksek kodlama kabiliyeti sebebiyle değişik uygulamalarda ve servislerde yaygınlaşmaya başlamıştır. H.264/AVC standartı ile kodlanan içerikler çoğaldıkça, görüntü düzeltmeye duyulan ihtiyaç da artış göstermektedir. H.264/AVC ile kodlanan bu içerikleri daha az hesaplama gücü harcayarak düzenlemek de H.264/AVC satndartının MPEG-4 ve MPEG-2 gibi standartlara göre yoğun işlem gereketirmesi sebebiyle kaçınılmaz hale gelmiştir. Bunun sonucu olarak H.264/AVC standartı ile kodlanan içerikler, daha az işlem gücü ve hafıza kullanan, düzenlemeyi daha çabuk yapan, ve görüntü kalitesini kodçözme/yeniden kodlama gibi adımlar içermediği için koruyan sıkışmış uzayda görüntü düzenleme teknikleriyle düzenlenmektedirler.

Bu tez çalışması kapsamında, iki adet sıkışmış uzayda H.264/AVC düzenleme tekniği ele alınmıştır: ilki GOP tabanlı çalıştığı için hızlı bir yöntem olan GOP-Tabanlı düzenleme yöntemidir. H.264/AVC standartında kapalı GOP yönteminde GOP lar arasında bir bağ bulunmadığı için kesme ve yapıştırma işlemleri, orjinal kodlanmış içeriği değiştirmeden rahatlıkla gerçekleştirilebilir.

İkinci yöntem olarak tam-çerçeve tabanlı yöntem ele alınmıştır. Bu yöntemde kesme noktaları GOP içinde seçilebilir ve kodlanmış çıkış görüntüsünde seçilen bütün çerçeveler yer almalıdır. Bu yöntemde ardısıra bağlanmış kodçözücü/kodlayıcı kullanılmıştır. Çıkışta üretilen kodlanmış GOP lar giriş GOP larına göre farklı boyutlarda olabilir fakat H.264/AVC standartına uygun olmak zorundadırlar.

Her iki yöntemle de değişik durumlar için sıkıştırılmış uzayda görüntü düzenleme yapılmış, ve oluşturulan kodlanmış çıktıların H.264/AVC standartına uygunluğu referans kodçözücü yardımıyla kanıtlanmıştır.

1. INTRODUCTION

The increasing demand to video data into telecommunications services, the corporate environment, the entertainment industry, and at home has made digital video technology a necessity. There is a problem that still image and digital video data rates are very large. Data rates of high magnitude consume a lot of the bandwidth, storage and computing resources in the typical personal computer. For this reason, video compression standards have been developed to eliminate picture redundancy, allowing video information to be transmitted and stored in a compact and efficient way.

H.264/AVC is a video compression standard jointly developed by ITU-T VCEG and ISO/IEC MPEG standards committees. The standard is becoming more popular as it promises much higher compression than earlier video coding standards. The standard provides flexibilities in coding and organization of data which enable efficient error resilience. The increased coding efficiency offers new application areas. As expected, the increase in compression efficiency and flexibility come at the expense of increase in complexity, which is a fact that must be overcome.

As the importance of compression increases in digital video applications, the need for editing video in the compressed domain has substantially increased. Video editing is a natural and necessary operation that is most commonly employed by users for finalizing and organizing their video content. The benefits of video editing in the compressed domain are abundant. While the obvious ones include savings for memory, processing power and delay (due to decoding and then re-encoding the edited video back to the compressed domain), the most significant benefit is the preservation of picture quality by avoiding the lossy decode-encode chain [9].

There are several works in literature about compressed domain editing. These are generally on MPEG-2 video coding standard. In [2], picture-type conversion method

in compressed domain for MPEG-2 content for trimming at arbitrary frame is explained.

In [9], methods to concatenate MPEG-2 segments while maintaining video buffer verifier requirements are proposed. Also methods to overcome buffer underflow/overflow problems in MPEG-2 video coding standard are proposed in [11].

Techniques about editing of H.263 and MPEG-4 video coding standards are proposed in [8] and [10]. These techniques include not only cutting and splicing operations, but also contain fading and blending on MPEG-4 and H.263 coded videos.

In [7], a fast frame accurate editing method of H.264/AVC is proposed. But this work based on baseline profile and the problems when occur in the existence of B frames are ignored.

2. OVERVIEW OF H.264/AVC

H.264 is the newest video coding standard, developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) as the product of a partnership effort known as the Joint Video Team (JVT).

The main goals of the H.264/AVC standardization effort have been enhanced compression performance and provision of a "network-friendly" video representation addressing "conversational" (video telephony) and "non-conversational" (storage, broadcast, or streaming) applications.

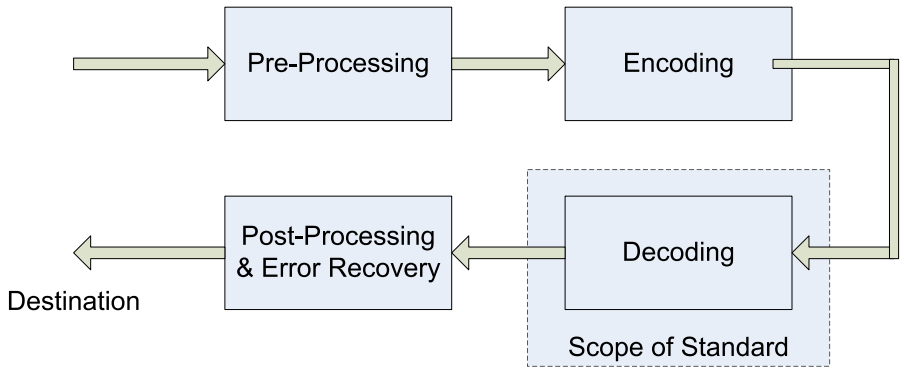

Figure 2.1: Scope of Video Coding Standardization

The scope of the standardization is illustrated in Figure 2.1, which shows the typical video encoding/decoding chain (excluding the transport or storage of the video signal). In all ITU-T and ISO/IEC video coding standards, only the central decoder is standardized, by imposing restrictions on the bitstream and syntax, and defining the decoding process of the syntax elements. Every decoder conforming to the standard will produce similar output when given an encoded bitstream that conforms to the constraints of the standard.

An H.264 video stream is organized in discrete packets, called “NAL units” (Network Abstraction Layer units). Each of these packets can contain a part of a slice and there may be one or more NAL units per slice. But not all NAL units contain

slice data; there are also NAL unit types for other purposes, such as signalling, headers and additional data.

The slices contain a part of a video frame. In normal bitstreams, each frame consists of a single slice whose data is stored in a single NAL unit. Nevertheless, the possibility to spread frames over an almost arbitrary number of NAL units can be useful if the stream is transmitted over an error-prone medium: The decoder may resynchronize after each NAL unit instead of skipping a whole frame if a single error occurs.

H.264 also supports optional interlaced encoding. In this encoding mode, a frame is split into two fields. Fields may be encoded using spacial or temporal interleaving.

To encode color images, H.264 uses the YCbCr color space like its predecessors, separating the image into luma (or "luminance", brightness) and chroma (or "chrominance", color) planes. It is, however, fixed at 4:2:0 subsampling, i.e. the chroma channels each have half the resolution of the luma channel.

2.1 Comparison of H.264/AVC and Prior Standards

As described in [3], H.264/AVC uses translational block based motion compensation and transform based residual coding as in prior coding standards. However, H.264/AVC promises significant differences in details which are stated in Table 2.1.

Table 2.1: Comparison of H.264/AVC and Prior Compression Standards

	H.264	MPEG-1/2/4, H.261/3
Prediction in space domain	- Spatial prediction - Encode the prediction modes (Use predictive coding if 4x4 modes are used)	- No spatial prediction
Transform	- Integer Transform	- 8x8 Discrete Cosine Transform (DCT) for pixel values
Quantization	- Quantization including scaling	- Quantization
Prediction in	- No coefficient prediction	- Coefficient prediction (for DC values in MPEG-2 and AC

frequency domain		values in the first row and column in MPEG-4)
References	- Permits up to 15 (2 mostly used) reference pictures - Bi-predictive B-slices - A P-slice may reference a picture that has B-slices - Supports explicit weighting coefficients and (a+b)/2 type	- A P-slice references only one I-picture - Bi-directional B-slices - Only permit (a+b)/2 type prediction weighting
Block Sizes	- Tree-structured (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4)	- Either 16x16 or 8x8
Motion Estimation	- half or ¼-pixel accuracy - 6-point interpolation for half-pixel and 2-point linear interpolation for ¼-pixel	- MPEG2 permits half-pixel accuracy and MPEG4 permits ¼-pixel accuracy - 2-point linear interpolation

Relative to prior video coding methods, some highlighted features of the H.264/AVC design that enable enhanced coding efficiency include the following enhancements of the ability to predict the values of the content of a picture to be encoded:

• Variable block-size motion compensation with small block sizes: H.264/AVC supports more flexibility in the selection of motion compensation block sizes and shapes than any previous standard, with a minimum luma motion compensation block size as small as 4x4 [3].

• Quarter-sample-accurate motion compensation: Most prior standards enable half sample motion vector accuracy at most. The new design improves up on this by adding quarter sample motion vector accuracy, as first found in an advanced profile of the MPEG-4 Visual (Part 2) standard, but further reduces the complexity of the interpolation processing compared to the prior design [3].

• Multiple reference picture motion compensation: H.264/AVC extends upon the enhanced reference picture selection technique found in H.263++ to enable efficient coding by allowing an encoder to select among a larger number of pictures (15) that

have been decoded and stored in the decoder. The same extension of referencing capability is also applied to motion-compensated bi-prediction, which is restricted in MPEG-2 to using two specific pictures only (one of these being the previous I or P picture in display order and the other being the next I or P picture in display order) [3].

• Decoupling of referencing order from display order: In prior standards, there was a strict dependency between the ordering of pictures for motion compensation referencing purposes and the ordering of pictures for display purposes. In H.264/AVC, these restrictions are largely removed, allowing the encoder to choose the ordering of pictures for referencing and display purposes with a high degree of flexibility constrained only by a total memory capacity bound imposed to ensure decoding ability [3].

• Weighted prediction: A new innovation in H.264/AVC allows the motion-compensated prediction signal to be weighted and offset by amounts specified by the encoder. This can dramatically improve coding efficiency for scenes containing fades, and can be used flexibly for other purposes as well [3].

• Directional spatial prediction for intra coding: A new technique of extrapolating the edges of the previously-decoded parts of the current picture is applied in regions of pictures that are coded as intra (i.e., coded without reference to the content of some other picture). This improves the quality of the prediction signal, and also allows prediction from neighboring areas that were not coded using intra coding [3].

• Small block-size transform: All major prior video coding standards used a transform block size of 8x8, while the new H.264/AVC design is based primarily on a 4x4 transform. This allows the encoder to represent signals in a more locally adaptive fashion, which reduces artifacts known colloquially as "ringing". (The smaller block size is also justified partly by the advances in the ability to better predict the content of the video using the techniques noted above, and by the need to provide transform regions with boundaries that correspond to those of the smallest prediction region.) [3]

• Exact-match inverse transform: In previous video coding standards, the transform used for representing the video was generally specified only within an error tolerance

bound, due to the impracticality of obtaining an exact match to the ideal specified inverse transform. As a result, each decoder design would produce slightly different decoded video, causing a "drift" between encoder and decoder representation of the video and reducing effective video quality [3].

• Arithmetic entropy coding: An advanced entropy coding method known as arithmetic coding is included in H.264/AVC. While arithmetic coding was previously found as an optional feature of H.263, a more effective use of this technique is found in H.264/AVC to create a very powerful entropy coding method known as CABAC (context-adaptive binary arithmetic coding) [3].

• Context-adaptive entropy coding: The two entropy coding methods applied in H.264/AVC, termed CAVLC (context-adaptive variable-length coding) and CABAC, both use context-based adaptivity to improve performance relative to prior standard designs [3].

• In-the-loop deblocking filtering: Block-based video coding produces artifacts known as blocking artifacts. These can originate from both the prediction and residual difference coding stages of the decoding process. Application of an adaptive deblocking filter is a well-known method of improving the resulting video quality, and when designed well, this can improve both objective and subjective video quality. Building further on a concept from an optional feature of H.263+, the deblocking filter in the H.264/AVC design is brought within the motion compensated prediction loop, so that this improvement in quality can be used in inter-picture prediction to improve the ability to predict other pictures as well [3].

Robustness to data errors/losses and flexibility for operation over a variety of network environments is enabled by a number of design aspects new to the H.264/AVC standard including the following highlighted features [3].

• Parameter set structure: The parameter set design provides for robust and efficient conveyance header information. As the loss of a few key bits of information (such as sequence header or picture header information) could have a severe negative impact on the decoding process when using prior standards, this key information was separated for handling in a more flexible and specialized manner in the H.264/AVC design [3].

• NAL unit syntax structure: Each syntax structure in H.264/AVC is placed into a logical data packet called a NAL unit. Rather than forcing a specific bitstream interface to the system as in prior video coding standards, the NAL unit syntax structure allows greater customization of the method of carrying the video content in a manner appropriate for each specific network [3].

• Flexible slice size: Unlike the rigid slice structure found in MPEG-2 (which reduces coding efficiency by increasing the quantity of header data and decreasing the effectiveness of prediction), slice sizes in H.264/AVC are highly flexible, as was the case earlier in MPEG-1 [3].

• Flexible macroblock ordering (FMO): A new ability to partition the picture into regions called slice groups has been developed, with each slice becoming an independently-decodable subset of a slice group. When used effectively, flexible macroblock ordering can significantly enhance robustness to data losses by managing the spatial relationship between the regions that are coded in each slice [3].

• Arbitrary slice ordering (ASO): Since each slice of a coded picture can be decoded independently of the other slices of the picture, the H.264/AVC design enables sending and receiving the slices of the picture in any order relative to each other. This capability, first found in an optional part of H.263+, can improve end-to-end delay in real-time applications, particularly when used on networks having out-of-order delivery behavior (e.g., internet protocol networks) [3].

• Data Partitioning: Since some coded information for representation of each region (e.g., motion vectors) is more important or more valuable than other information for purposes of representing the video content, H.264/AVC allows the syntax of each slice to be separated into up to three different partitions for transmission, depending on a categorization of syntax elements. Here the design is simplified by having a single syntax with partitioning of that same syntax controlled by a specified categorization of syntax elements [3].

