uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Sharmeen Shahabuddin
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Compressed Domain Spatial Adaptation of H.264 Videos
TITRE DE LA THÈSE / TITLE OF THESIS

Shervin Shirmohammadi
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Mojtaba Hosseini
CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Chris Joslin

Abdulmotaleb El Saddik

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Compressed Domain Spatial Adaptation of H.264 Videos

by

Sharmeen Shahabuddin

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-73868-9
Our file Notre référence
ISBN: 978-0-494-73868-9

Abstract

A gigantic amount of multimedia contents is readily available for an ever growing consumer base at present due to advances in video coding technology and standardization along with rapid improvements of storage capacity and computing power. However, today's pervasive media environment which includes heterogeneous terminals and networks presents a serious obstacle in achieving seamless access to these contents. Storing individual content in several formats taking into account a wide variety of possible user preferences and resource constraints or adapting the content on the fly by cascaded decoding/re-encoding is not at all suitable for real-time applications. In this thesis, an innovative spatial adaptation technique for H.264/AVC videos is presented that adapts videos in the compressed domain without decoding/re-encoding them. The proposed adaptation strategy encodes each video frame into several slices according to the proposed slicing strategies. The adaptation process can reduce video size by cropping individual frames by dropping the undesired slices prior to transmitting that video to the clients. In order to ensure codec independence, structured metadata-based adaptation utilizing MPEG-21 generic Bitstream Syntax Description (gBSD) is used to perform adaptation operations. The adaptation process is completely transparent to the client as the adapted video-stream is H.264 standard compliant. A prototype is developed for the proposed scheme. The performance results of the prototype are presented in this thesis. The results show that the proposed spatial adaptation scheme is very suitable and effective for adaptation of both pre-recorded and live video streams.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my thesis supervisor, Dr. Shervin Shirmohammadi, for giving me an opportunity to work in the challenging field of multimedia adaptation and providing scholarly guidance throughout the course of this research. I am extremely thankful to him for providing me with all the necessary facilities and financial support required for this task. This thesis would not have reached its completion without his continued support, patience and encouragement that kept me focused.

I also thank my co-supervisor, Dr. Mojtaba Hosseini, for helping me enhance my knowledge on related concepts and providing invaluable technical guidance.

Special thanks go to Dr. Ali Nazari, Postdoctoral Fellow at the Discover Lab, for his abundant help and insightful comments and suggestions at various stages of writing this thesis.

I also greatly acknowledge the assistance of Razib Iqbal, my colleague at the Discover Lab, in co-writing the published papers that have come from this research.

A special word of gratitude should be dedicated to my husband, Sadrul Habib Chowdhury and my parents and other family members for their endless love, care and support. Last but not the least, I thank God Almighty who has bestowed upon me the strength and patience to successfully complete this research.

Table of Contents

List of Figures

List of Tables

List of Definitions

AVC -	Advanced Video Coding
BSD -	Bitstream Syntax Description
BSDL -	Bitstream Syntax Description Language
CIF -	Common Intermediate Format
DCT -	Discrete Cosine Transform
DI -	Digital Item
DIA -	Digital Item Adaptation
FMO -	Flexible Macroblock Ordering
gBSD -	generic Bitstream Syntax Description
HD -	High Definition
IDR -	Instantaneous Decoding Refresh
ISO -	International Standards Organization
MBAmap -	Macroblock Allocation Map
MC -	Motion Compensation
ME -	Motion Estimation
MPEG -	Moving Picture Experts Group
MV -	Motion Vector
NAL -	Network Abstraction Layer
PPS -	Picture Parameter Sets
PSNR -	Peek Signal to Noise Ratio
QCIF -	Quarter Common Intermediate Format
ROI -	Region of Interest
SEI -	Supplemental Enhancement Information
SPS -	Sequence Parameter Sets
SSIM -	Structural Similarity
SVC -	Scalable Video Coding
UMA -	Universal Multimedia Access
URI -	Uniform Resource Identifier
VCEG -	Video Coding Experts Group
VCL -	Video Coding Layer

XML - Extensible Markup Language

XSL - Extensible Stylesheet Language

XSLT - Extensible Stylesheet Language Transformation

Chapter 1 : Introduction

1.1 Motivation

In recent years video applications in both wired and wireless environments have experienced an exponential growth. Due to advances in video coding technology and standardization as well as the improvements of storage capacity, and computing power, we now have a rich variety of devices capable of supporting a wide range of video applications such as multimedia messaging, video telephony, video conferencing, Internet video streaming, standard and high-definition TV broadcasting etc. Even video applications for mobile devices such as mobile video streaming, mobile video conferencing and mobile TV broadcasting are feasible these days with the advent of 3G Networks and advanced mobile devices. Simultaneously, the rapid development of network infrastructures has enabled a vast amount of multimedia contents to be accessible through various networks. However, due to the heterogeneity of devices in terms of screen resolution, processing power or memory and bandwidth limitations and diverse user preferences, seamless access to these contents is being hampered, thus failing to conform to the vision of Universal Multimedia Access (UMA) which refers to the ability to access any multimedia content by any user over any type of network with any device anywhere anytime.

Video adaptation, with the objective of maximizing the utility of the adapted video based on user preferences and satisfying diverse resource constraints, has emerged as a promising approach for accomplishing the goal of UMA. There are many interesting adaptation techniques that can be clustered into two categories -adaptation by selection and adaptation by transformation. In the former approach, multiple versions of a single video are stored in the server and the server selects the most appropriate version based on the specific capabilities of the terminal devices, network conditions or user preferences. The greatest advantage of this approach is that the adaptation process is very quick and requires a small amount of computational resources. On the other hand, considering the large number of varying user preferences, this approach will incur a great storage cost on the server. Other problems include the unavailability of the appropriate content that fits the user condition and increased work from the author to manage the content. Moreover, if some changes are made to the original content, all the variations also need to be changed, which requires a lot of time and power.

Consequently, this is not a feasible solution. A smarter approach is adaptation by transformation where a single multimedia content is transformed into an adapted multimedia content on-the-fly. The main advantage of this approach is low storage cost. In addition to this, changes to the original content can be made easily compared to the previously mentioned approach. The main drawback of this approach is that it may require a lot of processing resources to adapt the video.

A video content can be adapted in terms of spatial resolution, temporal resolution, quality/Signal-to-Noise Ratio (SNR), modality, content length, coding format, coding parameters, presentation, color, properties etc. to maximize the experience of the consumer. Although there have been a lot of research activities in this field, majority of the approaches developed focus on temporal resolution reduction i.e. frame rate reduction. The limitations of display capability and network bandwidth are still critical factors in mobile applications although the limitation of computation power is greatly improved recently. It is very much desired for the users of devices with limited display size to have an enhanced viewing experience. This is where the motivation behind this thesis lies, with the need to come up with a technique for spatial adaptation by transformation that will address the concept of UMA. With this purpose in mind, a novel scheme for spatial adaptation of H.264 videos has been developed and presented in this thesis.

1.2 Research Problem and Objective

Although the recent advances in mobile device capabilities and wireless networks have enabled a new array of applications in video, it is still difficult to render even Standard-Definition videos to multimedia enabled mobile devices due to their resource limitations in terms of screen resolution, processing power, and bandwidth. An easy solution to adapt a high resolution video to heterogeneous devices is to first completely decode the video, and then re-encode it according to the requested preferences. This way, it would be possible to store only one version of the high quality video on the server and adapt the videos on request before delivery. However, this cascaded decoding/re-encoding process can be very computationally expensive. The excessive computation can have an adverse effect on the delivery time of the video and may render this process infeasible for real-time video delivery. Therefore, the problem of efficiently adapting video data in real-time to the specific network and device context of the user remains an open issue and subject to extensive research.

One solution for the adaptation process would be to adapt the video on request without completely decoding the video first. In other words, the adaptation process can generate the adapted video with the requested parameters from the original high-resolution encoded video in compressed domain.

The main objective of this thesis is to devise such a process for spatial adaptation. The proposed adaptation process will crop a video in the compressed domain and deliver only the region of interest (ROI) to the user. The adaptation process should be able to adapt videos to include regions of different dimensions. It should also conform to ISO/IEC MPEG-21 Part-7 Digital Item Adaptation specifications.

1.3 Research Contributions

This thesis contains a number of contributions:

- A novel scheme for real-time spatial adaptation of video in the compressed-domain.
- Innovative slicing strategies for dividing frames into slices that more efficiently map the video to heterogeneous devices.
- Implementation of a prototype and performance evaluation as proof of concept and validation of the proposed adaptation technique, for both live and stored videos.

1.4 Scholastic Output

- Peer-reviewed publications:
    i. R. Iqbal, *S. Shahabuddin*, S. Shirmohammadi, "Compressed-Domain Spatial Adaptation Resilient Perceptual Encryption of Live H.264 Video", Proc. International Conference on Information Sciences, Signal Processing and their Applications, Kuala Lumpur, Malaysia, May 2010
    ii. *S. Shahabuddin*, R. Iqbal, A. Nazari, S. Shirmohammadi, "Compressed Domain Spatial Adaptation for H.264 Video", Proc. ACM Multimedia, Beijing, China, October 19 - 24, 2009
    iii. R. Iqbal, *S. Shahabuddin*, S. Shirmohammadi, "Compressed-Domain Spatio-Temporal Adaptation System for Video Delivery", Proc. ACM Multimedia, Beijing, China, October 19 - 24, 2009 (Demo)

1.5 Organization of Thesis

The rest of the thesis is organized as follows:

- **Chapter 2** introduces the reader with video adaptation, MPEG-21 Digital Item Adaptation and H.264 video coding standard. It also reviews and analyzes existing techniques for spatial adaptation in literature.
- **Chapter 3** proposes the spatial adaptation scheme for H.264 videos. Different aspects of the proposed scheme are also analyzed in details in this chapter.
- **Chapter 4** details a prototype for the proposed adaptation process. It also evaluates the performance of the proposed scheme in different performance criteria.
- **Chapter 5** summarizes the thesis, and puts forward recommendation for future research in this area.

Chapter 2 : Background: Video Coding and Adaptation

2.1 Video Adaptation

Video adaptation is a promising field which offers a rich set of techniques for facing the challenges of pervasive media environment which includes different types of terminals and network infrastructures. Usually adaptation tools take into account the content characteristics, device requirements, usage environments, and user preferences to adapt a video content. Adaptation engines are typically deployed in the intermediate locations between the server and the client, such as proxy servers, although they may be included in the servers or clients in some applications.

There are several techniques for adapting videos to satisfy diverse resource conditions or user preferences. As discussed in Chapter 1, these techniques can be roughly categorized into two groups - adaptation by selection and adaptation by transformation.

Adaptation by selection uses pre-stored versions of different qualities and/or modalities of the same multimedia content and delivers the most suitable version depending on the usage environment. This approach is very quick and computationally inexpensive but requires a lot of storage for multiple versions of the content. One of the first approaches of this adaptation paradigm is the InfoPyramid framework developed by IBM in 1998 [1]. This framework provides a multimodal, multi-resolution representation hierarchy for multimedia content. On the other hand, Adaptation by transformation transforms the multimedia content on-the-fly. In this approach the storage requirement is very low and video can be adapted in real-time. Some adaptation by transformation operations are described in the following:

- **Format Transcoding:**

  This is a basic adaptation process for transcoding video from one format to another (e.g. MPEG-2 to MPEG-4). This approach is applied to make the video content compatible with the new usage environment. [2]

- **Coding Parameter Transcoding:**

  This is another form of transcoding which is used in a constrained environment to trade some components of the video entity such as the bitrate (e.g. 6.0 Mbps of TV broadcast to 128 kbps

for mobile phones), frame-rate (e.g. 30 fps to 24 fps), or spatial resolution (e.g. CIF to QCIF) for saving some resources (such as machine power) or to meet resource constraints (such as limited display capabilities). [2]

- **Transmoding:**

The objective of this adaptation process is to modify the modality (e.g., audio, video, image, text) in order to satisfy transmission constraints and/or user preferences. Examples of this approach include speech-to-text conversion, video-to-image conversion, image-to-text conversion etc. [2]

- **Scalable Coding:**

In recent times, scalable coding technique is emerging as a very important tool for adaptation. In scalable coding, the video is encoded once and then by simply truncating some layers or bits from the original stream, videos of lower qualities, spatial resolution, and or temporal resolution could be obtained. One example is the scalable extension of H.264, Scalable Video coding (SVC) [3], which is still in the process of being standardized. [2]

2.1.1 Spatial Adaptation

In recent years, the ability to create high resolution video sequence is has become more commonplace, but not all devices support display of frames at such resolutions. This necessitates spatial video adaptation schemes to overcome display constraints for such devices. Spatial adaptation can be done in two ways: scaling the video to a particular display resolution or cropping a particular region form the video stream. For scaling the video, several techniques have been proposed in the literature and some of the techniques for H.264 videos have been discussed in section 2.4.

When cropping a desired region from the video, it is impractical to crop the video during (or before) encoding, because that would require having multiple encoded videos for multiple cropped regions. Having the ability to crop a compressed video can allow a server (or intermediate nodes) to serve different regions based on client demands from the same encoded high-resolution video stream. But cropping in the compressed domain is very difficult. This can be done in several ways. One way is to transcode the video i.e. to decompress the video, crop the desired regions and then recompress it before serving to the client. While simple, this approach requires significant computational complexities and may reduce the quality of the

video stream by recompressing the data which makes it quite infeasible for real time applications. Another approach is to create many smaller regions of the video and compress them separately thus creating a mosaic of video streams. In this approach it is easier to pick particular regions from the video. However, it requires the application to synchronize and merge together multiple video streams for display, which lead to added complexities. In this thesis, I present a cropping technique which can be executed in the compressed domain without decoding the video.

Due to the vast importance of media adaptation applications, several standards have been developed to facilitate deployment and interoperability of adaptation technologies. The most notable ones include MPEG-7 [4] and MPEG-21 [5]. Different standards are targeted at different applications. In the following section, I give a brief overview of the MPEG-21 standard.

2.2 MPEG-21

MPEG-21 Multimedia Framework provides a way of using multimedia resources across a wide range of networks and devices. To this purpose, the framework defines the mechanisms and elements required to exchange, access, consume, and manipulate content in an efficient and interoperable way. The framework terms each individual element as *Digital Items (DI)*. MPEG-21 formally defines a Digital Item as *a structured digital object with a standard representation, identification and meta-data within the ISO/IEC 21000 framework* [5]. Various aspects of the framework, including how Digital Items should be declared, or how these items can be manipulated or adapted, are described in the following parts:

- Part 1: Vision, Technologies and Strategy
- Part 2: Digital Item Declaration
- Part 3: Digital Item Identification
- Part 4: Intellectual Property Management and Protection Components
- Part 5: Rights Expression Language
- Part 6: Rights Data Dictionary
- Part 7: Digital Item Adaptation
- Part 8: Reference Software
- Part 9: File Format
- Part 10: Digital Item Processing

- Part 11: Evaluation Methods for Persistent Association Technologies
- Part 12: Test Bed for MPEG-21 Resource Delivery
- Part 13: Scalable Video Coding
- Part 14: Conformance Testing
- Part 15: Event Reporting
- Part 16: Binary Format
- Part 17: Fragment Identification of MPEG Resources
- Part 18: Digital Item Streaming
- Part 19: Media Value Chain Ontology

More details about these parts are available at [5].

2.2.1 Digital Item Adaptation

Adaptation of Digital Items is key to accomplishing the goal of interoperable transparent access of multimedia content across a diverse variety of networks and devices. Part 7 of the MPEG-21 Multimedia Framework explains the adaptation of Digital Items [6]. This concept of adaptation is presented in Figure 2.1. As shown in the figure, both a descriptor adaptation engine and a resource adaptation engine can operate on a Digital Item, i.e. media content, and produce the adapted digital item.

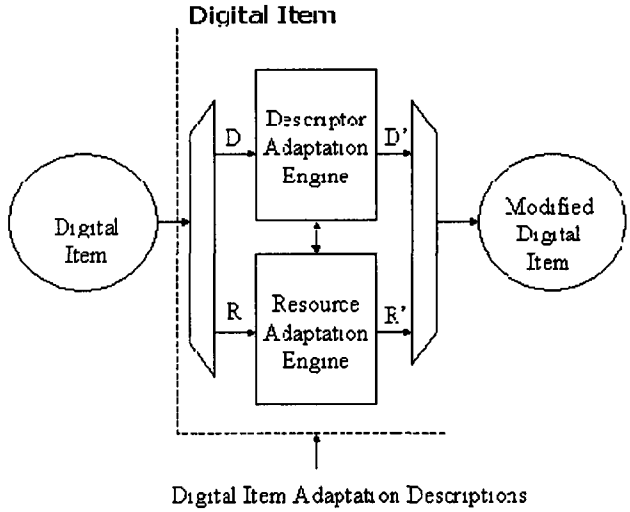


Figure 2.1: Digital Item Adaptation (DIA) [6]

To facilitate the adaptation, MPEG-21 standardizes the descriptions and format-independent mechanisms that provide support for adaptation of resource, description etc. These descriptions and mechanisms are collectively referred to as Digital Item Adaptation Tools.

Digital Item Adaptation tools are clustered into three major categories:

- **Usage Environment Description Tools**

  These tools provide information about user and device capabilities, network and environment characteristics.

- **Digital Item Resource Adaptation Tools**

  These tools include tools that describe the bitstream syntax, tools that describe relationships between network and environment constraints and content quality. There are also tools that allow filtering and scaling XML instances and reduce complexity. These tools target the adaptation of resources in a Digital Item.

- **Digital Item Declaration Adaptation Tools**

  The tools in this category are useful for adapting the Digital Item Declaration and/or providing information required for the configuration of Digital Item Adaptation Engines.

2.2.2 Bitstream Syntax Description [6]

A bitstream is a representation of binary media resource. The Bitstream Syntax describes the organization of the binary data in the bitstream. This structure of the binary data is specific to the coding format. An XML document can provide a high-level description of the bitstream. This XML document is called the Bitstream Syntax Description. This description is not a replacement for the original binary media resource; rather it is an additional layer of information, similar to metadata that describes the organization of the bitstream in different conceptual tiers or packets of data. This Bitstream Syntax Description (BSD) is scalable, i.e. the level of details of the bitstream can vary with the application. The Bitstream Syntax Description is also adaptable, i.e. it can be adapted to properly reflect an adapted bitstream.

A Resource Adaptation Engine can transform the Bitstream Syntax Description and generate an adapted bitstream. The transformation can be facilitated by an XSLT style sheet. This way, Bitstream Syntax Description and related Transformations can be exchanged in the form of XSLT style sheets.

To achieve the goal of complete interoperability, it is necessary that a processor can produce a Bitstream Syntax Description or generate a bitstream from its description without being aware of the specific coding format used to produce the bitstream. For this purpose, a language based on W3C XML Schema is specified in the DIA Framework, called Bitstream Syntax Description Language (BSDL). This language can be used to describe the syntax of particular coding formats in documents called Bitstream Syntax Schemas. A generic processor can process these schemas to automatically parse a bitstream and generate its description, and vice-versa.

The Bitstream Syntax Schemas described by BSDL are codec specific schemas, i.e. an adaptation engine is required to know the specific schema for a particular bitstream. This may be undesirable in some instances, for example when the adaptation is performed on a device with limited resources. For this reason, a codec-independent schema can be useful. The DIA Framework specifies a codec-independent schema for describing a bitstream syntax, called generic Bitstream Syntax Schema (gBS Schema).