• SP/SI synchronization/switching pictures: The H.264/AVC design includes a new feature consisting of picture types that allow exact synchronization of the decoding process of some decoders with an ongoing video stream produced by other decoders without penalizing all decoders with the loss of efficiency resulting from ending an I

picture. This can enable switching a decoder between representations of the video content that used different data rates, recovery from data losses or errors, as well as enabling trick modes such as fast-forward, fast-reverse, etc [3].

These major differences results make H.264/AVC standard to achieve 50% average coding gain over MPEG-2, and 47% average coding gain over H.263 baseline encoders [4]. In Figure 2.2, visual quality of H.264 can be seen clearly. In this case, input is raw video encoded at constant bitrate (QCIF, 30 fps, 100 kbit/s).

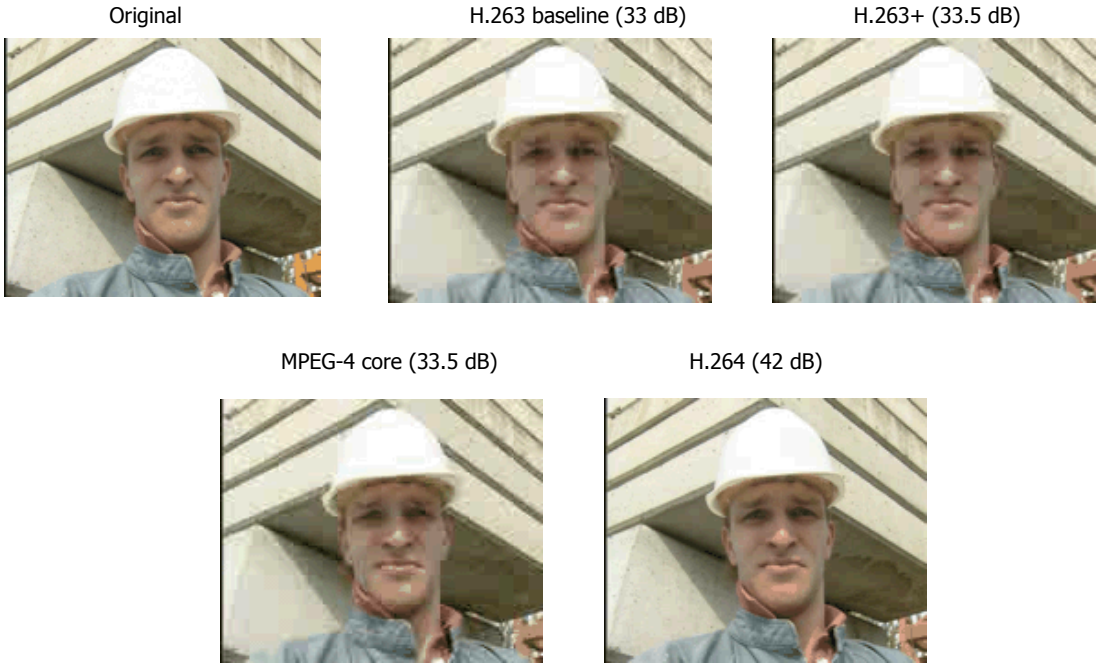

Figure 2.2: Visual Comparison of H.264/AVC and Prior Standards

2.2 H.264/AVC Profiles and Levels

The standard includes the following seven sets of capabilities, which are referred to as profiles, targeting specific classes of applications:

- Baseline Profile (BP): Primarily for lower-cost applications with limited computing resources, this profile is used widely in videoconferencing and mobile applications.

- Main Profile (MP): Originally intended as the mainstream consumer profile for broadcast and storage applications, the importance of this profile faded when the High Profile was developed for those applications.

- Extended Profile (XP): Intended as the streaming video profile, this profile has relatively high compression capability and some extra tricks for robustness to data losses and server stream switching.

- High Profile (HiP): The primary profile for broadcast and disc storage applications, particularly for high-definition television applications (this is the profile adopted into HD-DVD and Blu-ray Disc).

- High 10 Profile (Hi10P): Going beyond today's mainstream consumer product capabilities, this profile builds on top of the High Profile adding support for up to 10 bits per sample of decoded picture precision.

- High 4:2:2 Profile (Hi422P): Primarily targeting professional applications that use interlaced video, this profile builds on top of the High 10 Profile dding support for the 4:2:2 chroma subsampling format while using up to 10 bits per sample of decoded picture precision.

- High 4:4:4 Predictive Profile (Hi444PP): This profile builds on top of the High 4:2:2 Profile—supporting up to 4:4:4 chroma sampling, up to 14 bits per sample, and additionally supporting efficient lossless region coding and the coding of each picture as three separate color planes.

Table 2.2 shows the relationship between the three Profiles and the coding tools supported by H.264/AVC standard.

Performance limits for H.264/AVC encoders/decoders are defined by a set of Levels, each placing limits on parameters such as sample processing rate, picture size, coded bitrate and memory requirements. Table 2.3 shows some of the levels specified in the standard (for all of the levels, see [1]).

Table 2.2: H.264/AVC Profiles

	Baseline	Extended	Main	High	High 10	High 4:2:2	High 4:4:4 Predictive
I and P Slices	Yes	Yes	Yes	Yes	Yes	Yes	Yes
B Slices	No	Yes	Yes	Yes	Yes	Yes	Yes
SI and SP Slices	No	Yes	No	No	No	No	No
Multiple Reference Frames	Yes	Yes	Yes	Yes	Yes	Yes	Yes
In-Loop Deblocking Filter	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CAVLC Entropy Coding	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CABAC Entropy Coding	No	No	Yes	Yes	Yes	Yes	Yes
Flexible Macroblock Ordering (FMO)	Yes	Yes	No	No	No	No	No
Arbitrary Slice Ordering (ASO)	Yes	Yes	No	No	No	No	No
Redundant Slices (RS)	Yes	Yes	No	No	No	No	No
Data Partitioning	No	Yes	No	No	No	No	No
Interlaced Coding (PicAFF, MBAFF)	No	Yes	Yes	Yes	Yes	Yes	Yes
4:2:0 Chroma Format	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Monochrome Video Format (4:0:0)	No	No	No	Yes	Yes	Yes	Yes
4:2:2 Chroma Format	No	No	No	No	No	Yes	Yes
4:4:4 Chroma Format	No	No	No	No	No	No	Yes
8 Bit Sample Depth	Yes	Yes	Yes	Yes	Yes	Yes	Yes
9 and 10 Bit Sample Depth	No	No	No	No	Yes	Yes	Yes
11 to 14 Bit Sample Depth	No	No	No	No	No	No	Yes
8x8 vs. 4x4 Transform Adaptivity	No	No	No	Yes	Yes	Yes	Yes
Quantization Scaling Matrices	No	No	No	Yes	Yes	Yes	Yes
Separate Cb and Cr QP control	No	No	No	Yes	Yes	Yes	Yes
Separate Color Plane Coding	No	No	No	No	No	No	Yes
P. Lossless Coding	No	No	No	No	No	No	Yes

Table 2.3: Some of H.264/AVC Levels

Level number	Max MBs per second	Max frame size (MBs)	Max video bit rate (VCL) for BP, XP and MP	Max video bit rate (VCL) for HP	Max video bit rate (VCL) for Hi10P	Max video bit rate (VCL) for Hi422P and Hi444PP	Examples for high resolution @ frame rate
1.1	3000	396	192 kbit/s	240 kbit/s	576 kbit/s	768 kbit/s	176x144@30.3 320x240@10.0 352x288@7.5
1.3	11880	396	768 kbit/s	960 kbit/s	2304 kbit/s	3072 kbit/s	320x240@36.0 352x288@30.0
2	11880	396	2 Mbit/s	2.5 Mbit/s	6 Mbit/s	8 Mbit/s	320x240@36.0 352x288@30.0
2.1	19800	792	4 Mbit/s	5 Mbit/s	12 Mbit/s	16 Mbit/s	352x480@30.0 352x576@25.0
3	40500	1620	10 Mbit/s	12.5 Mbit/s	30 Mbit/s	40 Mbit/s	352x480@61.4 352x576@51.1 720x480@30.0 720x576@25.0
3.1	108000	3600	14 Mbit/s	17.5 Mbit/s	42 Mbit/s	56 Mbit/s	720x480@80.0 720x576@66.7 1280x720@30.0
3.2	216000	5120	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@60.0 1280x1024@42.2
4	245760	8192	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@68.3 1920x1088@30.1 2048x1024@30.0
5	589824	22080	135 Mbit/s	168.75 Mbit/s	405 Mbit/s	540 Mbit/s	1920x1088@72.3 (13) 2048x1024@72.0 (13) 2048x1088@67.8 (12) 2560x1920@30.7 (5) 3680x1536/26.7 (5)

3. H.264/AVC CODED DATA STRUCTURE

H.264/AVC is designed in two layers: a video coding layer (VCL), that is designed to represent the video content, and a network adaptation layer (NAL), which provides header information and VCL representation of the video for transfer and storage, as shown in Figure 3.1. The purpose of separately specifying the VCL and NAL is to distinguish between coding-specific features (at the VCL) and transport-specific features (at the NAL).

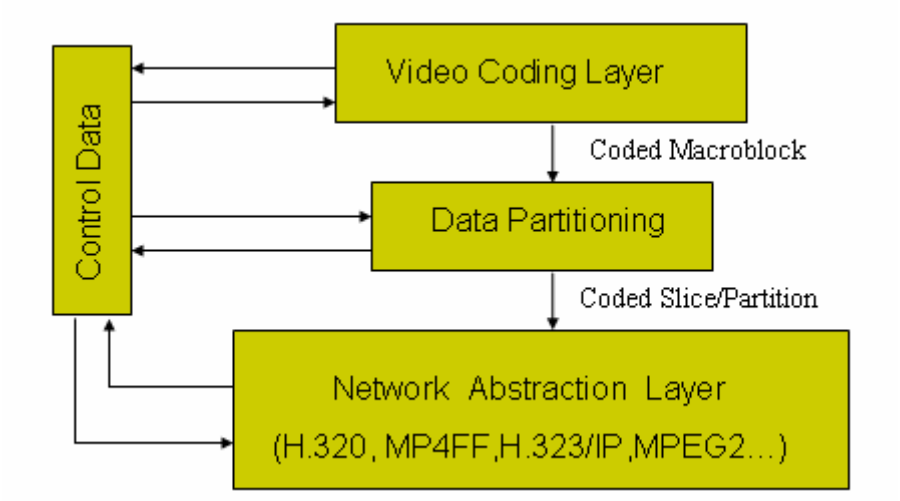

Figure 3.1: Structure of H.264/AVC Video Encoder

3.1 Network Abstraction Layer

The network abstraction layer (NAL) is designed in order to provide "network friendliness" to enable simple and effective customization of the use of the VCL for a broad variety of systems. The NAL facilitates the ability to map H.264/AVC VCL data to transport layers such as:

• RTP/IP for any kind of real-time wire-line and wireless internet services

• File formats, e.g. ISO MP4 for storage and MMS

• H.32X for wireline and wireless conversational services

• MPEG-2 systems for broadcasting services, etc.

The output of the encoding process is VCL data (a sequence of bits representing the coded video data) which are mapped to NAL units prior to transmission or storage. Each NAL unit contains a Raw Byte Sequence Payload (RBSP), a set of data corresponding to coded video data or header information. A coded video sequence is represented by a sequence of NAL units that can be transmitted over a packet-based network or a bitstream transmission link or stored in a file as shown in Figure 3.2.

Figure 3.2: Sequence of NAL Units

The NAL unit structure definition specifies a generic format for use in both packet-oriented and bitstream-oriented transport systems, and a series of NAL units generated by an encoder is referred to as a NAL unit stream.

3.1.1 VCL and non-VCL NAL Units

NAL units can be VCL or non-VCL NAL units. The VCL NAL units contain the data that represents the values of the samples in the video pictures, and the non-VCL NAL units contain any associated additional information such as parameter sets (important header data that can apply to a large number of VCL NAL units) and supplemental enhancement information (timing information and other supplemental data that may enhance usability of the decoded video signal but are not necessary for decoding the values of the samples in the video pictures).

3.1.2 Parameter Sets

A parameter set is supposed to contain information that is expected to rarely change and offers the decoding of a large number of VCL NAL units. There are two types of parameter sets; sequence parameter sets, which apply to a series of consecutive coded video pictures called a coded video sequence, and picture parameter sets,

which apply to the decoding of one or more individual pictures within a coded video sequence.

The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded representations of the values of the samples in the video pictures. Each VCL NAL unit contains an identifier that refers to the content of the relevant picture parameter set, and each picture parameter set contains an identifier that refers to the content of the relevant sequence parameter set. In this manner, a small amount of data (the identifier) can be used to refer to a larger amount of information (the parameter set) without repeating that information within each VCL NAL unit.

3.1.3 NAL Units in Transport Systems

Some systems (e.g., H.320 and MPEG-2 | H.222.0 systems) require delivery of the entire or partial NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns within the coded data itself. For use in such systems, the H.264/AVC specification defines a byte stream format. In the byte stream format, each NAL unit is prefixed by a specific pattern of three bytes called a start code prefix. The boundaries of the NAL unit can then be identified by searching the coded data for the unique start code prefix pattern. The use of emulation prevention bytes guarantees that start code prefixes are unique identifiers of the start of a new NAL unit.

In other systems (e.g., internet protocol / RTP systems), the coded data is carried in packets that are framed by the system transport protocol, and identification of the boundaries of NAL units within the packets can be established without use of start code prefix patterns. In such systems, the inclusion of start code prefixes in the data would be a waste of data carrying capacity, so instead the NAL units can be carried in data packets without start code prefixes.

3.1.4 Access Units

A set of NAL units in a specified form is referred to as an access unit. The decoding of each access unit results in one decoded picture.

Each access unit contains a set of VCL NAL units that together compose a primary coded picture. It may also be prefixed with an access unit delimiter to aid in locating the start of the access unit. Some supplemental enhancement information (SEI) containing data such as picture timing information may also precede the primary coded picture.

The primary coded picture consists of a set of VCL NAL units consisting of slices or slice data partitions that represent the samples of the video picture.

3.1.5 Coded Video Sequences

A coded video sequence consists of a series of access units that are sequential in the NAL unit stream and use only one sequence parameter set. Each coded video sequence can be decoded independently of any other coded video sequence.

At the beginning of a coded video sequence is an instantaneous decoding refresh (IDR) access unit. An IDR access unit contains an intra picture – a coded picture that can be decoded without decoding any previous pictures in the NAL unit stream, and the presence of an IDR access unit indicates that no subsequent picture in the stream will require reference to pictures prior to the intra picture it contains in order to be decoded. A NAL unit stream may contain one or more coded video sequences.