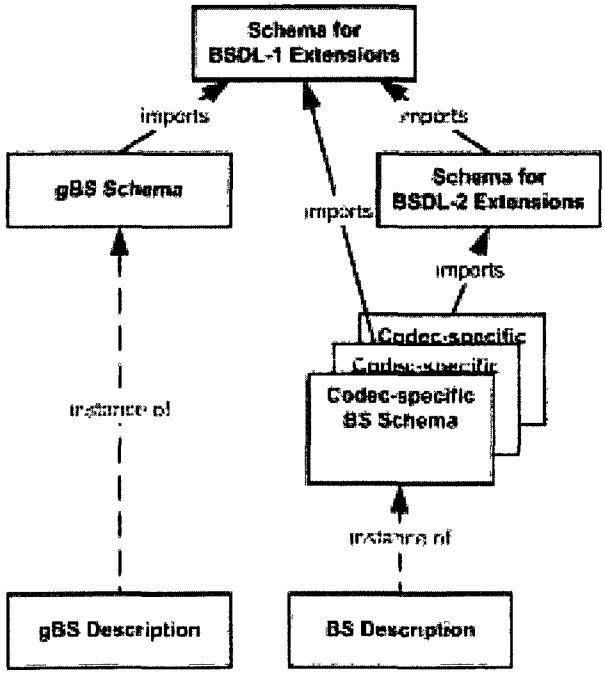


Figure 2.2: BSD Schema Hierarchy[6]

The gBS Schema provides the following functionalities:

— **Codec Independence**. The gBS Schema can be used to describe any bitstream in a codec independent manner. Codec specific schemas are not required to re-generate the bitstream from a gBS Description (gBSD).

— **Semantic Marking**. The gBS Schema allows syntactical elements to be marked using the `marker` handle. These `markers` allow semantically meaningful and efficient adaptations of bitstreams.

— **Hierarchical Descriptions**. The gBS Schema can describe a bitstream in hierarchical fashion. This allows bitstream elements to be grouped for efficient and hierarchical adaptations.

2.3 H.264/AVC

H.264, also referred to as MPEG-4 Part 10 or Advanced Video Coding (AVC), is the most efficient video compression standard available today. It was jointly developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) in 2003. It offers significant improvements over the previous coding standards in terms of compression efficiency, perceptual video quality, network friendliness and versatility. [7]

H.264 provides enhanced coding efficiency for real-time videoconferencing, digital video broadcasting, direct-broadcast satellite television service, cable television services, third-generation mobile telephony, digital cinema, video surveillance and military applications. It is mandatory for the HD-DVD and Bluray specifications, the two formats for high-definition DVDs. Videos from YouTube and the iTunes Store are also available in H.264. According to encoding.com, the leading global provider of studio-class video services for video sites and website development platforms, the H.264 format now makes up 66 percent of web videos and is now the largest format by far [8].

H.264 is designed to have a two-layer structure to distinguish coding-specific features from transport-specific features. The two layers are - Video Coding Layer (VCL) and Network Abstraction Layer (NAL). The VCL produces a coded representation of the video content and the NAL formats the VCL data and provides header information in a way appropriate for adaptation to various transportation protocols or storage media. [7]

2.3.1 *Video Coding Layer*

Similar to the previous video coding standards since H.261, the VCL of H.264 employs the block-based video coding technique where a coded frame/picture is partitioned into a number of fixed-size macroblocks, which are blocks of 16x16 Luma samples with associated chroma samples (8 × 8 Cb and 8 × 8 Cr samples). Luma represents the brightness in an image, whereas the chroma components represent the color information. H.264 uses the YCbCr colour space where Y is the luma component and Cb and Cr are color difference (chroma) components - blue minus luma (B-Y) and red minus luma (R-Y) respectively. Due to human visual system's greater sensitivity towards brightness levels than color, H.264 employs 4:2:0 sampling structure where each chroma component has one fourth of the number of the samples compared to the luma component. [7]

A sequence of coded pictures forms a coded video sequence. Within a coded video picture macroblocks are grouped into one or more self-contained slices, which can be parsed from the bit stream and decoded independently of other slices, given that the reference picture used are the same at the encoder and the decoder. [7]

- **Encoding and Decoding Process**

Instead of explicitly defining a CODEC (enCOder / DECoder pair), H.264 defines the syntax of the encoded bitstream and the process of decoding this bitstream. The basic functional elements of a H.264 compliant encoder are – prediction, transform, quantization and entropy coding. Figure 2.3 shows a high level architecture of a H.264 compliant encoder.

In this block-based encoding process a video frame is processed in units of macroblocks. Each macroblock is encoded in either intra or inter mode. In intra mode a prediction of the macroblock is constructed from the samples in the current slice that have already been encoded, decoded and reconstructed. For the luma components, prediction is formed for each 4 × 4 block for coding of parts of the picture with significant details or for a 16 × 16 macroblock for coding smoother parts. For the chroma components, intra-prediction is always applied on a macroblock basis. There exist a total of nine optional prediction modes for each 4 × 4 luma block, four modes for a 16 × 16 luma block and four modes for the chroma components. In most of the cases, the prediction mode which minimizes the difference between the prediction and the block to be encoded is chosen by the encoder.

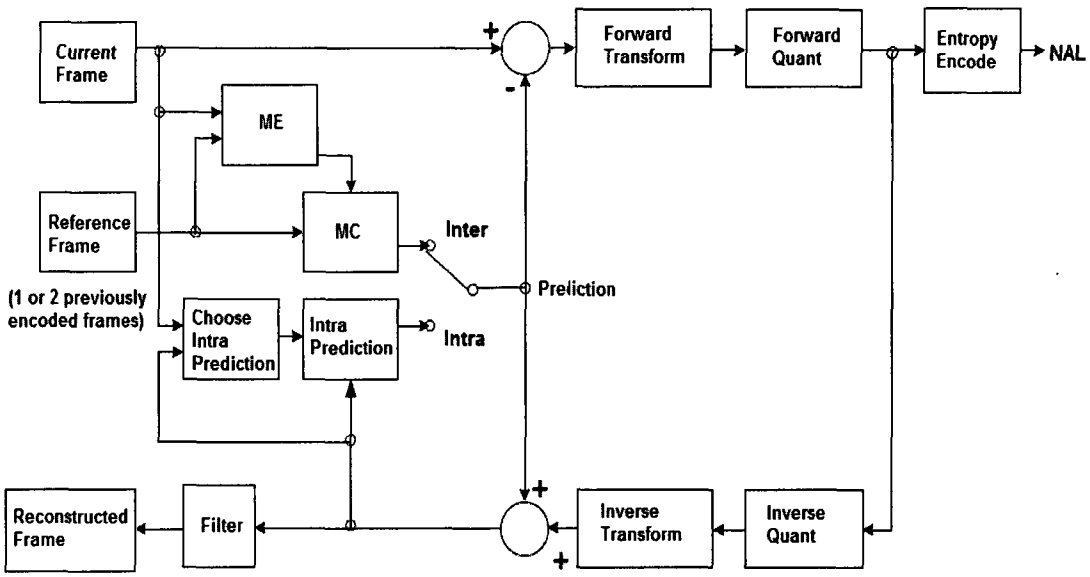


Figure 2.3: H.264 Encoder[9]

On the other hand, in inter mode the prediction is constructed from one or two previously encoded pictures known as reference pictures using block-based Motion Compensation (MC). The inter prediction process may split up the luma component of each macroblock in four ways - either as one 16×16 macroblock partition, two 16×8 partitions, two 8×16 partitions or four 8×8 partitions. Each 8×8 sub-macroblocks can further be partitioned either as one 8 × 8 sub-*macroblock partition, two 8×4 or 4×8 or four 4×4 sub-macroblock partitions. For each block,* the most similar block of the same size and shape is found in the reference pictures. This process is known as Motion Estimation (ME). The chosen region becomes the predictor for the current block and is subtracted from the current block to form a residual block and this process is known as motion compensation. The offset between the current block and the position of the *candidate block is known as Motion Vector (MV).* [9]

The residual of the prediction (either Intra or Inter mode), which is the difference between the current and the predicted block, is then transformed using a 4x4 or 8x8 integer transform, an approximate form of the Discrete Cosine Transform (DCT). The transform converts the samples into frequency domain in which they are represented by transform coefficients, each of which is a weighting value for a standard basis pattern. The coefficients are then quantized to remove insignificant values, leaving a small number of significant coefficients that can be encoded with a

small amount of data. Finally the quantized coefficients together with the side information for either Intra-frame or Inter-frame prediction are converted into binary codes using entropy coding. The encoded bitstream is then passed to the NAL for transmission or storage. [9]

At the decoder the coding process is reversed to recreate the video sequence. The decoder entropy decodes the received compressed bitstream to produce a set of quantized coefficients which are then scaled and inverse transformed to recreate a residual macroblock. For each macroblock, the decoder forms an identical prediction to the one created by the encoder using the decoded header information from the bitstream. The prediction is then added to the decoded residual. The result of that addition is filtered to reconstruct a decoded macroblock which can then be displayed as part of a video frame. The encoder also includes the decoder to conduct prediction. [9]

- **Slice Types and Slice Groups**

In H.264 there are five types of coded slices and a coded picture may contain different types of slices. The simplest one is the I (Intra) slice. In this slice all the macroblocks are coded using intra prediction mode. P (Predicted) slice contains macroblocks coded using inter prediction mode with one prediction signal for each predicted region and/or macroblocks coded using intra prediction mode. B (Bi-predictive) slice contains macroblocks coded using inter prediction mode with two prediction signals that are combined with a weighted average to form the predicted region and/or macroblocks coded using intra prediction. The other two types, SP (Switching P) and SI (Switching I) slices, are special types of slices which facilitate switching between coded streams. [9]

Macroblocks are usually processed in raster scan order within a frame. There is a special feature of H.264 called Flexible Macroblock Ordering (FMO) which offers flexibility to change this coding order of macroblocks by assigning the macroblocks in a picture to one or several slice groups (i.e., a set of slices) using a Macroblock Allocation Map (MBAmap) specified by the content of the Picture Parameter Set (PPS) which is described in the next section and some information from slice headers. This map consists of a slice group identification number for each macroblock in the picture, specifying which slice group the associated macroblock belongs to. H.264 allows up to eight slice groups in one picture. Macroblocks are coded in default scan order within a slice group and can be grouped into several slices. [10]

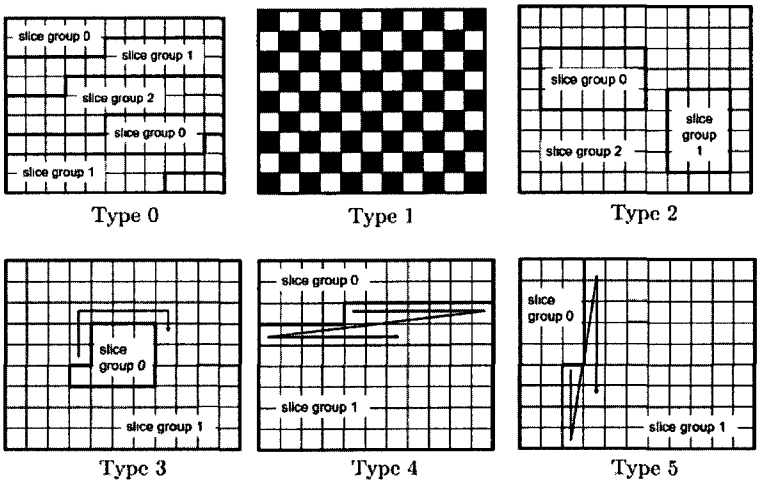


Figure 2.4: Types of FMO[10]

There are seven types of FMO specified in H.264. Examples of six types are illustrated in Figure 2.4. In case of FMO type 0, each slice group has a maximum number of macroblocks in raster scan order before another slice group is started. FMO type 1, known as scattered slices or dispersed slices, uses a mathematical function which is known at both the encoder and the decoder to spread the macroblocks. FMO type 2 uses one or more rectangular slice groups and a background. The slice groups can be overlapping. The coordinates of each of the rectangular slice groups are saved in the MBAmap. FMO types 3, 4, and 5 are the evolving slice groups where the configuration of slice groups can change over different pictures in a cyclic way. In case of FMO type 6, the entire map is coded into a picture parameter set. In rest of the six types, the coded representation of the map is much smaller due to certain patterns in the MBAmap. [10]

2.3.2 Network Abstraction Layer

A coded H.264 video sequence consists of a series of Network Abstraction Layer (NAL) units, each of which is a packet containing an integer number of bytes. Each NAL unit has payload data of some specific kind and a byte long header which indicates that type. There are two types of NAL units - VCL and non-VCL NAL units. The VCL NAL units contain coded slices or coded slice data partitions and the non-VCL NAL units contain any associated additional information such as parameter sets and Supplemental Enhancement Information (SEI). A parameter set contains

information that is unlikely to change and offers the decoding of a large number of VCL NAL units. There are two types of parameter sets - Sequence Parameter Sets (SPS), which apply to a series of consecutive coded video pictures called a coded video sequence and Picture Parameter Sets (PPS), which apply to all slices of one or more pictures within a coded video sequence. Every conforming H.264/AVC bit stream requires that at least one SPS and one PPS are available. Each VCL NAL unit contains a pointer to the relevant PPS in use and the PPS, in turn, has a reference to the SPS in use at that time. SEI includes timing information and other supplemental data that are not necessary for decoding the values of the samples in the video pictures but may enhance usability of the decoded video signal. A set of NAL units in a specified form is referred to as an access unit which is decoded to get one decoded picture. A set of access units that are sequential in the NAL unit stream and use only one SPS is referred to as a coded video sequence. Each coded video sequence can be decoded independently, given the necessary parameter set information. It always starts with an Instantaneous Decoding Refresh (IDR) access unit. An IDR access unit which contains an intra picture indicates the subsequent access units and itself can be decoded without decoding any previous pictures of the bit stream. [7]

2.4 Spatial Adaptation Techniques for H.264

There are several proposals for adding spatial adaptation capability in H.264 videos. Peng Zhang et al present a spatial adaptation technique in [11] that uses rescaling (downscaling) for adaptation. This scheme employs a kind of Cascaded Pixel Domain Transcoding (CPDT), i.e. it first decodes the compressed video. The decoded video is then downscaled and re-encoded using a mode-mapping method that makes the encoding process faster. While encoding, H.264 encoders usually try all the possible coding modes for each macroblock, and selects the best one. Since this is an expensive operation for a transcoder, in [11] the mode information of the macroblocks in the original encoded video is used to estimate the mode of the corresponding macroblock in the transcoded bitstream. A similar downscaling scheme is presented in [12]. In addition to computing the partition mode of the downsized macroblocks from the macroblocks in the original video, this scheme also computes the motion vector (MVs) of the low-resolution stream by scaling down the motion vectors of the corresponding blocks in high resolution. A similar approach where macroblock mode and motion vectors in the downscaled video are computed from the original encoded video to speed up the re-encoding process during transcoding is presented in [13].

Houqiang Li et al present an attention information based spatial adaptation framework for browsing videos via mobile devices in [14, 15] that use cropping for adaptation. In their proposal, the ROI is detected when encoding the video. The authors propose a new Supplemental enhancement information (SEI) header to encode this information about the ROI into the bitstream. The adaptation engine decodes the compressed video to extract the ROI information from this SEI header. This information is then used to generate the cropped video, which is then re-encoded and delivered to the client. In [16], authors use a similar transcoding process for cropping H.264 videos. Additionally, this scheme reduces transcoding complexity by using a special process for encoding SKIP mode macroblocks in the original video. When re-encoding these macroblocks in the cropped video, the transcoder compares the motion-vector predictors (MVP) for the macroblock in the original video with the computed motion vector predictors in the cropped video. If the MVPs are the same, then SKIP mode is selected for the macroblock. Thus, the transcoding complexity is reduced by avoiding the expensive rate-distortion optimization process to detect the macroblock modes for SKIP mode macroblocks in the input video.

All the above mentioned adaptation techniques use cascaded transcoding, i.e. the video is first decoding, completely [11, 12, 13] or partially [14, 15] and then re-encoded. This is usually computationally very expensive. A spatial adaptation technique that can adapt the video in compressed domain, i.e. generate the adapted video without having to first decode it, will have a significantly improved performance, and require a lot less resources. It will be especially useful for adaptation of live video streams.

In [17] Peter Lambert et al use FMO for coding arbitrary-shaped Regions of Interest (ROIs). For content adaptation, they replace the undesired slices with placeholder slices. A placeholder slice is a P-slice where skipped mode is used for all the macroblocks in it. They also use Bitstream Syntax Description (BSD), as defined in MPEG-21 framework, to describe the encoded bitstream. This BSD is later used during adaptation. The adapted video using this approach has the same resolution and dimension as the original video. In [18], the authors present an extension of this approach. In this new approach, the authors use the cropping flags in the Sequence Parameter Set (SPS), as defined by the H.264/AVC standard, to reduce the size of the adapted video. In this scheme, the decoder still has to process the placeholder slices, although it does not display

them to the client. An approach where only data that are essential to display the video is delivered to the client would be preferable. Also, in this approach, the selection of the ROI during encoding the video is not clearly defined. So a well-defined way of integrating ROI detection algorithms into the adaptation framework would be a significant improvement. FMO is not currently supported by the most popular video decoders, e.g. mplayer and ffmpeg. So a spatial adaptation technique that can crop a video using only the most widely supported features of the H.264/AVC standard will be more acceptable for commercial vendors.

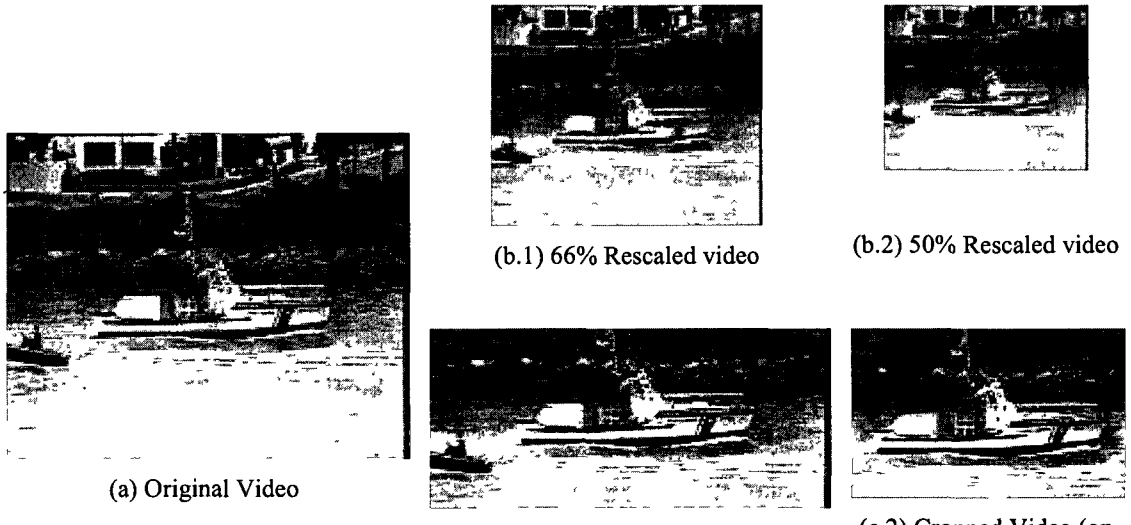
Therefore, I investigate compressed domain spatial adaptation technique that integrates ROI detection algorithms into the scheme and minimizes the resource consumption, and propose a scheme for this purpose in this thesis.

In this chapter, several adaptation approaches with the focus on spatial adaptation have been discussed. An overview of the related standards, H.264 and MPEG-21, is given. In addition, some of the existing spatial adaptation techniques for H.264 videos are also mentioned and described briefly.

Chapter 3 : Proposed Scheme for Spatial Adaptation

As discussed in Chapter 2, there are two possible techniques for spatial adaptation: Rescaling and Cropping. In this thesis, I look at the latter option, i.e. cropping. The main reasons for choosing cropping over rescaling are described below.

First, in a rescaled video, both dimensions of the adapted video, i.e. the width and height, will have to maintain the same ratio with the dimensions of the original video, i.e. for an original video of dimension of $W \times H$, the rescaled video will have a dimension of $W/K \times H/K$, for some real number K. Not maintaining this ratio will produce low-quality video that can lead to poor perceptual experience. On the other hand, the dimension of a cropped adapted video is dictated by the region of interest, and not by the dimension of the original video.