3.2 Video Coding Layer

As in all prior video coding standards, the VCL design follows the so-called block based hybrid video coding approach (as depicted in Figure 3.3) , in which each coded picture is represented in block shaped units of associated luma and chroma samples called macroblocks. The basic source-coding algorithm is a hybrid of inter-picture prediction to exploit temporal statistical dependencies and transform coding of the

prediction residual to exploit spatial statistical dependencies. The encoding process also includes decoding process (except for entropy decoding) since the motion estimation has to use the same reconstructed reference picture with the decoder.

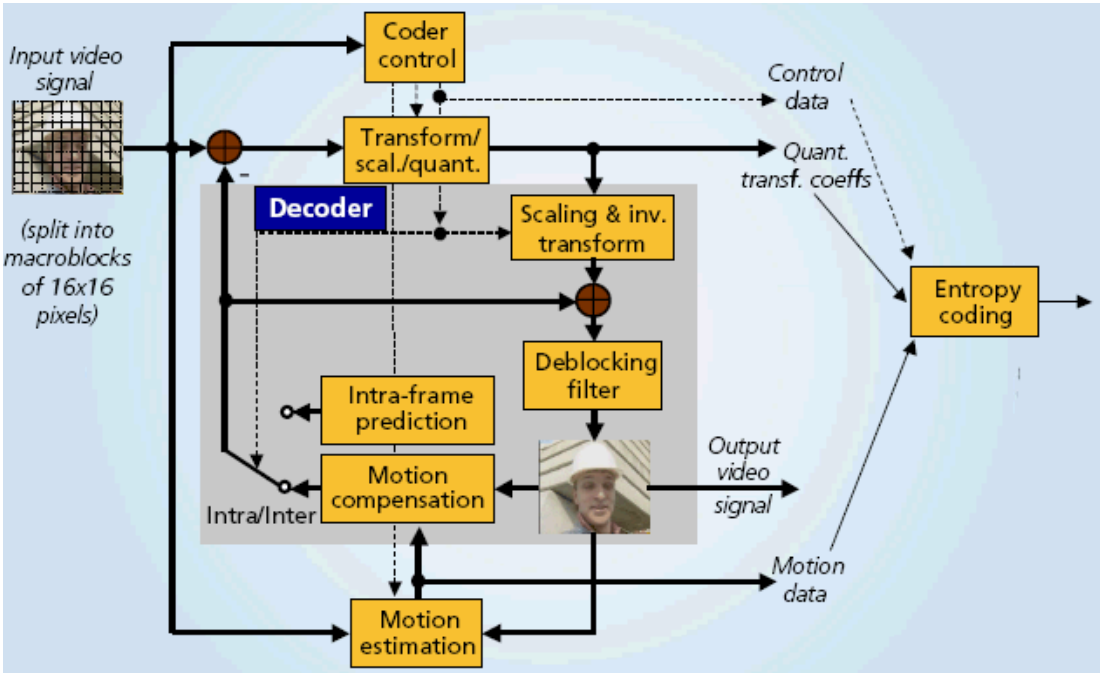

Figure 3.3: Basic coding structure of H.264/AVC for a macroblock

A coded video sequence in H.264/AVC consists of a sequence of coded pictures. A coded picture in can represent either an entire frame or a single field, as was also the case for MPEG-2 video.

3.2.1 Macroblocks, Slices, and Slice Groups

A picture is partitioned into fixed-size macroblocks that each covers a rectangular picture area of 16x16 samples of the luma component and 8x8 samples of each of the two chroma components. This partitioning into macroblocks has been adopted into all previous ITU-T and ISO/IEC video coding standards since H.261. Macroblocks are the basic building blocks of the standard for which the decoding process is specified.

Slices are a sequence of macroblocks which are processed in the order of a raster scan. A picture maybe split into one or several slices as shown in Figure 3.4.

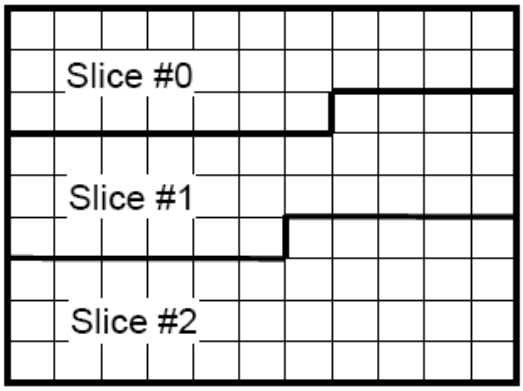

Figure 3.4: Subdivision of a picture into slices

A picture is therefore a collection of one or more slices in H.264/AVC. Slices are self-contained in the sense that given 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 use of data from other slices provided that utilized reference pictures are identical at encoder and decoder. Some information from other slices maybe needed to apply the deblocking filter across slice boundaries.

Each slice can be coded using different coding types as follows:

• I slice: A slice in which all macroblocks of the slice are coded using intra prediction.

• P slice: In addition to the coding types of I slices, some macroblocks of the P slice can also be coded using inter prediction with at most one motion compensated prediction signal per prediction block.

• B slice: In addition to the coding types available in a P slice, some macroblocks of the B slice can also be coded using inter prediction with two motion compensated prediction signals per prediction block.

The above three coding types are very similar to those in previous video coding standards with the exception of the use of reference pictures as described below. The following two coding types for slices are new:

• SP slice: A so-called switching P slice that is coded such that efficient switching between different precoded pictures becomes possible.

• SI slice: A so-called switching I slice that allows an exact match of a macroblock in an SP slice for random access and error recovery purposes. Encoding and decoding process for macroblocks.

3.2.2 Encoding and Decoding Process for Macroblocks

All luma and chroma samples of a macroblock are either spatially or temporally predicted, and the resulting prediction residual is encoded using transform coding. For transform coding purposes, each color component of the prediction residual signal is subdivided into smaller 4x4 blocks. Each block is transformed using an integer transform, and the transform coefficients are quantized and encoded using entropy coding methods.

Figure 2.5 shows block diagram of the video coding layer for a macroblock. The input video signal is split into macroblocks, the association of macroblocks to slice groups and slices is selected, and then each macroblock of each slice is processed as shown.

Every coded macroblock in an H.264 slice is predicted from previously-encoded data. This prediction is subtracted from the current macroblock and the result of the subtraction (residual) is compressed and transmitted to the decoder, together with information required for the decoder to repeat the prediction process (motion vector(s), prediction mode, etc.).

The decoder creates an identical prediction and adds this to the decoded residual block. The encoder bases its prediction on encoded and decoded image samples (rather than on original video frame samples) in order to ensure that the encoder and decoder predictions are identical.

3.2.2.1 Intra Prediction Process

Intra prediction is derived from decoded samples of the same decoded slice in encoder and this process extracts the spatial redundancy between adjacent macroblocks in a slice. The intra predicted pictures usually give better quality and lower distortion than inter predicted picture, but intra prediction requires much more bits to represent the samples. Because of the higher bit rate requirement of an intra predicted slice, the number of the intra predicted slices is quite less than the inter slices for reduction of bits in stream.

H.264/AVC standard supports intra-prediction for blocks of 4x4 to help achieve better compression for high motion areas. There are nine prediction modes in Intra_4x4 mode as shown in Figure 3.5. This mode is supported only for luma blocks.

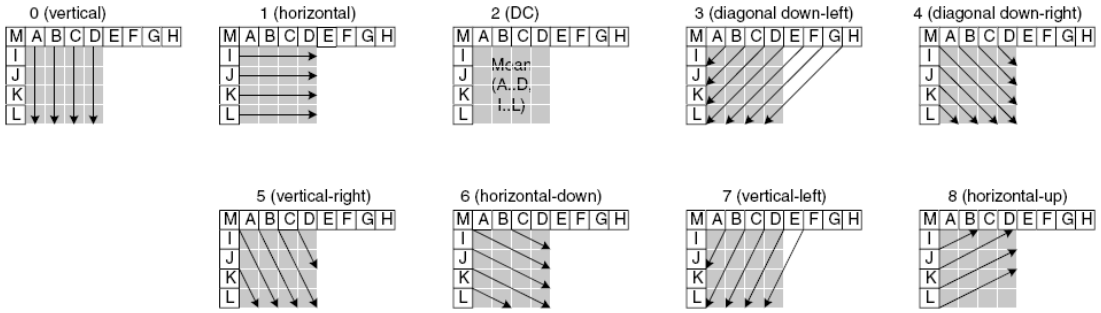

Figure 3.5: Intra_4x4 Prediction Modes

H.264/AVC also has a 16 x 16 mode, which is aimed to provide better compression for flat regions of a picture at lower computational costs. This mode is also helpful to avoid the irritating gradients that show up in flat regions of the picture quantized with high quantization parameters. Intra_16x16 mode supports 4 direction modes as shown in Figure 3.6. This mode is supported for 16x16 luminance blocks and 8x8 chrominance blocks.

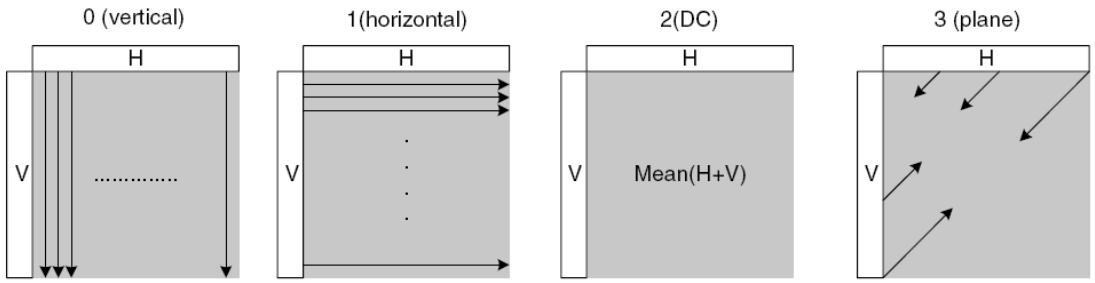

Figure 3.6: Intra_16x16 Prediction Modes

A further intra coding mode, I_PCM, enables encoder to transmit the values of the image samples directly (without prediction or transformation). The I_PCM mode allows the encoder to precisely represent the values of the samples. It enables placing a hard limit on the number of bits a decoder must handle for a macroblock without harm to coding efficiency.

3.2.2.2 Inter Prediction Process

In video coding, it is well known that temporal correlations of macroblocks are stronger than spatial correlations of macroblocks. Inter prediction is carried out on the decoded samples of reference pictures other than the current decoded picture, and this process eliminates the temporal redundancy between successive pictures for the compression.

Inter prediction in H.264/AVC supports variable block sizes from 16x16 to 4x4 as depicted in Figure 3.7 and in Figure 3.18.

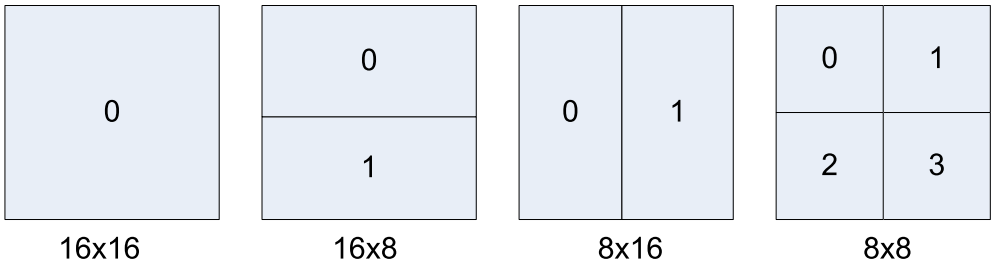

Figure 3.7: Macroblock Partitions: 16x16, 16x8, 8x16, and 8x8

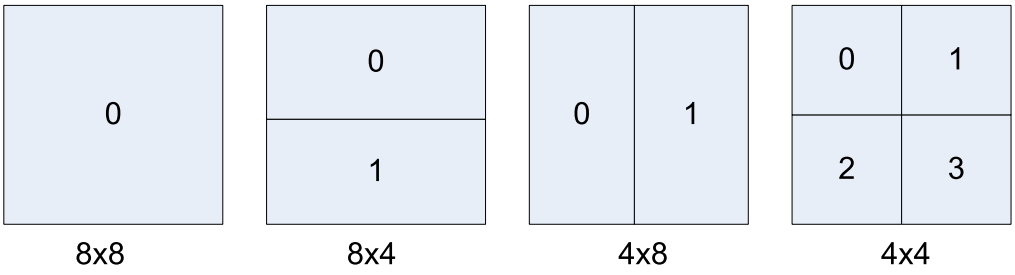

Figure 3.8: Sub-macroblock Partitions: 8x8, 8x4, 4x8, and 4x4

The luminance component of each macroblock (16×16 samples) may be split up in four ways (Figure 3.7) and motion compensated either as one 16×16 macroblock partition, two 16×8 partitions, two 8×16 partitions or four 8×8 partitions. If the 8×8 mode is chosen, each of the four 8×8 sub-macroblocks within the macroblock may be split in a further 4 ways (Figure 3.8), either as one 8 × 8 sub-macroblock partition, two 8 × 4 sub-macroblock partitions, two 4×8 sub-macroblock partitions or four 4×4 sub-macroblock partitions.

These partitions and sub-macroblock give rise to a large number of possible combinations within each macroblock. This method of partitioning macroblocks into motion compensated sub-blocks of varying size is known as tree structured motion compensation.

In video coding standards, the macroblock motion is found out in motion estimation block using reference picture(s). Motion estimation block can be summarized as finding the minimal difference for original block and reference block (from reference picture). As the process has computational overhead, there are several methods and algorithms for approximated, restricted and fast motion estimation such as using search windows, stepped searches etc.

A separate motion vector is required for each partition or sub-macroblock. Each motion vector must be coded and transmitted and the choice of partitions must be encoded in the compressed bitstream. Choosing a large partition size (16×16, 16×8, 8×16) means that a small number of bits are required to signal the choice of motion vectors and the type of partition but the motion compensated residual may contain a significant amount of energy in frame areas with high detail. Choosing a small

partition size (8×4, 4×4, etc.) may give a lower-energy residual after motion compensation but requires a larger number of bits to signal the motion vectors and choice of partitions. The choice of partition size therefore has a significant impact on compression performance. In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas.