(a) Original Video

(b.1) 66% Rescaled video

(b.2) 50% Rescaled video

(c.1) Cropped video (top and bottom)

(c.2) Cropped Video (on all sides)

Figure 3.1: Rescaling vs. Cropping

Second, in an adapted video after rescaling, the resolution of the region of interest (ROI) is reduced. As a result, although the adapted video contains all the contents in the original video, it loses a lot of the details of the ROI. Cropping, on the other hand, keeps the details of the region of interest in the adapted video, which can often be more desirable than the rescaled content of the entire video. For example, for security videos, it is important to keep the details

of the region of interest, while the region with no activity, i.e. the region of no interest (RONI) can be cropped without loss of any value.

Both of these differences are illustrated in Figure 3.1. When the sample video is rescaled, there is only a rescaling factor that determines the dimension of the rescaled video. With cropping on the other hand, amount to be cropped can be adjusted on all four sides of the video, and this provides a greater range of options for spatial adaptation. Also, the resolution in the cropped video is the same as in the original video, whereas the resolution in the rescaled video is less, since it is also rescaled by the same factor as the dimension.

3.1 Slice Coding

The compressed video format of H.264 does not immediately lend itself to cropping. It is very difficult to extract a particular region in a video frame from an H.264 encoded video without decoding it first, unless some special technique is used while encoding the video. This is because the location and dimension of a region in the frame represented by a portion of the bitsream is compressed into the bitstream itself. To allow cropping, we take advantage of a feature in H.264 standard: Slice coding, which was introduced in H.264 primarily as a mechanism for mitigation of errors, packet losses and network variability. A *Slice* is a spatial segment in a video frame. Each picture in the video can be divided into one or more slices. If there is an error or missing data in a slice, then they cannot be propagated to the other slices within the picture. Slice coding can thus reduce the impact of errors. Each slice is encoded into one Network Abstraction Layer Unit (NALU) in the compressed bitstream.

To make it possible to crop H.264 videos for spatial adaptation, we utilize slice coding. During the encoding process, each picture in the video is divided into slices of two categories: slices that contain the region of interest, and slices that contain the region of no interest. An adaptation engine can later extract from the H.264 videos encoded using this scheme the slices that contain the regions belonging to the ROI and thus generate the adapted video.

3.2 Encoding

The spatial adaptation process begins with encoding. The encoding process takes into consideration the region of interest in the video. It then encodes the ROI into one or more slices.

The rest of the picture is also encoded into one or more slices. Thus each frame has at least two slices. The slices that contain the spatial data for the region of interest are called *essential* slices. The rest of the slices are called *disposable* slices. The essential slices are always served to the client, irrespective of the adaptation performed on the video. The disposable slices are included in the adapted video only if the client requests for video data of that region. Thus, it is possible that only some disposable slices are included in the adapted video, while a number of disposable slices are not. This mechanism makes it possible to generate videos of varying degree (or granularity) of spatial adaptation, e.g. some client can be served only the essential slices belonging to the region of interest, while at the same time another client can be served the essential slices and some disposable slices surrounding the ROI.

The exact number of slices used to encode the ROI and the RONI can vary, depending on the nature of the ROI, e.g. dimension and location of the ROI in the picture. It can also depend on the desired granularity of adaptation. The selection of slices is explained in greater detail in later subsections.

3.2.1 Region of Interest

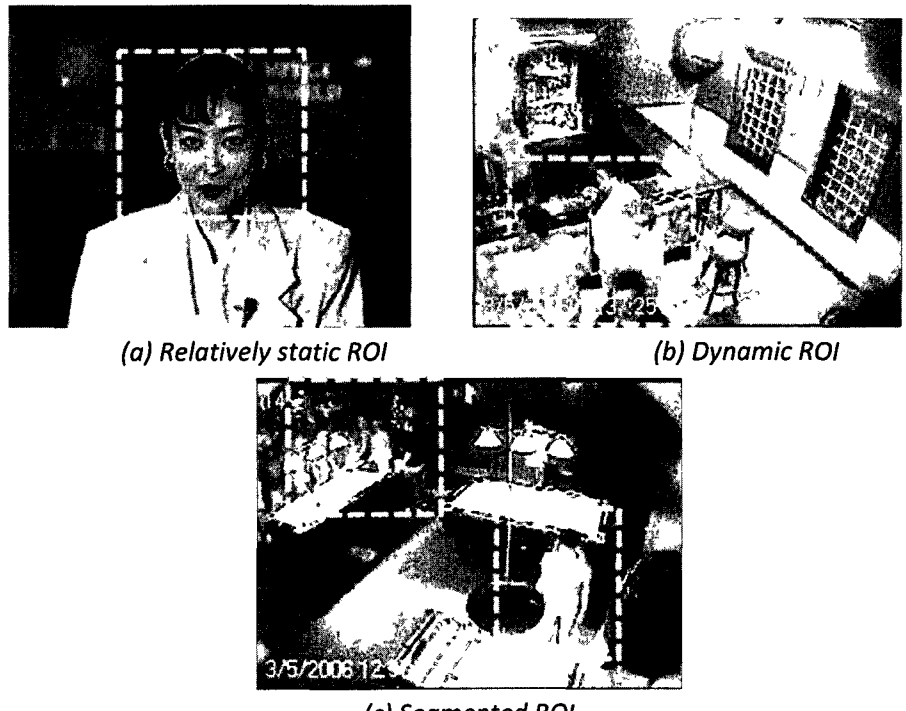
One important aspect of spatial adaptation is detecting the region of interest in the video. The ROI should not be cropped out of the adapted video, because that would defeat the purpose of serving the video in the first place. For the spatial adaptation technique presented in this thesis, the region interest must be enclosed into a single rectangular region in the video, and the encoding process needs to be aware of this region. Based on the knowledge of the ROI, the encoding process creates the essential slices and the disposable slices, ensuring that the ROI is always encoded into essential slices.

There are various algorithms that can be used to compute the region of interest in a video sequence. Some algorithms try to predict the portions in the scene that will involuntarily attract visual attention based on available saliency information. A user attention model for video skimming and summarization presented in [19, 20] utilizes audio-visual features, for example, motion, speech, camera operation etc. In [21], a framework for video focus detection is proposed where different parameter sets are optimized for different genres of videos for computing the saliency information required for computing the ROI. The framework presented

in [22] is based on a visual attention model that uses intensity, color and motion as the visual attention features. There are also algorithms that determine the ROI in a video based on face detection [23].

3.2.2 ROI Detector

The spatial adaptation scheme proposed in this thesis can be used with any ROI detection algorithms. The ROI detection algorithm is implemented as an ROI Detector module. The module may implement any of the available ROI detection algorithms. By having a separate module for ROI detection, it is possible to easily and seamlessly switch the algorithm used to detect the ROI in the video. This makes it possible, for example, to use the ROI detection algorithm in [23] when encoding for a video conference, and use [22] for a video of a sporting event by simply plugging in the relevant ROI Detector module in the encoder.



(a) Relatively static ROI

(b) Dynamic ROI

(c) Segmented ROI

Figure 3.2: Variability of Region of Interest

The ROI detected in a video by the ROI detection algorithms can change in location and dimension during the playtime of a video. However, it is important during the display of a video that the video does not change its dimensions, i.e. it is desirable that a video maintains the same

width and height during the entire length of its playtime. Therefore, it is necessary to enforce a constant dimension for the ROI. To this end, an additional module is introduced to the encoding process. This module reads the region of interest from the ROI Detector module, and modifies the dimension of the region, if necessary, to ensure a constant width and height for the region of interest. This module is named ROI Distiller.

3.2.3 ROI Distiller

The desired fixed dimension of the region of interest in the video varies according to the nature of the video. For example, during a video conference, the region of interest is not expected to change greatly. On the other hand, in a surveillance video, the region of interest can change constantly, both in location and in dimension.

The ROI Detector module detects the region of interest as a rectangular region in the video frame. The ROI Distiller module takes this region as input R, and the desired fixed dimension of the ROI in the adapted video, and produces a Refined Region of Interest (RROI) that includes R in it. In the event of multiple regions of interest in the picture, or segmented ROIs, a larger rectangle is created to encompass all such regions. This is illustrated in Figure 3.2 (c).

The functionality of ROI Distiller module is described in pseudocode in Table 3.1. The first loop in the subroutine determines the smallest rectangle to enclose all the detected ROIs by the ROI Detector. It then compares the dimension of this rectangle with the desired dimension. If the width of the computed rectangle is smaller than the desired width, then the rectangle is increased by equal amount on both sides. However, if the width of the computed rectangle is larger than the desired width, then the rectangle is decreased by equal amount on both sides to ensure that the region fits the exact desired width. Similarly, the height of the computed rectangle is adjusted to match exactly with the desired height. This refined rectangle is then produced as the output of this module.

In the subsequent sections, this refined region of interest is implied as the sole Region of Interest. The ROI Detection and ROI Refinement processes presented in this chapter are presented primarily as a guideline. The main focus of this thesis is concentrated on defining

several slicing strategies that can be used to divide each frame into slices that can lead to efficient cropping from compressed video. These slicing strategies are presented next.