Each chroma component in a macroblock (Cb and Cr) has half the horizontal and vertical resolution of the luma component. Each chroma block is partitioned in the same way as the luma component, except that the partition sizes have exactly half the horizontal and vertical resolution (an 8×16 partition in luma corresponds to a 4×8 partition in chroma). The horizontal and vertical components of each motion vector (one per partition) are halved when applied to the chroma blocks.

The concept of B slices is generalized in H.264 when compared with prior video coding standards. In B slices, to build the prediction signal, some macroblocks or blocks may use a weighted average of two distinct motion-compensated prediction values. B slices employ two distinct lists of reference pictures, which are referred to as the first (list 0) and the second (list 1) reference picture lists. Four different types of inter prediction are supported: list 0, list 1, bi-predictive, and direct prediction. For the bi-predictive mode, a weighted average of motion-compensated list 0 and list 1 prediction signals is used for the prediction signal. The direct prediction mode is inferred from previously transmitted syntax elements and can be any of the other types of modes. For each 16x16, 16x8, 8x16, and 8x8 partition, list 0, list 1 or bi-predictive methods can be chosen separately. An 8x8 partition of a B macroblock can also be coded in direct mode. Similar to P-Skip mode, if no prediction error signal is transmitted for a direct macroblock mode, it is also referred to as B-Skip mode.

3.2.2.3 Transform & Quantization

In H.264 similar to the other standards, a transformation and quantization is applied on the prediction residuals. However, H.264/AVC employs a 4x4 integer transform as opposed to the 8x8 floating point DCT transform used in the other standard codecs. The transform is an approximation of the 4x4 DCT and hence it has similar

coding gain to the DCT transform. Since the integer transform has an exact inverse operation, there is no mismatch between the encoder and the decoder which is a problem in all DCT based codecs.

$$T_{\text{int}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

Figure 3.9: Integer 4x4 Forward Transformation Matrix

After the transformation using the matrix of Figure 3.9, each coefficient is scaled with a specified factor to make the final transform coefficient. The scaled coefficients are then quantized with a quantization step size determined by a given Quantization Parameter (QP). In H.264/AVC the values of the quantizer step sizes have been defined such that the scaling and quantizing stages are mixed to be performed by simple integer operations in both encoder and decoder [2]. In particular, the inverse operations of scaling, quantizing and transforming are directly described in the standard using pure integer operations [1].

3.2.2.4 Entropy Coding

Before transmission, generated data of all types are entropy coded. H.264 supports two different methods of entropy coding namely Context Adaptive Variable Length Coding (CAVLC) and Context Adaptive Binary Arithmetic Coding (CABAC). As well as a conceptual difference between the two methods, CABAC is more efficient than CAVLC which itself is superior to the conventional VLC (Huffman) used in the other standard video coding standards.

In CAVLC mode, the residual data is coded using CAVLC but other data are coded using simple Exp-Golomb codes. These data are first appropriately mapped to the Exp-Golomb codes depending on the data type (e.g. MB headers, MVs, etc.), and then the corresponding code words are transmitted.

The zigzag scanned quantized coefficients of a residual block are coded using Context Adapting VLC tables. The already coded information of the neighboring blocks (i.e. upper and left blocks) and the coding status of the current block determine the context. Optimized VLC tables are specifically provided for each context to efficiently code the coefficients in different statistical conditions.

In CABAC mode, the generated data including headers and residual data are coded using a binary arithmetic coding engine. The compression improvement of CABAC is the consequence of non-integer length symbol assignment, adaptive probability estimation and improved context modeling scheme.

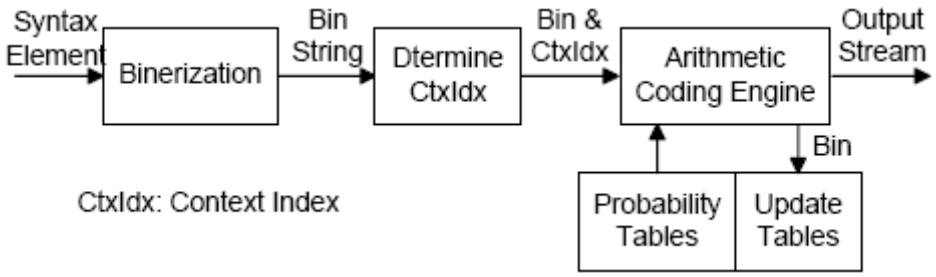

Figure 3.10: A Simplified Block Diagram of CABAC Coder

A block diagram of CABAC coding process is depicted in Figure 3.10. In order to code a syntax element, it is first mapped to a binary sequence called bin string. In the standard, proper binarization mapping schemes are provided for different types of data. For each element of the bin string (i.e. bin) a context index is defined based on the neighboring information and the coder status. There are 399 different contexts in the standard for various types of data, and the context modeling scheme (i.e. derivation of the context index) for each data type is clearly specified. The binary arithmetic coder engine then codes the bins using associated probability estimation tables addressed by the context index and generates the output stream. Subsequently, the probability tables are updated based on the coded bins for the future use.

3.2.2.5 Deblocking Filter

A deblocking filter is applied to each decoded macroblock to reduce blocking distortion. The deblocking filter is applied after the inverse transform in the encoder (before reconstructing and storing the macroblock for future predictions) and in the

decoder (before reconstructing and displaying the macroblock). The filter smoothes block edges, improving the appearance of decoded frames. The filtered image is used for motion-compensated prediction of future frames and this can improve compression performance because the filtered image is often a more faithful reproduction of the original frame than a blocky, unfiltered image.

In summary, a hybrid video encoding algorithm is used in H.264/AVC standard and typically proceeds as follows: Each picture is split into blocks. The first picture of a video sequence (or for a "clean" random access point into a video sequence) is typically coded in intra mode. For all remaining pictures of a sequence or between random access points, typically inter picture coding modes are used for most blocks. The encoding process for inter prediction consists of choosing motion data comprising the selected reference picture and motion vector to be applied for all samples of each block. The motion and mode decision data, which are transmitted as side information, are used by encoder and decoder to generate identical inter prediction signals using motion compensation [12].

The residual of the intra or inter prediction, which is the difference between the original block and its prediction, is transformed by a frequency transform. The transform coefficients are then scaled, quantized, entropy coded, and transmitted together with the prediction side information [12].

The encoder duplicates the decoder processing so that both will generate identical predictions for subsequent data. Therefore, the quantized transform coefficients are constructed by inverse scaling and are then inverse-transformed to duplicate the decoded prediction residual [12].

The residual is then added to the prediction, and the result of that addition may then be fed into a deblocking filter to smooth out block-edge discontinuities induced by the block-wise processing. The final picture (which is also displayed by the decoder) is then stored for the prediction of subsequent encoded pictures [12].

4. GOP BASED H.264/AVC VIDEO EDITING

4.1 Compressed Domain Processing

Compressed-domain processing performs a user-defined operation on a compressed video stream without going through a complete decompress/process/re-compress cycle; the processed result is a new compressed video stream. In other words, the goal of compressed-domain processing (CDP) algorithms is to efficiently process one standard-compliant compressed video stream into another standard-compliant compressed video stream with a different set of properties. In this thesis, cutting a portion from a H.264/AVC coded stream and splicing of two H.264/AVC streams into one stream are discussed.

A conventional solution to the problem of processing compressed video streams, shown in the top path of Figure 4.1, involves the following steps: first, the input compressed video stream is completely decompressed into its pixel domain representation; this pixel-domain video is then processed with the appropriate operation; and finally the processed video is recompressed into a new output compressed video stream. Such solutions are computationally expensive and have large memory requirements. In addition, the quality of the coded video can deteriorate with each re-coding cycle.

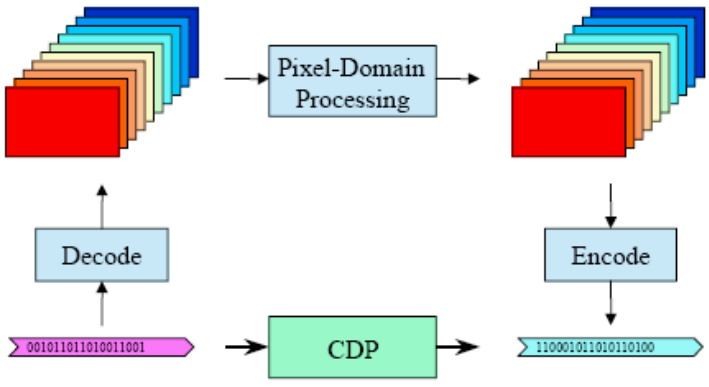

Figure 4.1: Pixel Domain Processing & Compressed Domain Processing

Compressed domain processing methods can lead to a more efficient solution by only partially decompressing the bitstream and performing processing directly on the compressed-domain data. CDP algorithms can have benefits like savings for memory, processing power and delay (due to decoding and then re-encoding the edited video back to the compressed domain), and the preservation of picture quality by avoiding lossy decode/re-encode chain.

4.2 H.264/AVC GOP Structure

In MPEG encoding, GOP specifies the order in which intra-frames and inter frames are arranged. The GOP is a group of successive pictures within an MPEG-coded video stream. Each MPEG-coded video stream consists of successive GOPs. From the MPEG pictures contained in it the visible frames are generated.

Each MPEG sequence has a sequence header at its leading end and a number of GOPs following the sequence header. The sequence header carries various parameters necessary to expand the sequence. Each GOP includes a GOP header and a plurality pictures following the GOP header, such as I-picture (intra frame), B-picture (bidirectional interpolated frame) and P-picture (predictive frame), which are produced in a pattern I,B,B,P,B,B, repeatedly.

The I-picture includes one complete data for one frame and can reproduce one frame picture by itself using parameters from GOP header and sequence header. The B-picture includes data for one frame, but can not reproduce one frame picture by itself. Similarly, the P-picture includes data for one frame, but can not reproduce one frame picture by itself.

GOP always begins with an I-frame. Afterwards several P-frames follow, in each case with some frames distance. In the remaining gaps are B-frames. With the next I-frame a new GOP begins.

The GOP structure is often referred by two numbers, for example M=3, N=12. The first one tells the distance between two anchor frames (I or P). The second one tells the distance between two full images (I-frames), it is the GOP length. For the above

example, the GOP structure is IBBPBBPBBPBB. Instead of the M parameter one can use the maximal count of B-frames between two consecutive anchor frames.

The more I-frames the MPEG stream has, the more it is editable. However, having more I-frames increases the stream size. In order to save bandwidth and disk space, videos prepared for internet broadcast often have only one I-frame per GOP.

H.264/AVC standard itself does not actually include a definition of a concept called GOP. GOP term is used to identify a group of pictures with starts with an I frame. Every I frame is a new GOP starting point.

4.2.1 Byte Stream NAL Unit Syntax and Parsing

Since H.264/AVC standard does not define a GOP structure, there is no identical header for GOP (which MPEG-2 standard has). In order to catch the start of a new GOP, one has to search for a new I frame. This is done by decoding the bitstream headers and look for a header of a NAL unit that contains an I frame. Table 4.1 shows the syntax of byte stream NAL units.

Table 4.1: Byte Stream NAL Unit Syntax

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream( ) && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}		

This table specifies how a NAL unit is parsed from the bitstream. Categories (labelled in the table as C) specify the partitioning of slice data into at most three slice data partitions. Slice data partition A contains all syntax elements of category 2.

Slice data partition B contains all syntax elements of category 3. Slice data partition C contains all syntax elements of category 4 [1].

The following descriptors specify the parsing process of each syntax element [1]:

– ae(v): context-adaptive arithmetic entropy-coded syntax element.

– b(8): byte having any pattern of bit string (8 bits).

– ce(v): context-adaptive variable-length entropy-coded syntax element with the left bit first.

– f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first.

– i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements.

– me(v): mapped Exp-Golomb-coded syntax element with the left bit first.

– se(v): signed integer Exp-Golomb-coded syntax element with the left bit first.

– te(v): truncated Exp-Golomb-coded syntax element with left bit first.

– u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements.

– ue(v): unsigned integer Exp-Golomb-coded syntax element with the left bit first.

The parsing process for these descriptors are specified in H.264/AVC standard [1].

Input to parsing of byte stream NAL units process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures. Output of parsing process consists of a sequence of NAL unit syntax structures.

At the beginning of the parsing process, the decoder initializes its current position in the byte stream to the beginning of the byte stream. It then extracts and discards each leading_zero_8bits syntax element (if present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence 0x00000001.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded [1]:

1. When the next four bytes in the bitstream form the four-byte sequence 0x00000001, the next byte in the byte stream (which is a zero_byte syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.

2. The next three-byte sequence in the byte stream (which is a start_code_prefix_one_3bytes) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this threebyte sequence.

3. NumBytesInNALunit is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of any of the following conditions:

    a. A subsequent byte-aligned three-byte sequence equal to 0x000000, or

    b. A subsequent byte-aligned three-byte sequence equal to 0x000001, or

    c. The end of the byte stream, as determined by unspecified means.

4. NumBytesInNALunit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNALunit bytes. This

sequence of bytes is nal_unit(NumBytesInNALunit) and is decoded using the NAL unit decoding process.

5. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to 0x000001 and the next bytes in the byte stream do not start with a four byte sequence equal to 0x00000001, the decoder extracts and discards each trailing_zero_8bits syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence 0x00000001 or the end of the byte stream has been encountered (as determined by unspecified means).

4.2.2 NAL Unit Syntax and Semantics

After parsing the NAL header, NAL unit is obtained. In Table 4.2, NAL unit syntax is given.

Table 4.2: NAL Unit Syntax

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		
for(i = 1; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
}		
}		

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit.

“forbidden_zero_bit” shall be equal to 0.

“nal_ref_idc” not equal to zero specifies that the content of the NAL unit contains a SPS or a PPS or a slice of a reference picture or a slice data partition of a reference picture. For a new GOP start point, this variable shall not be zero.

“nal_unit_type” specifies the type of RBSP data structure contained in the NAL unit as specified in Table 4.3. VCL NAL units are specified as having “nal_unit_type” equal to 1 to 5. Remaining NAL units are called non-VCL NAL units.

Table 4.3: NAL Unit Type Codes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	10
14..18	Reserved	
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	2, 3, 4
20..23	Reserved	
24..31	Unspecified	

4.3 GOP Based Editing

In this thesis, only closed GOPs are studied. In these GOPs, all the frames in the current GOP can be decoded even if the previous GOPs do not exist. That means the frames inside the GOP do not referenced from the frames that belong to previous GOPs. Since GOPs are independent, splicing or cutting operations is become just a copy and a paste operation of GOPs in compressed domain.

As stated in [6], some terminology is used in compressed domain video editing. "Starting cut point" is the point where the extraction operation is going to start and "ending cut point" is the point where extraction operation is going to end. The last frame before the starting cut point is called "exit frame" and the first frame after the ending cut point is called "entry frame". Correspondingly, the GOPs containing the exit frame and the entry frame are called exit GOP and entry GOP respectively. After the editing operation frames between exit frame and entry frame are extracted from the original sequence and output sequence is formed.

By means of GOP based editing, it is meant to say there is no frame based operations on the video sequences. Every processing remains at GOP level. Even if the starting and/or ending cut points are selected inside the GOPs, cutting and splicing operations are done on GOPs and resulting output stream may include frames that are between start cut point and ending cut point (which they are not intended to be included in output stream). An example is shown in Figure 4.2.

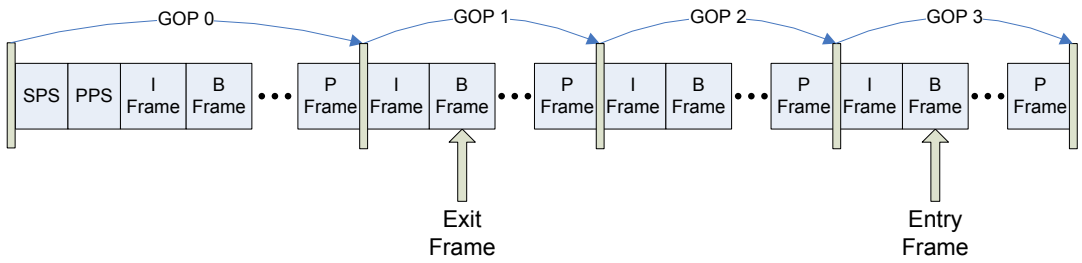

Figure 4.2: An Example of GOP-Based Editing (Cutting)

After the cutting operation of the above example, GOP 2 is discarded and the output video sequence consists of GOP 0, GOP 1, and GOP 3. This can be seen on Figure 4.3.

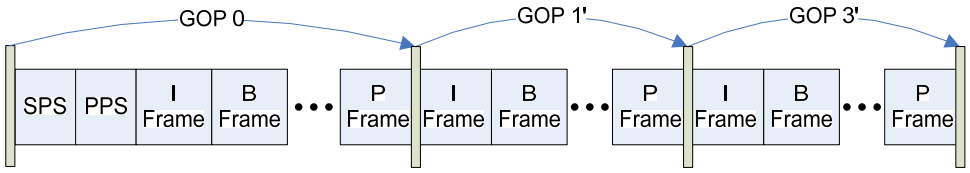

Figure 4.3: Output Video Sequence after Cutting Operation

If the starting cut point and ending cut point is selected at GOP boundaries, after GOP based editing, every frame is contained in the output stream. In this case, exit frame is an I frame (and a starting point of a new GOP), and entry frame is the last frame of a GOP (can be a P or a B frame). This case is illustrated in Figure 4.4.

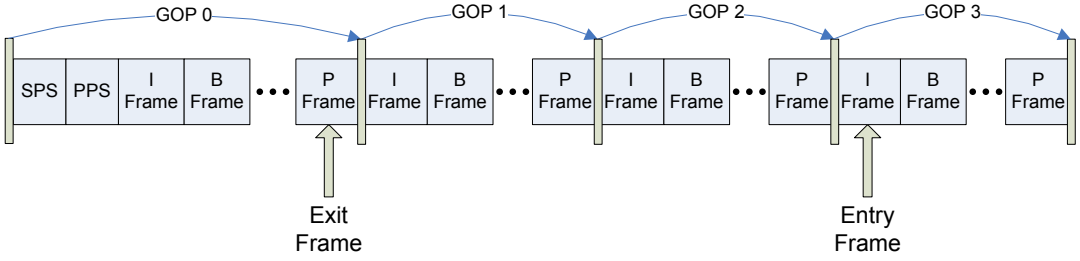

Figure 4.4: An Example of Editing at GOP Boundaries

In the above example starting cut point is just after the exit frame, and the ending cut point is just before the entry frame. After the cutting operation GOP 1 and GOP 2 are discarded and the output video sequence consists of GOP 0, and GOP 3. Accurate editing is done in this example and output video sequence is illustrated in Figure 4.5.

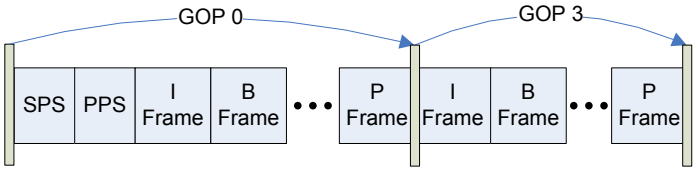

Figure 4.5: Output Video Sequence after Cutting Operation

Simulations are done on H.264/AVC reference software, which is also called JM [5]. This software includes encoder/decoder pair. Version 8.0 is used in simulations since newer versions come with more complexity.

Input streams that are used in GOP based editing is generated from the encoder which is included in JM Software. As GOP length, period of I frames is selected 12. Than a control mechanism is setup and included in decoder code to simulate GOP based editing in compressed domain. The steps during the GOP based cutting operation are:

a) Decode the input bitstream until a new NAL unit header (0x000001 or 0x00000001) is found.

b) Decode “nal_unit_type” (5 bits) and determine the type of the NAL unit. If the NAL unit is an SPS or a PPS than copy this NAL unit to the output stream.

c) If the NAL unit is an I frame, this is a new GOP starting point. Copy this NAL unit and the upcoming NAL units until the GOP that includes exit frame.

d) After exit frame is found, bypass the next GOPs until entry frame is found.

e) After the GOP that includes entry frame is found, concatenate this GOP and the residual GOPs to the output stream.

5. FRAME ACCURATE H.264/AVC VIDEO EDITING

5.1 Frame Accurate Cutting & Splicing

In frame accurate editing, as opposed to GOP based editing, it is desired that every selected frame is contained in the output stream after the editing operation. If the starting cut point and ending cut point is selected at GOP boundaries, the editing process becomes GOP based editing. But if the starting cut point or ending cut point is selected inside the GOP, additional processing is required to discard every frame between starting cut point and ending cut point.

Figure 5.1 shows an example of frame accurate cutting operation. Here, the numbers under the frames shows the frame numbers in display order. Exit frame is fourth frame and entry frame is nineteenth frame in this case. Exit GOP is first GOP and entry GOP is fourth GOP respectively.

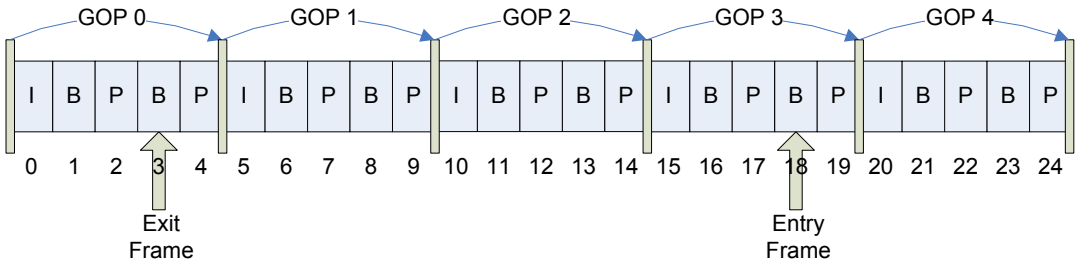

Figure 5.1: Frame Accurate Cutting Operation

After cutting operation, output video sequence consists of frames 0-3 and 18-24 (3 and 18 included). Clearly some transformations and additional processing are required around exit frame and entry frame regions. These are going to be explained in next pages. Output video sequence of the above example is shown in Figure 5.2.

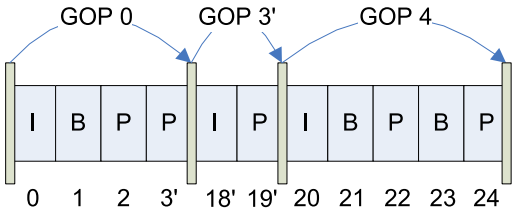

Figure 5.2: Output of Frame Accurate Cutting Operation

As explained later, “entry GOP processing” and “exit GOP processing” are the basis for these two operations (cutting, and splicing). Only difference between them is; splicing has two exit and entry frames (and two entry/exit GOP processing accordingly). Frame accurate splicing operation of two video segments from two different video sequences is shown in Figure 5. In this example, video frames between 3 and 10 of video sequence 2 are concatenated after video frames between 6 and 17 from video sequence 1. Output video sequence is shown in Figure 5.4.

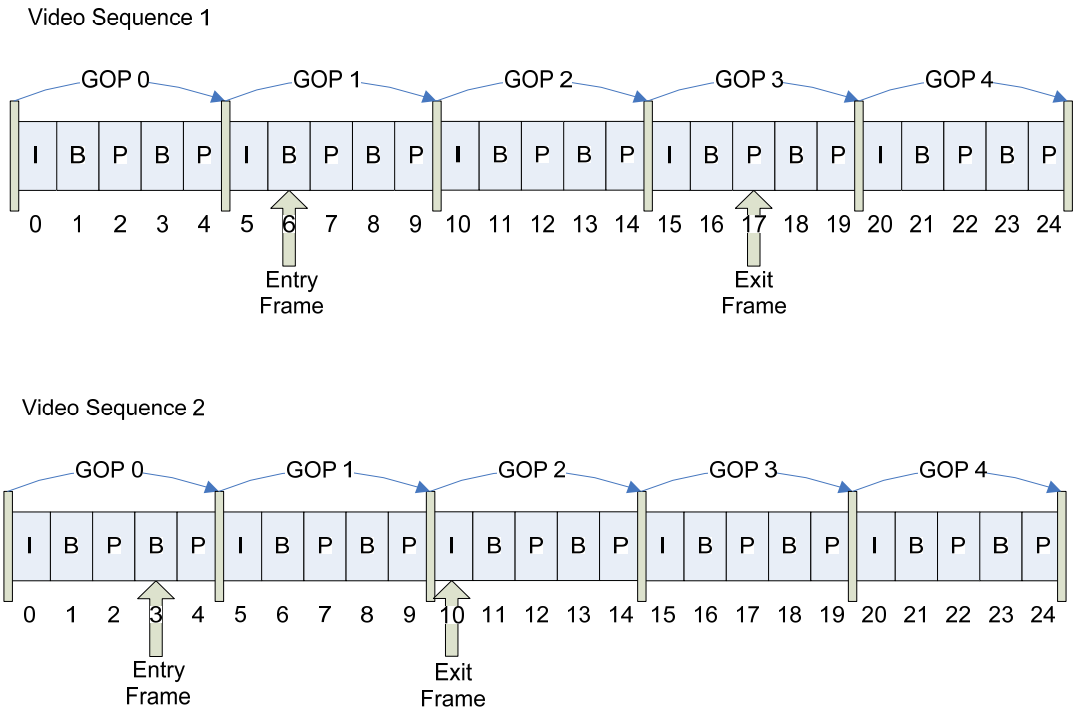

Figure 5.3: Frame Accurate Splicing Operation

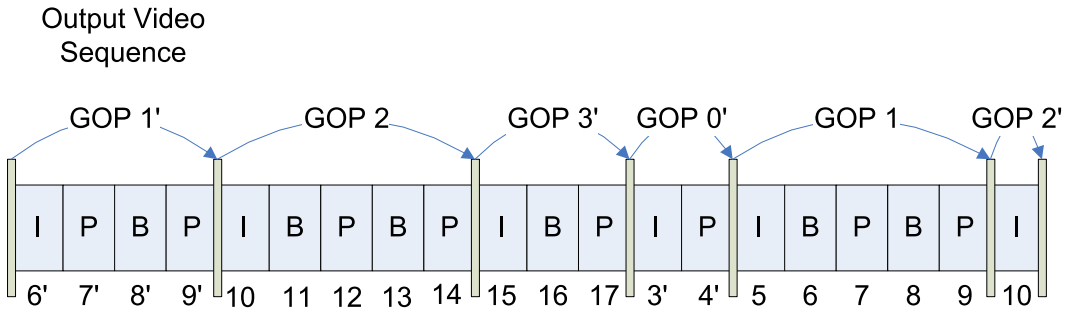

Figure 5.4: Output of Frame Accurate Splicing Operation

5.2 Conversion Procedures

As the nature of the video coding standards, there are temporal dependencies between frames. When we cut a portion of video sequence from a video stream, it is likely that the reference frames of some inter-coded frames (P, B) in the output sequence may not be included. So, there must be frame conversion algorithms to solve this problem. In this work, two procedures are going to be explained:

i) Exit GOP Processing

ii) Entry GOP Processing

These conversion procedures determine necessary frame conversions that can be possible decoding, re-encoding, and other processing. These two procedures work on the related GOPs (entry and exit GOPs) and resultant video sequence is frame-accurate, and must be conformable to H.264/AVC standard.

These conversion procedures are independent from each other and handled separately. For cutting operation, as seen on Figure 5.1, firstly exit GOP processing is applied to exit GOP and then entry GOP processing is applied to entry GOP. For splicing operation, as seen on Figure 5.3, the processing order is reversed. First entry GOP processing is applied to entry GOP in first input video sequence and then exit GOP processing is applied to exit GOP in first input video sequence. Same operations are done for the second input video sequence and the extracted GOPs are combined together.

Before presenting these conversion procedures, the problematic case of existence of B frames should be presented: The order in which decoded pictures are displayed is called the display order. The order in which the pictures are transmitted and decoded is called the coded order. The coded order is the same as the display order if there are no B-frames in the sequence. However, if B-frames are present, the coded order may not be the same as the display order because B-frames typically use temporally future reference frames as well as temporally past reference frames, and a temporally future reference frame for a B-frame precedes the B-frame in coded order. Difference between display order and the coded order can be clearly seen on Figure 5.5 and Figure 5.6.