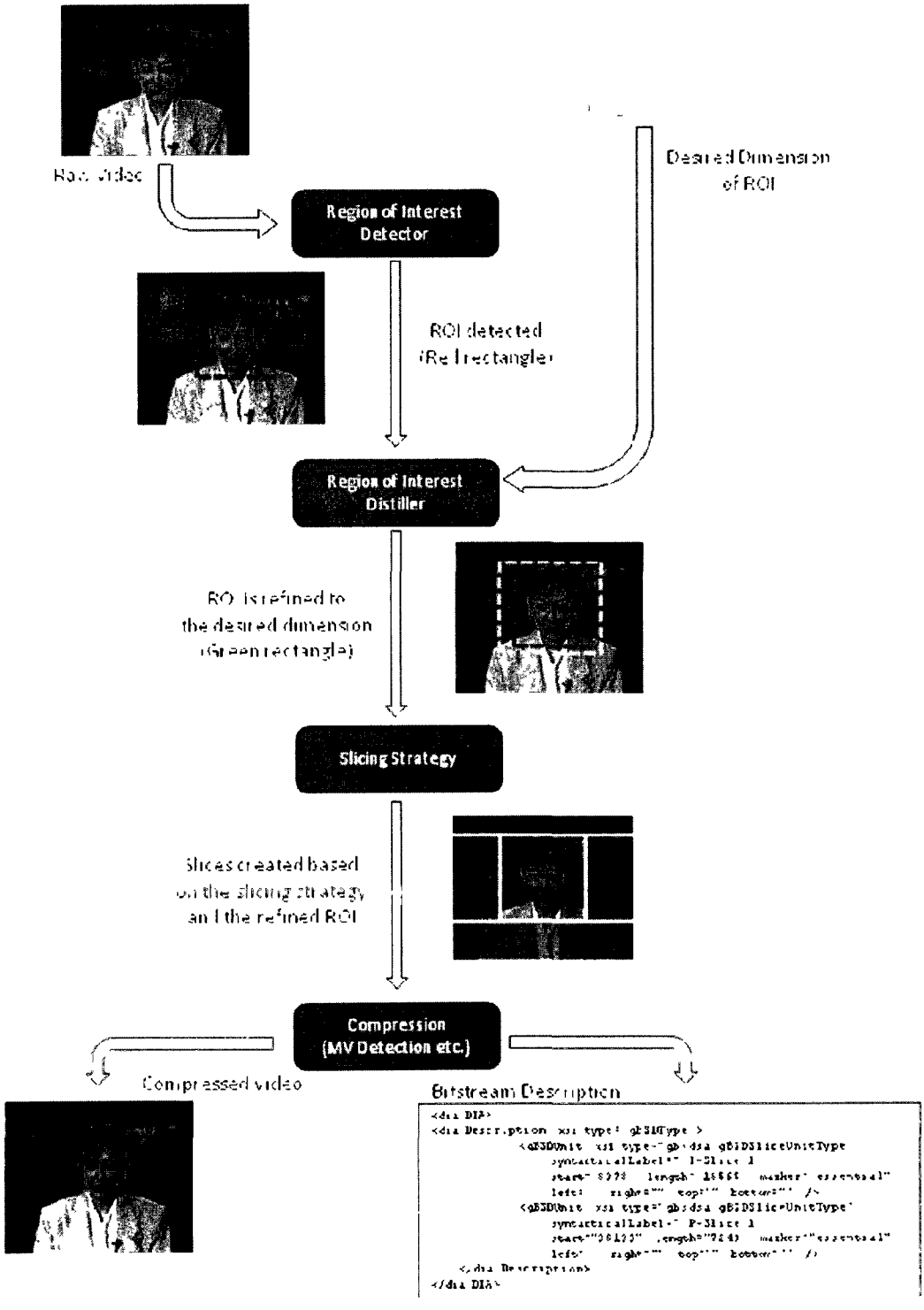


Figure 3.3: Encoding Process

Table 3.1: ROI Distiller Module

Assumptions:

- The y-coordinate of the video is 0 at the top, and increases towards the bottom by each pixel
- The x-coordinate of the video is 0 at the left, and increases towards the right by each pixel

Inputs:

- **SR**, *A set of rectangular Regions of Interest detected by the ROI Detector module*
- **D**, *Rectangular desired dimension of ROI*

Output:

- **RR**, *A refined rectangular region of interest*

ROI Distiller (SR, D):

```
    RR.Left = RR.Right = RR.Top = RR.Bottom = -1

    for each region R in SR:

        -- Find the left edge of all the regions
        if RR.Left = -1 or RR.Left > R.Left:
            RR.Left = R.Left

        -- Find the right edge of all the regions
        if RR.Right = -1 or RR.Right < R.Right:
            RR.Right = R.Right

        -- Find the top edge of all the regions
        if RR.Top = -1 or RR.Top > R.Top:
            RR.Top = R.Top

        -- Find the bottom edge of all the regions
        if RR.Bottom = -1 or RR.Bottom < R.Bottom:
            RR.Bottom = R.Bottom

    Width = RR.Right - RR.Left
    Height = RR.Bottom - RR.Top

    -- Now, position the Refined ROI so that it contains the center of the
    -- ROI (when computed ROI is larger than D), or the ROI is at the center
    -- of it (when D is larger than the computed ROI).

    if D.Width ≠ Width:
        Diff = D.Width - Width
        RR.Left = Min(Max(0, RR.Left - Diff / 2), Picture.Width - D.Width)
        RR.Right = RR.Left + D.Width

    if D.Height ≠ Height:
        Diff = D.Height - Height
        RR.Top = Min(Max(0, RR.Top - Diff / 2), Picture.Height - D.Height)
```

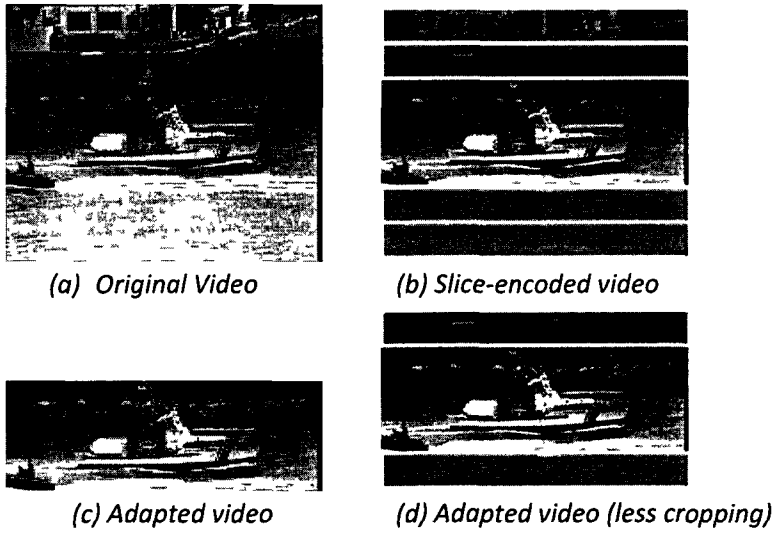
```
        RR.Bottom = RR.Top + D.Height

    RR.Width = D.Width
    RR.Height = D.Height

    return RR
```

3.2.4 Slicing Strategies

Several slicing strategies are proposed and explained in this section. The criteria for selecting a slicing strategy are also explained.

a. Wide Strategy

In this slicing strategy, the picture is divided into a number of horizontal slices of possibly varying height. The width of each slice is the same as the width of the picture itself. This slicing strategy can be useful for videos where there is a spatially wide region of interest. In this strategy, the entire ROI can be encoded into one essential slice. The heights of the disposable slices depend on the height of the video, and the desired granularity. For example, if a finer granularity is desired for spatial adaptation, then a lot of disposable slices of small heights are used to encode the RONI. This way, it is possible to have adapted videos of a larger variation of heights (Figure 3.4 (c) and (d)). If the video height itself is too small, for example, QCIF or SQCIF videos, or if finer granularity is not desired, then there will be only a few disposable slices with greater heights.



(a) Original Video *(b) Slice-encoded video*

(c) Adapted video *(d) Adapted video (less cropping)*

Figure 3.4: Wide Strategy with Varying Degree of Spatial Adaptation

The wide strategy can be adopted when the refined region of interest computed by the ROI Distiller has almost the same width as the picture. More precisely, the wide slicing strategy is used when:

$$Picture.Width - RROI.Width \leq 64$$

The desired level of granularity for spatial adaptation, *G*, is specified by a non-zero integer value. The higher the value of *G*, the greater granular cropping will be possible.

For example, for a CIF video (352x288), if the height of the ROI is 144 pixels, then the single essential slice will encode the ROI of 352x144, and the disposable slices will encode regions that add up to an area of 352x144. The number of disposable slices and the heights of the slices depend on the specified granularity *G*. A value of 1 for G, the lowest possible value, will use as few disposable slices as possible. So, if the ROI is at the top or at the bottom of the frame, then there will only be a single disposable slice of dimension 352x144. However, if there are regions in the frame both above and below the ROI, then one disposable slice will be used to encode the regions above the ROI, and another disposable slice is used to encode the regions below the ROI. For a higher value of *G*, more disposable slices will be used. A more precise computation of the number of slices and their heights is presented in Table 3.2.

Table 3.2: Wide Slicing Strategy

Inputs: Picture, RROI, G

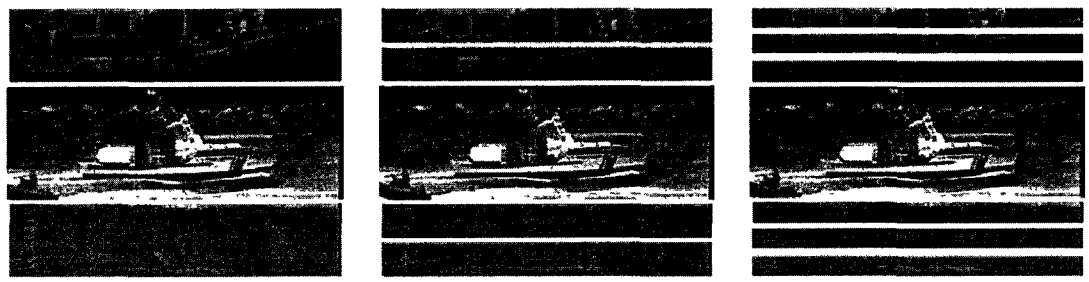
Outputs:

- **S**, *A set of regions, where each region should be encoded into one disposable slice*
- **E**, *A set of regions, where each region should be encoded into one essential slice*

Wide Strategy:

```
Above.Top = 0
Above.Bottom = RROI.Top
Above.Height = Above.Bottom

Below.Top = RROI.Bottom
Below.Bottom = Picture.Height
Below.Height = Below.Bottom - Below.Top

S = {}

Regions = {Above, Bottom}
for each region R in Regions:
```

```
        -- Compute the number of slices required for the selected granularity
level
        SliceHeight = Max(16, R.Height / G)
        CountSlices = R.Height / SliceHeight

        -- Determine the region belonging to each slice
        for count = 0 to CountSlices - 1:
            Slice.Top = R.Top + count * SliceHeight
            Slice.Bottom = Slice.Top + SliceHeight
            Slice.Left = 0
            Slice.Right = Picture.Width

            S = S U {Slice}

    -- There is exactly one essential slice for the entire frame
    Essential.Top = RROI.Top
    Essential.Bottom = RROI.Bottom
    Essential.Left = 0
    Essential.Right = Picture.Width
    E = {Essential}

    return S, E
```



(a) G = 1 *(b) G = 2* *(c) G = 3*

Figure 3.5: Wide Strategy at Various Granularities

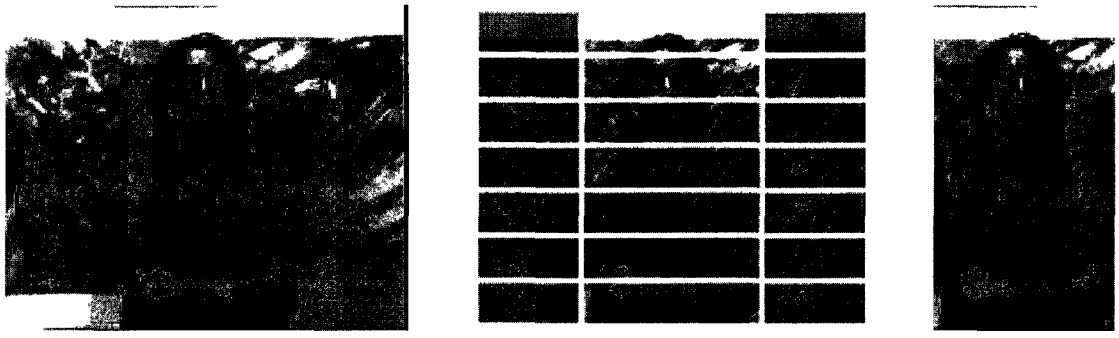
b. Block Strategy

This slicing strategy is useful for videos that have a tall but narrow region of interest. For this strategy, the picture is divided into a lot of slices each of single row height, and a few columns wide. This is necessary because in a slice, the macroblocks are expected to be in a sequential order. However, the nature of the tall and narrow ROI in the video makes it necessary to create small slices for both the essential and disposable slices. This is explained in Figure 3.6.

In this strategy, each row in the video has exactly one essential slice. The rest of the row is encoded into one or more disposable slices. The number of disposable slices in each row depends on the size of the video and *G*, the cropping granularity desired for spatial adaptation.

The block strategy can be adopted when the refined region of interest computed by the ROI Distiller has almost the same height as the picture, but considerably narrower than the picture. More precisely, the block slicing strategy is used when:

- `Picture.Height - RROI.Height ≤ 32, and`
- `Picture.Width - RROI.Width > 64`



(a) Original video *(b) Slice-encoded video* *(c) Adapted Video*

Figure 3.6: Non-Consecutive Macroblocks Necessitates Small Slices

The selection of slices in block strategy is described in detail in Table 3.3.

Table 3.3: Block Slicing Strategy

Inputs: Picture, RROI, G

Outputs:

- **S**, *A set of regions, where each region should be encoded into one disposable slice*
- **E**, *A set of regions, where each region should be encoded into one essential slice*

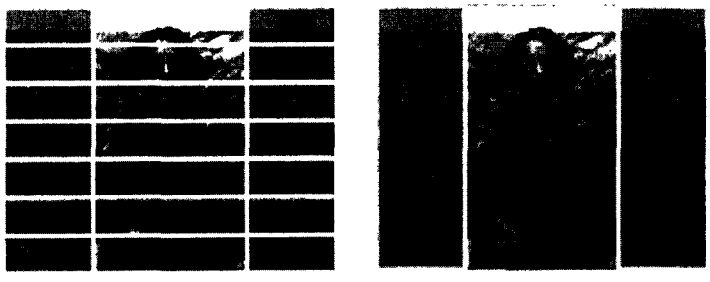
Block Strategy:

```
L.Left = 0
L.Right = RROI.Left
L.Width = L.Right

R.Left = RROI.Right
R.Right = Picture.Width
R.Width = R.Right - R.Left

S = {}

Regions = {L, R}
for each region R in Regions:
    -- Compute the number of slices required for each row
```

```
        SliceWidth = Max(48, R.Width / G)
        CountSlices = R.Width / SliceWidth

        for count = 0 to CountSlices - 1:
            for top = 0 to Picture.Height - 16 with increment of 16:
                -- Determine the region for each slice in each row
                Slice.Top = top
                Slice.Bottom = top + 16
                Slice.Left = R.Left + count * SliceWidth
                Slice.Right = Slice.Left + SliceWidth

                S = S U {Slice}

    -- There is one essential slice in each row of the frame
    E = {}
    for top = 0 to Picture.Height - 16 with increment of 16:
        Slice.Top = top
        Slice.Bottom = top + 16
        Slice.Left = RROI.Left
        Slice.Right = RROI.Right

        E = E U {Slice}

    return S, E
```

- **Flexible Macroblock Ordering**

As discussed, the block strategy requires a lot of slices for encoding each frame. However, having a lot of slices adds to the bitstream overhead of the slice headers, and the small slices reduce the number of macroblocks available to be used as reference. For this reason, a solution that requires only a small number of slices, but provides the same cropping options is more desirable. Flexible Macroblock Ordering (FMO) provides such a solution. Using FMO enables putting non-consecutive macroblocks into one slice. As a result, the region of interest can be encoded into a single slice, and the regions of non-interest (RONI) can be encoded into a smaller number of slices. Figure 3.7 illustrates this more clearly.



(a) Without using FMO *(b) Using FMO*

Figure 3.7: Encoding Videos using Flexible Macroblock Order using Block Strategy

Among the available FMO options, using FMO type 2 is most appropriate for this strategy. FMO type 2 allows defining rectangular regions in the picture, and a single slice can encode such a region. Therefore, in this strategy, the entire ROI is defined as one rectangular region, and only a single slice is used to encode it. The rest of the regions are divided into one or more regions, each encoded in a single disposable slice. The number and width of the disposable slices depend on the size of the video itself, and G, the desired granularity of spatial adaptation. The slice selection process is detailed in Table 3.4.

Table 3.4: Block Slicing Strategy using FMO

```
Inputs: Picture, RROI, G

Output: S, A set of regions, where each region should be encoded into one
disposable slice

Block Strategy using FMO:
    L.Left = 0
    L.Right = RROI.Left
    L.Width = L.Right

    R.Left = RROI.Right
    R.Right = Picture.Width
    R.Width = R.Right - R.Left

    S = {}

    Regions = {L, R}
    for each region R in Regions:
        -- Compute the number of slices required
        SliceWidth = Max(48, R.Width / G)
        CountSlices = R.Width / SliceWidth

        -- Determine the region belonging to each slice
        for count = 0 to CountSlices - 1:
            Slice.Top = 0
            Slice.Bottom = Picture.Height
            Slice.Left = R.Left + count * SliceWidth
            Slice.Right = Slice.Left + SliceWidth
            S = S U {Slice}

    -- There is exactly one essential slice for the entire frame
    Essential.Top = Picture.Top
    Essential.Bottom = Picture.Bottom
    Essential.Left = RROI.Left
    Essential.Right = RROI.Right

    E = {Essential}

    return S, E
```

c. O Strategy

This slicing strategy is useful when the region of interest in the video is a relatively small area such that the main video can be cropped both horizontally and vertically. In this strategy, the area above and below the ROI can be encoded into one or more disposable slices of possibly varying heights. These disposable slices will have the same width as the width of the picture. The height and the number of the disposable slices above and below the ROI depend on the height of the video, and the desired granularity for spatial adaptation. For the region containing the ROI, each row is divided into exactly one essential slice, and several disposable slices. The width and number of disposable slices in this region are also dictated by the width of the video and the cropping granularity.

(a) Original video *(b) Slice-encoded video* *(c) Adapted video*

Figure 3.8: O-Strategy

The o-strategy can be thought of as a combination of the block strategy and the wide strategy. It uses wide disposable slices above and below the ROI, like in the wide strategy, and uses narrow and small disposable slices on both sides of the ROI, like in the block strategy.

The o-strategy can be used when the ROI is relatively small in the video frame. More specifically, o-strategy should be used when:

- `Picture.Height - RROI.Height > 32, and`
- `Picture.Width - RROI.Width > 64`

The selection of slices in o-strategy is described in detail in Table 3.5.