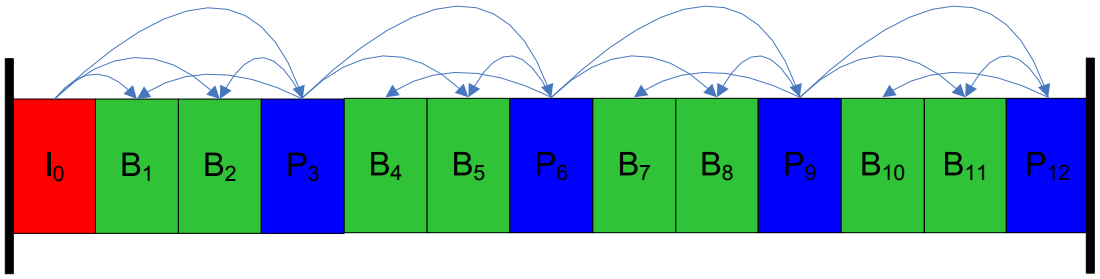

Figure 5.5: Display Order

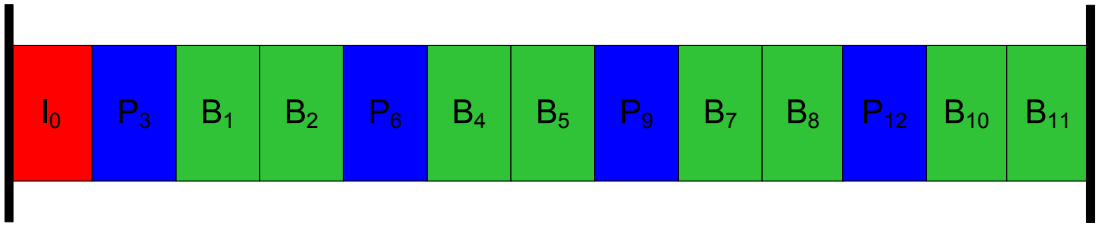

Figure 5.6: Coded Order

If the video sequence is composed of only I and P pictures, the processing is relatively simple since there is no backward processing. However if the sequence has B frames (and any of the cut points is selected adjacent to B frames), then the conversion procedures embody re-encoding and reordering of these coded pictures. Because after cutting or splicing operation, backward or forward reference of B frames may be discarded.

5.2.1 Exit GOP Processing

This processing works around the exit frame. Exit GOP processing may include decoding, and re-encoding of some frames around the exit frame. There are three possible cases for this process. These cases completely depend on the type of the exit frame. In all of the cases, it is assumed that the input video sequence consists of I, P, and B frames.

5.2.1.1 I Frame Processing

If the exit frame is an I frame, this means this frame is the first frame of the GOP. Also this frame does not need any of the following frames to be decoded. So, terminating the exit GOP after getting the I frame would not be problematic. This is illustrated in Figure 5.7.

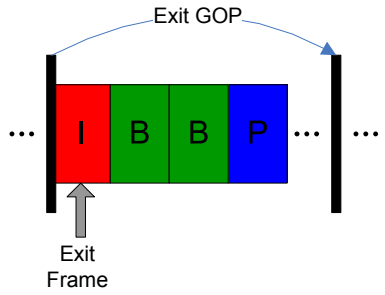

Figure 5.7: Exit GOP (exit frame is an I frame)

For I frame case process works as follows: Until the exit frame is met, decode the NAL headers and copy these NAL units to the output stream. When the exit frame is met, copy also this NAL unit to the output stream. Exit processing is completed. Figure 5.8 shows the output GOP that is going to be added to the output stream.

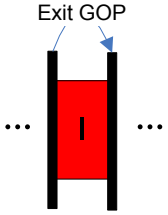

Figure 5.8: Output GOP (exit frame is an I frame)

5.2.1.2 P Frame Processing

If the exit frame is a P frame, procedure is almost like the above situation. The only difference is continuing grabbing operation of frames after the exit frame is met. This grabbing lasts until the frames that come after exit frame in coded order but come before in display order, are all read from the input stream and copied to the output stream.

This case is shown in Figure 5.9. Frames are arranged in display order and subscript numbers shows the coded order. P_1 frame is read from the input stream before frames B_2 and B_3. After copying NAL unit that includes P_1 frame in it, processing must continue and copying all of the frames that resides between I_0 and P_1 in display order must be copied to the output stream. Output GOP is shown in Figure 5.10.

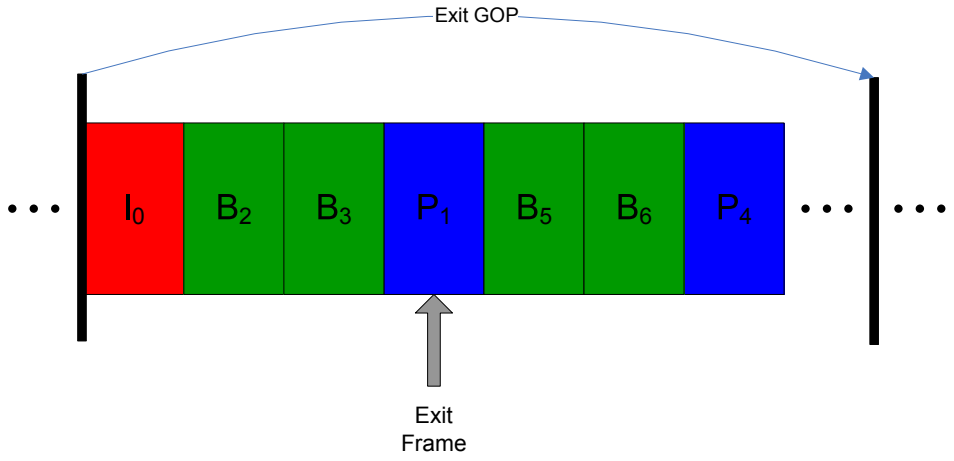

Figure 5.9: Exit GOP (exit frame is a P frame)

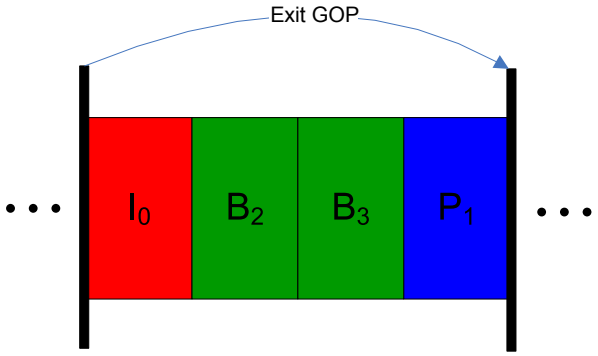

Figure 5.10: Output GOP (exit frame is a P frame)

5.2.1.3 B Frame Processing

If the exit frame is a B frame, additional effort is needed because backward reference frame of exit frame (and several possible frames) is going to be discarded in the output stream.

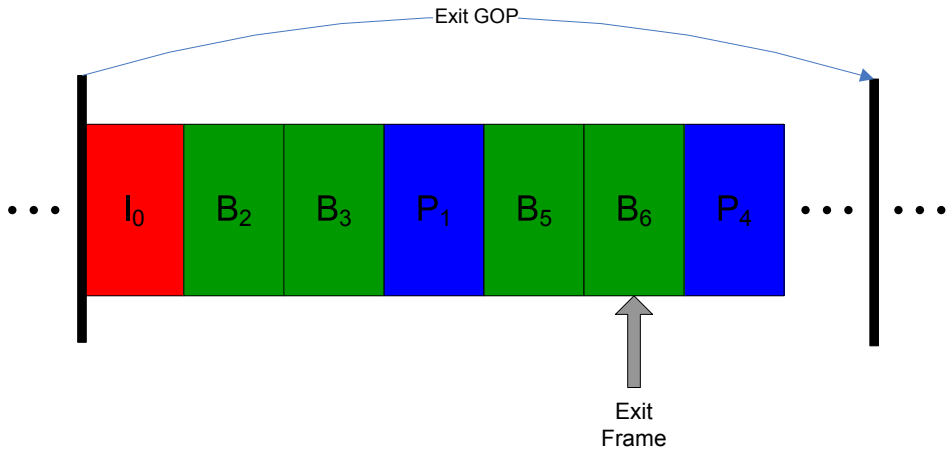

Figure 5.11: Exit GOP (exit frame is a B frame)

As seen from the figure, when exit frame is B_6, backward reference frame of B_5 and B_6 (P_4) is going to be discarded in the output stream. Since only P_1 can be a reference to B_5 and B_6, each of these B frames must be converted to a P frame (that uses only forward prediction). This operation includes decoding all of the B frames and then re-encoding these frames in frame type of P.

The steps during this operation are:

a) Copy NAL units that include frames from starting of the exit GOP to the last P frame that comes before the exit frame. In above example these are I_0, P_1, B_2, and B_3.

b) To use as a reference frame, decode the previous reference frame (I or P frame). In above example this frame is P_1.

c) Decode all of the B frames starting from the previous forward reference (I or P frame) to the exit frame. In above example these frames are B_5 and B_6.

d) Encode these frames (P_1, B_5, and B_6), with the supplemental information gathered while decoding operation.

e) After encoding, discard first frame (which is re-encoding of P_1) and copy remaining NAL units that include P coded frames (which is re-encoding of B_5 and B_6) to the output stream.

This operation is simulated in JM software by using both the decoder and the encoder. First, related frames are fully decoded (not just the headers). Then the raw image data sent to the encoder with the necessary ancillary data that is acquired while decoding. These are; profile and level of sequence, frame width & height, QP (quantization parameter) that determines the visual quality of the frames, number of reference frames (generally one). Alongside with these parameters, SPS and PPS of the input video sequence are sent to the encoder to prevent any incompatibility. All of these parameters are held in a structure and sent to encoder with the raw image data. Figure 5.11 illustrates this operation.

Figure 5.12: Frame Conversion of B Frames

After encoding operation we have a stream that is composed of an I frame (re-encoding of P_1) and following P frames. The NAL units that contain P frames (P_1 and P_2) are copied to output stream. Resulting output GOP is shown in Figure 5.13.

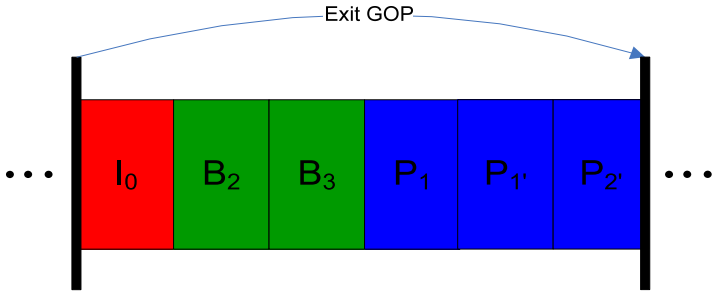

Figure 5.13: Output GOP (exit frame is a B frame)

There is a problem must be handled. When looked at the output exit GOP, it is seen that $P_{1'}$ and $P_{2'}$ are referencing from P_1. But the actual reference of $P_{1'}$ and $P_{2'}$ is I_0, decoded and re-encoded version of P_1. So, there will be a drift error that while decoding the resulting output stream, decoded frames of $P_{1'}$ and $P_{2'}$ are different from the frames that were get at encoder side. To minimize this error, while coding the P_1, the encoder parameters are set to give the high picture quality. This error drifts until to the end of the exit GOP, in this example for two frames. And after doing the simulations, it is seen that there is minimum visual difference between original input frames and the converted frames (B_5, B_6 and $P_{1'}$ and $P_{2'}$).

5.2.2 Entry GOP Processing

This processing works around the entry frame. Entry GOP processing includes decoding, and re-encoding of frames after the exit frame.

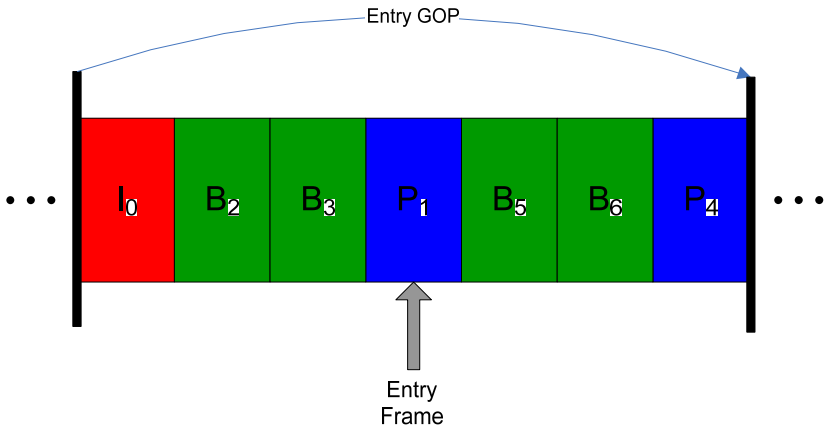

Figure 5.14: Entry GOP Processing

Since frames before entry frame are not supposed to be at the output stream, there are no forward reference frames for the frames after the entry frame. This can be seen on Figure 5.14. Any conversion of the entry frame will tend to conversion of the other frame until the end of the entry GOP. So, the procedure in entry GOP processing is:

a) Decode all of the frames starting from entry frame to the end of the entry GOP. In above example these frames are P_1, P_4, B_5, and B_6.

b) Encode these frames (P_1, P_4, B_5, and B_6), with the supplemental information gathered while decoding operation.

c) After encoding, copy the NAL units that include coded frames (which is re-encoding of P_1, P_4, B_5, and B_6) to the output stream.

Output of the above example is shown in Figure 5.15.

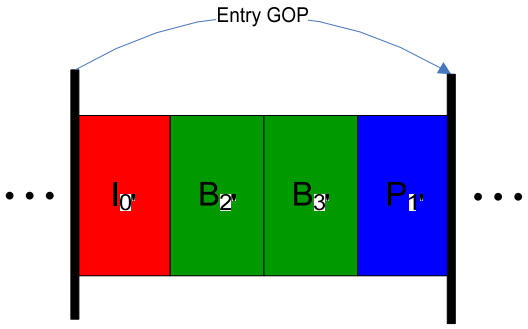

Figure 5.15: Output GOP (Entry GOP Processing)

5.3 Rate Control

Block-based hybrid video encoding schemes such as the MPEG-2 and H.264 are lossy processes. They achieve compression not only by removing truly redundant information, but also by making small quality compromises that perceptible. In particular, the quantization parameter QP regulates how much spatial detail is saved. When QP is very small, almost all the detail is retained. As QP is increased, bit-rate decreases which results some increase in distortion and some loss of quality.

There are two types of coding: CBR (Constant Bit Rate), and VBR (Variable Bit Rate). Because video compression is done by manipulating motion frames, if a video picture does not change dynamically, the compression rate can be much higher since there are few radical changes in the picture. VBR provides for a smaller file size by being able to vary the compression ratio as the content of video sequence changes. CBR keeps the transfer rate constant regardless of any changes in the video picture.

When encoding CBR sequences, the encoder usually enforces some rate control mechanisms to ensure that the generated bitstream will not cause buffer violations at the decoders.

In decoder side, video buffer is initially empty, then coded data bits are placed into the buffer at a constant bitrate from the input channels. After an initial delay, the decoder will accumulate enough bits (about 80% full) in the video buffer and start to fetch data from the buffer one frame at a time with a specific interval [11].

The decoder video buffer may overflow when the bitstream has many low bitrate frames in a series (such as a fade-in sequence) such that the decoder can not remove bits from the video buffer fast enough. Newly arrived bits will be lost due to lack of available buffer space. On the other hand, the video buffer may underflow if the bitstream has many large frame within a short period of time (due to abrupt scene changes or video editing). The decoder will quickly drain the buffer. Then at the supposed decoding time, the decoder cannot get a complete picture from the buffer. The video/audio will be jittery [11].

In editing of CBR sequences, the average bitrate of the output sequence should be kept the same as the input sequence and decoder buffer constraint sholud not be violated. Thus it is needed to allocate appropriate amount of bits before re-encoding the frames that need type conversion. Some methods are introduced for H.264 in [7], and for MPEG-2 in [11].

Simulation environment is JM Software and the decoder in JM software works as follows:

a) Search for a NAL unit header. After finding the NAL unit header, mark this point and search for the next NAL unit header. After finding the next NAL unit header mark this point as well.

b) Then copy the NAL unit which is between the marked points (whatever the size is) to the buffer. And decode the NAL unit.

c) Repeat this operation until the end of sequence is found.

In this thesis work, only VBR coding case is investigated. Decoder in JM Software works on NAL units and has no time stamp dependencies. Thus overflow and underflow situations can not be seen in this environment.

During frame conversion operations, encoder parameters are selected depending on the input video stream parameters. So, it is concluded that there is no overflow/underflow situation at cut points. Figure 5.16 shows the comparison of the data rate of the original video stream and the compressed domain edited output stream around cut point. In below example, frames between 15th and 22nd seconds of input video sequence are cut using compressed domain techniques proposed in this thesis work.

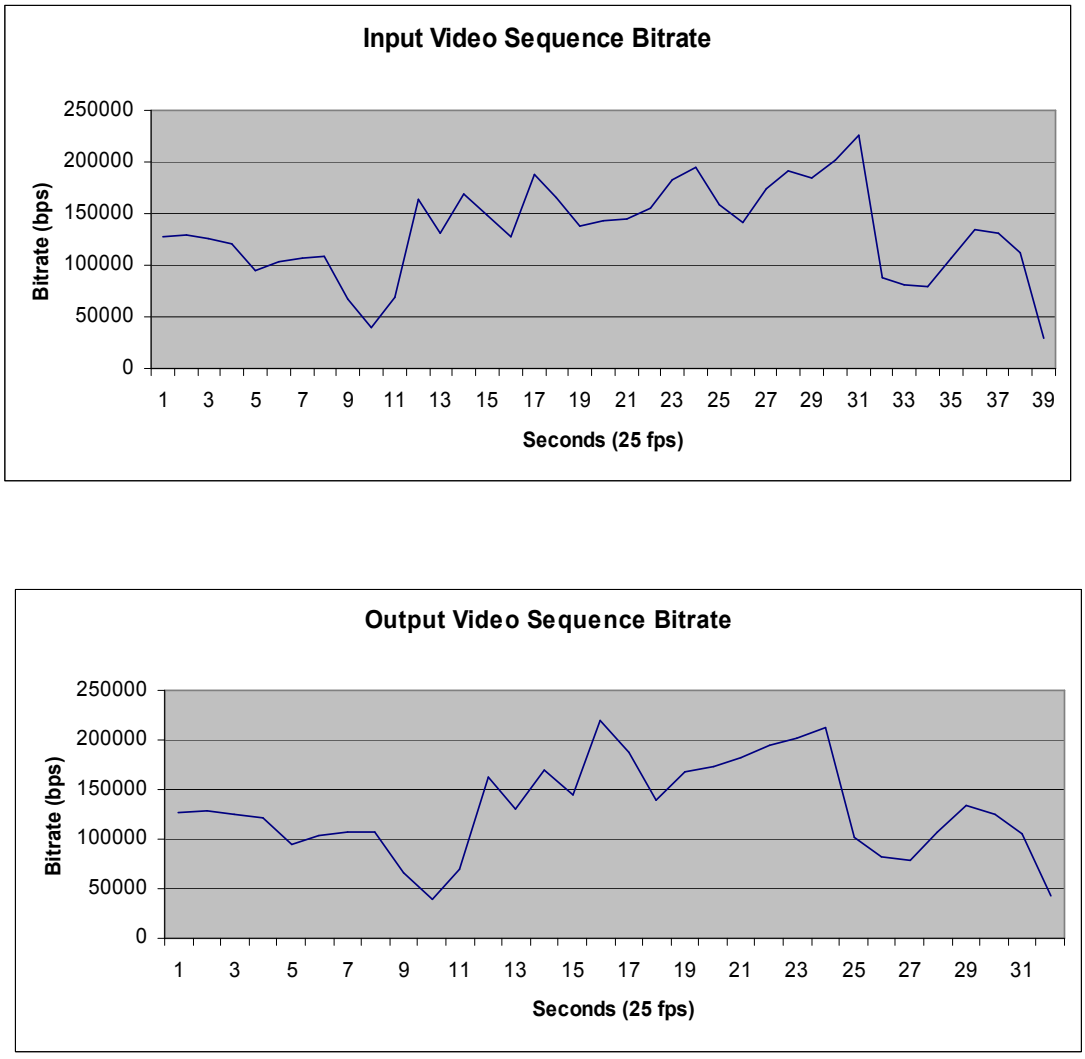

Figure 5.16: Bitrate Change after Compressed Domain Cutting Operation

5.4 Results

As mentioned before, for simulating the algorithms that proposed in this thesis work, H.264/AVC JM reference software is used. This software is released by the JVT and can be downloaded from the web [5].

This software packet is open source and includes both H.264/AVC encoder and H.264/AVC decoder. Codes are written in plain C and can be compiled and run under MS Windows and Linux operating systems. After being familiar with the structure of the encoder and decoder code, any algorithm can be embedded into this code and simulations can be done.

To simulate the compressed domain video cutting operation (both GOP-based editing and frame-accurate editing), a control mechanism is written in C language and embedded into the JM H.264 decoder code. Resulting program receives exit frame and entry frame from the user. In order to operate, decoder needs a configuration file which is shown in Figure 5.17.

```
airshow_jm132.264       ........H.26L coded bitstream
test_dec.yuv            ........Output file, YUV 4:2:0 format
test_rec.yuv            ........Ref sequence (for SNR)
10                      ........Decoded Picture Buffer size
0                       ........NAL mode (0=Annex B, 1: RTP packets)
596                     ........Start Cut Point (last frame in exit GOP) (in display order)
687                     ........End Cut Point (first frame in entry GOP) (in display order)
500000                  ........Rate_Decoder
104000                  ........B_decoder
73000                   ........F_decoder
leakybucketparam.cfg    ........LeakyBucket Params

This is a file containing input parameters to the JVT H.264/AVC decoder.
The text line following each parameter is discarded by the decoder.
```

Figure 5.17: JM H.264/AVC Decoder Configuration File

Then, it starts to decode the H.264 coded input video stream. This mechanism prevents full decoding of input video stream. In this mechanism, it is allowed to decode NAL headers and decode necessary frames around cut/splice points. After finding the exit frame and/or entry frame, mechanism decides whether re-encoding is necessary or not.

In exit GOP processing, the mechanism looks for the exit frame. After finding the exit frame (from NAL header), type of the frame is extracted from slice header. According to the type of the frame, exit GOP processing is executed on the exit GOP. If the frame is a B frame, re-encoding of certain frames is needed (5.2.1.3). To achieve this, these certain frames are fully decoded and then sent to the encoder. Then resulting encoded stream is added to the output video stream.

In entry GOP processing, same procedure is performed. In this case, all the frames from the entry frame to the end of the GOP are sent to encoder (5.2.2). Resulting video stream is added to the output stream.

To measure the performance of the video editing, a raw input video with a resolution of CIF is taken. It is first encoded in JM encoder with main profile, level 3.0, CAVLC, GOP length of 13, IBBPBBPBBBBBP format, parameters. Then this stream is used as input video sequence of video cutting operation and the exit frame is selected as a B frame. QP of P frames of input stream is 20 and QP of B frames is 30. At frame type conversion step (decoding, re-encoding), these values are used.

For conformance tests, several H.264/AVC coded streams are used. These streams are edited so that each case of exit GOP processing and entry GOP processing can be experimented. Then, these streams are decoded with JM decoder software. It is seen that compressed domain edited streams are completely frame-accurate and are fully complaint to the H.264/AVC standard.

To see the visual performance of the edited streams, an application called Stream Eye from Elecard Inc. is used. This application provides the user with a visual representation of the encoded video features and a stream structure analysis of MPEG-1/2/4 or AVC/H.264 Video Elementary Streams (VES), MPEG-1 System Streams (SS), MPEG-2 Program Streams (PS) and MPEG-2 Transport Streams (TS). More information can be obtained from their website (http://www.elecard.com).

Below, snapshots of Stream Eye application are given. In these figures visual quality of the re-encoded frames and bit-rate change can be seen clearly.

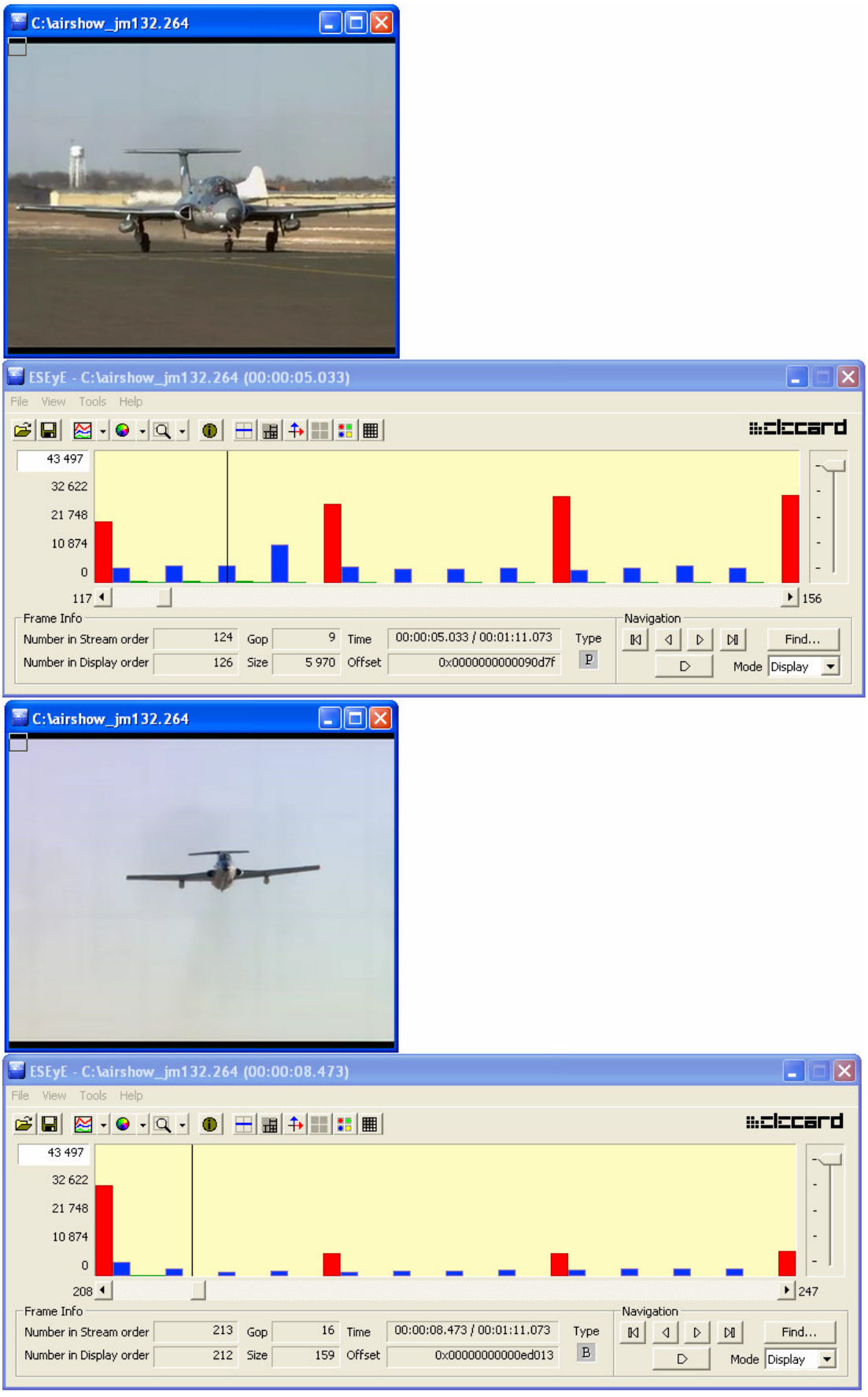

Figure 5.18: Exit and Entry Frames (and GOPs) in Input Stream

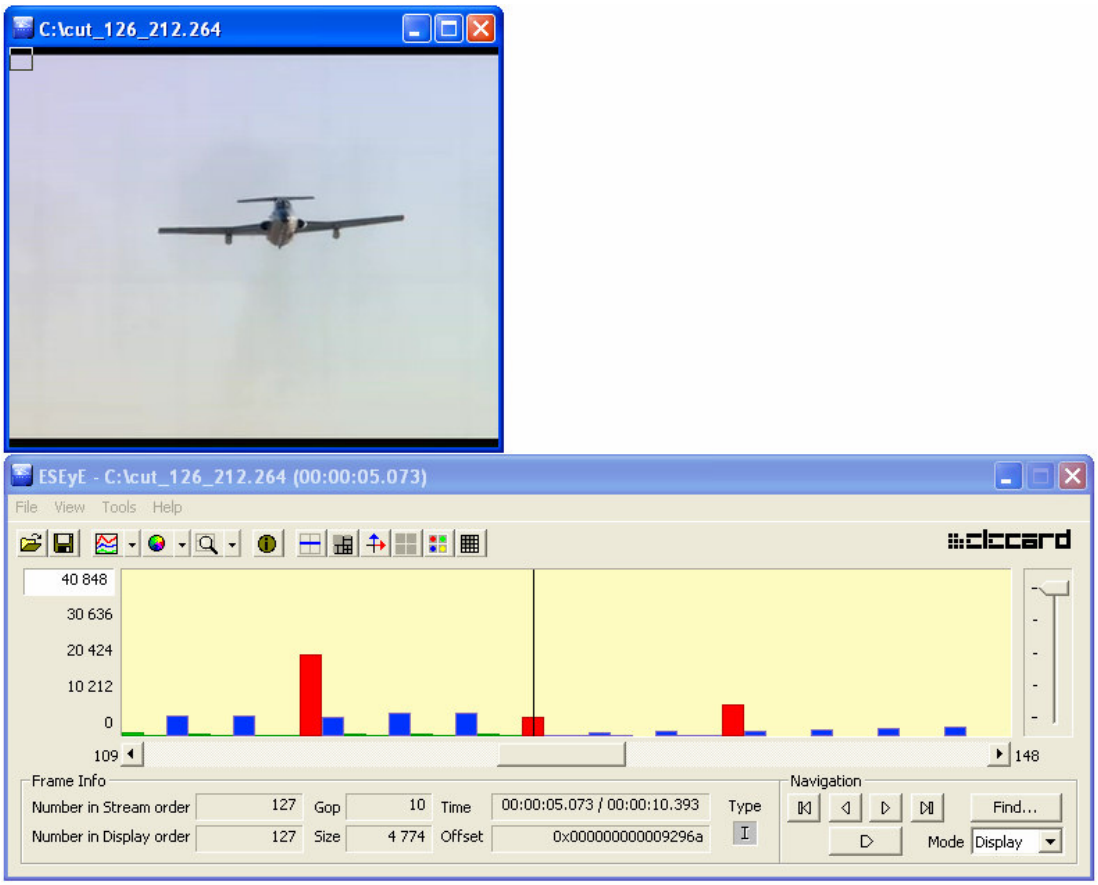

Figure 5.19: Output Stream after Cutting Operation

Above figures (Figure 5.18 and Figure 5.19) are examples of frame-accurate editing when exit-frame is a P frame and entry frame is a B frame. In this example, frames between display #126 and display #212 are cut. In exit GOP, frames up to the exit frame are directly copied to the output stream. In entry GOP, frames are re-encoded and appended to the output stream. It should be noticed that in Stream Eye program, frames are arranged in coded order.

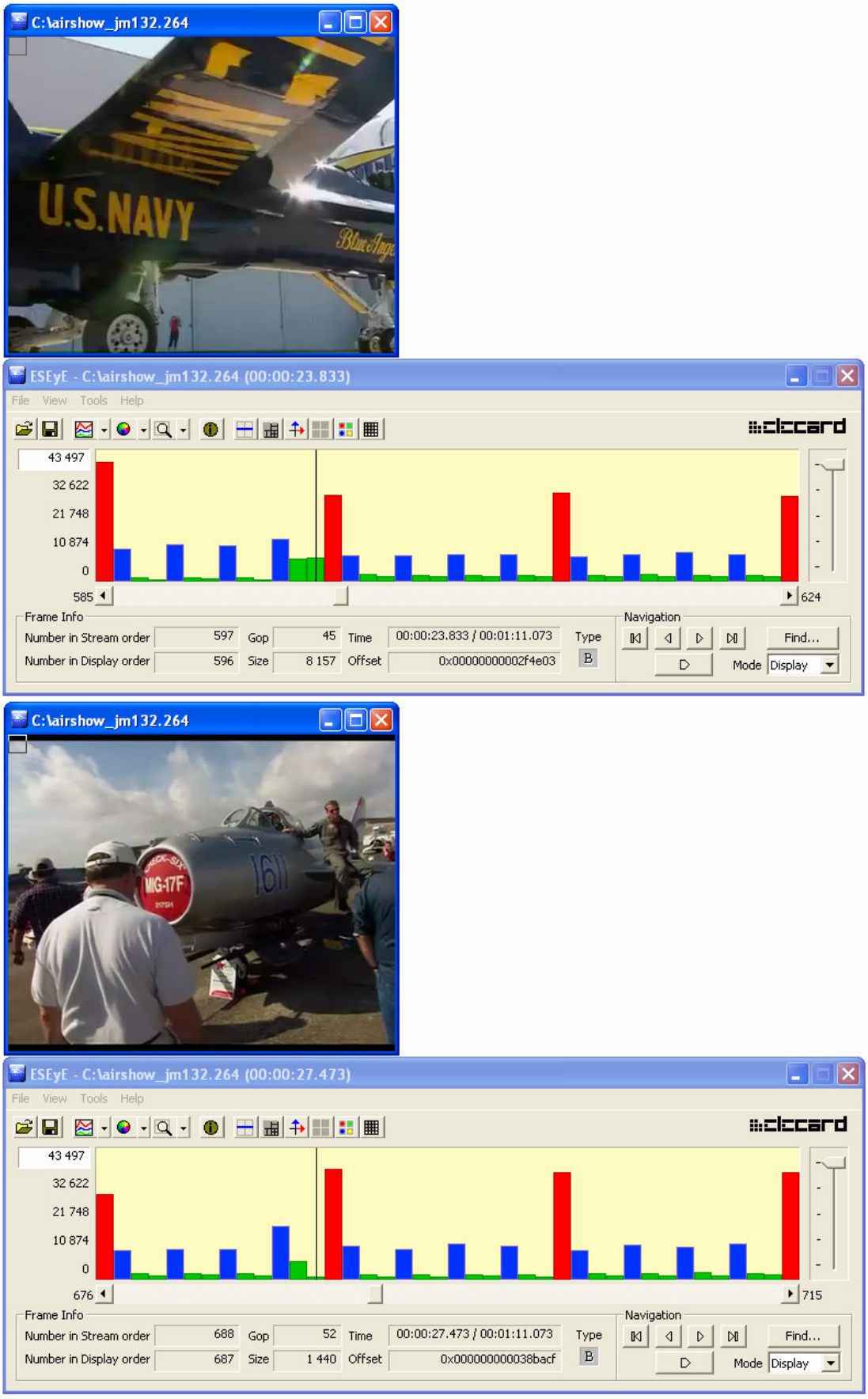

Figure 5.20: Exit and Entry Frames (and GOPs) in Input Stream

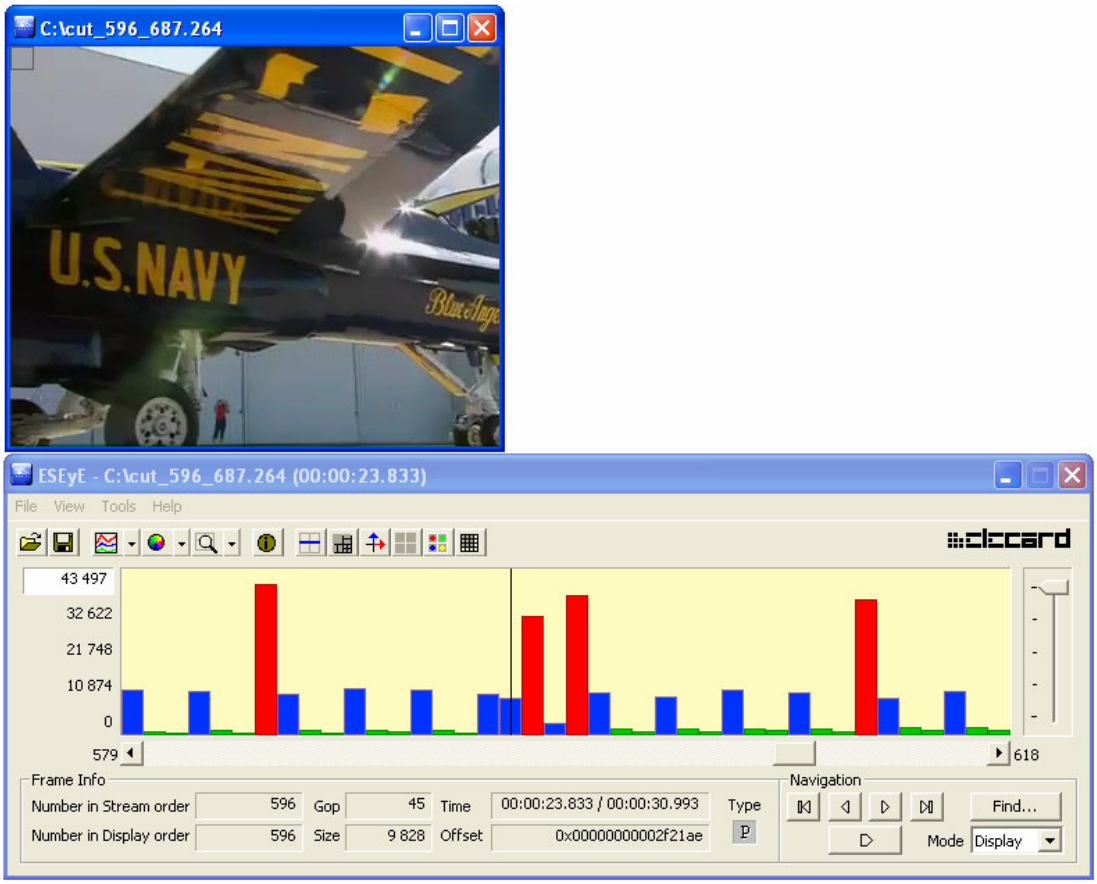

Figure 5.21: Output Stream after Cutting Operation

In above figures (Figure 5.20 and Figure 5.21) an example of frame-accurate editing when exit-frame is a B frame and entry frame is a B frame is illustrated. In this example, frames between display #596 and display #687 are cut. In exit GOP, frames with display numbers 594, 595, and 596 are fully decoded and re-encoded. Resulting H.264/AVC coded stream are appended to the output stream. In entry GOP, frames 687 and 688 are re-encoded and appended to output stream.

To compare the visual performance, PSNR of the originally B coded frame (display #596) and re-encoded and converted to P frame (display #596) are calculated and compared. Table 5.1 shows that after video editing, visual performance is decreased in acceptable amount.

Table 5.1: Visual Quality Comparison of Original Frame and Compressed Domain Edited Frame

	Original Frame (B frame, QP: 30)	Compressed Domain Edited Frame (P frame, QP: 20)
Y PSNR	38.1	37.69
Cb PSNR	45.7	45.16
Cr PSNR	47.7	47.14

CONCLUSION & FUTURE WORK

In this thesis, compressed domain editing techniques on H.264/AVC coded video sequences are investigated. Both GOP based methods and frame accurate methods are proposed. These methods are used for cutting portions from input video sequence or for splicing two different H.264/AVC coded video sequences.

Since there is no dependency between consecutive GOPs when GOPs are closed, cut and splice operaions are easily done on contents without decoding the originally coded stream.

As opposed to GOP-based editing, frame-accurate editing needs extra effort because of interframe coding. For exit GOP and entry GOP processing, several cases are investigated and methods are proposed. These methods require partial decoding; headers of the NAL units, some important frames around exit frame and entry frame. Also in some cases frame conversion is required. For that purpose, encoding operation with appropriate parameters is needed.

These methods have advantageous results over conventional solution (that is complete decoding and re-encoding of input stream). These are: savings for memory, processing power and time delay, and the preservation of picture quality by avoiding the lossy decode/re-encode chain.

These methods are embedded in the H.264/AVC standard reference code (JM) and proposed methods are simulated. Results show that, with small and acceptable decrease in visual quality and a small amount of increase in bitrate around cut points, proposed methods work successfully.

For future work, if the buffer control mechanism is setup, this work can be extended to CBR compliant model. After that, the use of these methods becomes more reasonable on limited capacity channels such as streaming and broadcast applications.

These methods are applied on CAVLC coded H.264/AVC streams. After adding CABAC support, main profile and other high profiles would be completely supported.

REFERENCES

[1] ITU-T Recommendation H.264 and ISO/IEC 11496-10 (MPEG-4, "AVC: Advanced Video Coding for Generic Audiovisual Services", version 3, 2005

[2] **Richardson, Iain E.G.,** H.264 and MPEG-4 Video Coding, John Wiley & Sons Ltd, UK, 2003

[3] **Wiegand, T. and Sullivan, G.J.,** Overview of the H.264/AVCVideo Coding Standard, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, 560-576

[4] **Kamaci, N. and Altunbasak, Y.**, 2003. Performance Comparison of the Emerging H.264 Video Coding Standard With the Existing Standards, *ICME '03 Proceedings 2003 International Conference on Multimedia and Expo*, Baltimore, Maryland, USA, 6-9 July, Vol. 1, 345-348

[5] Joint Video Team (JVT) of ITU-T VCEG and ISO IEC MPEG. H.264-AVC Joint Model (JM) Reference Software Version 8.0, http://iphome.hhi.de/suehring/tml/.

[6] **Wang, K. and Woods, J.W.**, Compressed Domain Mpeg-2 Video Editing, IEEE International Conference on Multimedia and Expo, 2000.

[7] **Yoneyama A., Takishima Y., and Nakajima Y.**, A Fast Frame-Accurate H.264/MPEG-4 AVC Editing Method, *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on Volume ,* Issue , 6-6 July 2005 , 1298 – 1301

[8] **Islam, A. Chebil, F. Hourunranta, A.** , Efficient Algorithms for Editing H.263 and MPEG-4 Videos on Mobile Terminals, *IEEE International Conference on Image Processing*, 2006.

[9] **Egawa R.**, Compressed Domain MPEG-2 Video Editing with VBV Requirement, *ICI*, (Vol I: 1016-1019)., 2000.

[10] **F. Chebil, R. Kurceren, A. Islam, U. Budhia**, Compressed domain editing of H.263 and MPEG-4 videos, *IEEE Transactions on Consumer Electronics*, Vol. 51, No. 3, 947-957.

[11] **Jianhoo M., Shih-Fu C.**, Buffer Control Techniques for Compressed Domain Video Editing, *ISCAS '96*, 1996.

[12] **Sulivan G.J., Wiegand T.**, Video Compression – From Concepts to the H.264/AVC Standard, *Proceedings of the IEEE Volume 93, Issue1*, Jan 2005, 18-31

CURRICULUM VITAE

Ertuğrul Doğan was born in Ayancık in 06.12.1983. After he has completed his elementary and high schools in Nicosia and İstanbul, he has attended Electrical and Electronics Engineering programme at Dokuz Eylül University in 2000. In 2005 he has graduated with the 2nd honors degree. He attended to his graduate study in the programme of Telecommunications Engineering at İstanbul Technical University in 2005, which he is about to graduate at the moment.

He has been working as a Software Design Engineer in Vestek A.Ş since 2005. His interest areas are: Video Coding, Embedded Systems, Real Time Processing, GPGPU (General-Purpose Computing on Graphics Processing Units).