Table 3.5: O-Slicing Strategy

Inputs: Picture, RROI, G

Outputs:

- **S**, *A set of regions, where each region should be encoded into one disposable slice*
- **E**, *A set of regions, where each region should be encoded into one essential slice*

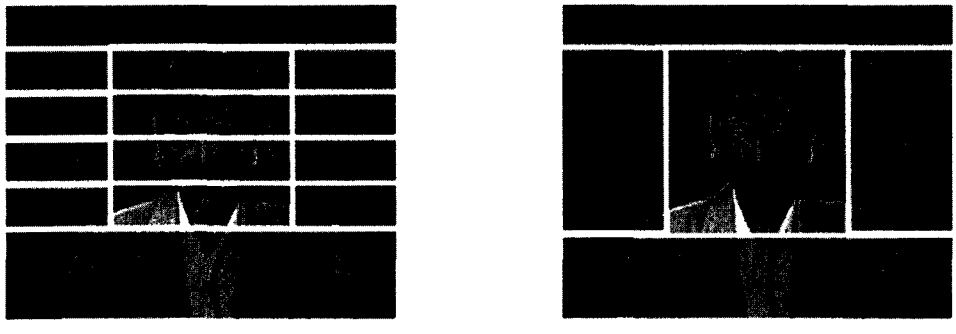
O Strategy:

```
    -- Compute the regions on either side of the RROI
    L.Left = 0
    L.Right = RROI.Left
    L.Top = RROI.Top
    L.Bottom = RROI.Bottom

    R.Left = RROI.Right
    R.Right = Picture.Width
    R.Top = RROI.Top
    R.Bottom = RROI.Bottom

    -- Compute the regions completely above and completely below the RROI
    Above.Left = 0
    Above.Right = Picture.Width
    Above.Top = 0
    Above.Bottom = RROI.Top

    Below.Left = 0
    Below.Right = Picture.Width
    Below.Top = RROI.Bottom
    Below.Bottom = Picture.Height

    S = {}

    -- For both of the regions completely above and completely below the RROI,
    -- use wide slicing strategy
    Regions = {Above, Below}
    for each region R in Regions:
        -- Count the number of slices required for the requested granularity
        R.Height = R.Bottom - R.Top
        SliceHeight = Max(16, R.Height / G)
        CountSlices = R.Height / SliceHeight

        -- Compute the region belonging to the slice
        for count = 0 to CountSlices - 1:
            Slice.Top = R.Top + count * SliceHeight
            Slice.Bottom = Slice.Top + SliceHeight
            Slice.Left = R.Left
            Slice.Right = R.Right

            S = S U {Slice}

    -- Use block slicing strategy for the region containing the RROI
    Regions = {L, R}
    for each region R in Regions:
```

```
        -- Count the number of slices required for each row
        R.Width = R.Right - R.Left
        SliceWidth = Max(48, R.Width / G)
        CountSlices = R.Width / SliceWidth

        -- Compute the region belonging to each slice in each row
        for count = 0 to CountSlices - 1:
            for top = R.Top to R.Bottom - 16 with increment of 16:
                Slice.Top = top
                Slice.Bottom = top + 16
                Slice.Left = R.Left + count * SliceWidth
                Slice.Right = Slice.Left + SliceWidth

                S = S U {Slice}

    -- Compute the slices belonging to the RROI
    E = {}
    for top = RROI.Top to RROI.Bottom - 16 with increment of 16:
        Slice.Top = top
        Slice.Bottom = top + 16
        Slice.Left = RROI.Left
        Slice.Right = RROI.Right

        E = E U {Slice}

    return S, E
```

- **Flexible Macroblock Ordering**

Using flexible macroblock ordering can be useful for O-strategy too. As in block strategy, FMO type 2 can be useful in this strategy too. The region of interest is defined in a single rectangular region. The rest of the region is divided into several rectangular regions. Each of these regions of RONI is encoded into a disposable slice.



(a) Without using FMO *(b) Using FMO*

Figure 3.9: Encoding Videos using Flexible Macroblock Order using O-Strategy

The selection process of slices for O-strategy needs to be changed when using FMO. The initial selection of slices above and below the ROI is the same as described in Table 3.5 (`Above` and `Below` regions). However, for the regions on the left and right of the ROI, and for the essential slice, some modifications need to be made to the process. The modifications are listed in Table 3.6.

Table 3.6: O Slicing Strategy using FMO

Inputs: Picture, RROI, G

Outputs:

- **S**, *A set of regions, where each region should be encoded into one disposable slice*
- **E**, *A set of regions, where each region should be encoded into one essential slice*

O Strategy using FMO:

```
    Above and Below are processed the same way as in Table 3.5

    Regions = {L, R}
    for each region R in Regions:
        -- Count the number of slices required for the requested granularity
        R.Width = R.Right - R.Left
        SliceWidth = Max(48, R.Width / G)
        CountSlices = R.Width / SliceWidth

        -- Compute the region belonging to each slice
        for count = 0 to CountSlices - 1:
            Slice.Top = RROI.Top
            Slice.Bottom = RROI.Bottom
            Slice.Left = R.Left + count * SliceWidth
            Slice.Right = Slice.Left + SliceWidth

            S = S U {Slice}

    -- There is exactly one essential slice in each frame
    Essential.Top = RROI.Top
    Essential.Bottom = RROI.Bottom
    Essential.Left = RROI.Left
    Essential.Right = RROI.Right

    E = {Essential}

    return S, E
```

In all the discussed strategies, the scheme for selecting the number and size of disposable slices depend on the size of the video picture, and especially on the desired level of granularity for

spatial adaptation. However, having a large number of slices for each frame in the video is undesirable, because:

- Smaller slices reduce available regions to be used as reference macroblocks.
- Each slice has its own header. So a larger number of slices can incur a considerable cost on the bandwidth.

Using Flexible Macroblock Ordering in Block and O-strategies enables encoding each frame in the video in a smaller number of slices, thus making it a more suitable solution than the non-FMO version. However, there can be at most eight slice groups within a picture in H.264/AVC. This can restrict the available cropping options, especially in high-definition videos where a wide range of granularity is possible and may be desired. Also, many of the widely used decoders at present do not implement some of the advanced H.264 features, and FMO is one of them. For example, the decoder in popular media players such as MPlayer, VLC media player, Windows media player etc. cannot correctly decode an FMO encoded H.264 video. Therefore, media providers should take these criteria under consideration when deciding whether to take advantage of the FMO-encoding, or use the non-FMO slicing techniques.

3.2.5 *Generation of generic* Bitstream Syntax Description

As discussed in Chapter 2, the MPEG-21 standard provides a specification for digital item adaptation (DIA). In this specification, each digital item has a format-independent description. For example, Usage Environment Description describes the environment in the context of the user, which includes user preferences, network and device characteristics and natural environment characteristics. In the case of bitstream, Bitstream Syntax Description describes the bitstream in a format-agnostic manner that a network node can use to adapt the content without having any knowledge about the representation format of the bitstream. Usually the logical structure of the bitstream is explained in the Bitstream Syntax Description. XML is used to provide this description. Such a description facilitates a Digital Item resource adaptation engine to transform the bitstream and the corresponding description by removing or modifying data. This transformation can be controlled by an XSLT Style Sheet.

In the adaptation strategy presented in this thesis, the generic bitstream syntax description is generated by the encoder, since the encoder is readily aware of the details of the bitstream. In

the gBSD generated by the encoder, `addressMode` is set to *Absolute* and `addressUnit` is set to *byte* for the entire video.

a. Describing a Slice

In order to be able to spatially adapt a video by dropping slices that don't belong to the region of interest, it is necessary to know during adaptation what region in the picture a slice belongs to. Since the adaptation engine is format-agnostic, it is not possible to compute a slice's region during adaption from the bitstream. Therefore, it is necessary to describe the region of the slice in the bitstream description. To allow for this, we introduce an extension for the gBS Schema. We extend the `gBSDUnitType` and create `gBSDSliceUnitType` to describe each slice. In addition to `start`, `length` and `marker` attributes that are defined for `gBSDUnitType`, a `gBSDSliceUnitType` has `left`, `right`, `top` and `bottom` attributes of type `nonNegativeInteger`, describing the rectangular region a slice belongs to in the frame. This schema is presented in Table 3.7. It is also necessary to know during adaptation whether a slice is disposable or not. The `marker` for a slice is set to *disposable* for disposable slices, and to *essential* for essential slices. The `syntacticalLabel` attribute of a slice is set to *:Y-Slice:X*, where *X* is the sequence number of the slice in the frame, and *Y* is the type of slice (i.e. I, P or B).

b. Describing a Frame

During spatial adaptation, a decision is made regarding whether a slice should be delivered to the end user or not. This decision depends not only on the spatial region of the slice in the frame, but also the relative spatial distance of this region from the region of interest in the frame. To facilitate this decision, the dimension and location of the region of interest is necessary before any slice is processed. For this reason, we need to describe the ROI in the frame. To this end, we introduce an extension to gBS schema. We extend the `gBSDUnitType` and create `gBSDFrameUnitType` to describe each frame. In addition to start and length attributes defined for `gBSDUnitType`, a `gBSDFrameUnitType` has `left`, `right`, `top` and `bottom` attributes of `nonNegativeInteger`, describing the rectangular region of interest in the frame. This schema is presented in Table 3.7. The optional `marker` attribute is not specified for frames. The `syntacticalLabel` for a

frame is set to *:Y-Frame:X*, where *X* represents the sequence number of a frame in the video sequence, and *Y* represents the frame type (i.e. I, P or B).

Table 3.7: gBS Schema for Spatial Adaptation

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7 -->
<!-- Schema for Spatial Adaptation Extension -->
<schema targetNamespace="urn:mpeg:mpeg21:2010:01-DIA-SpatialAdaptation-NS"
      xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
      xmlns:gbsdSA="urn:mpeg:mpeg21:2010:01-DIA-SpatialAdaptation-NS"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <annotation>
            <documentation>
                  Schema for Spatial Adaptation Extension
            </documentation>
      </annotation>

      <import namespace="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS" />

      <complexType name="gBSDSliceUnitType">
            <complexContent>
                  <extension base="gbsd:gBSDUnitType">
                        <attribute name="left" type="nonNegativeInteger" />
                        <attribute name="right" type="nonNegativeInteger" />
                        <attribute name="top" type="nonNegativeInteger" />
                        <attribute name="bottom" type="nonNegativeInteger" />
                  </extension>
            </complexContent>
      </complexType>

      <complexType name="gBSDFrameUnitType">
            <complexContent>
                  <extension base="gbsd:gBSDUnitType">
                        <attribute name="left" type="nonNegativeInteger" />
                        <attribute name="right" type="nonNegativeInteger" />
                        <attribute name="top" type="nonNegativeInteger" />
                        <attribute name="bottom" type="nonNegativeInteger" />
                  </extension>
            </complexContent>
      </complexType>
</schema>
```

An example bitstream description generated by the encoder for a spatially adaptable video is presented in Table 3.8.

Table 3.8: gBSD for Spatially Adaptable Video

```
<dia:DIA
      xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
      xmlns:gbsdsa="urn:mpeg:mpeg21:2010:01-DIA-SpatialAdaptation-NS"
```

```
        xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <dia:Description xsi:type="gBSDType">
        <gBSDUnit xsi:type="gbsdsa:gBSDFrameUnitType"
                syntacticalLabel=":I-Frame:0" start="576" length="34660"
                left="0" right="352" top="81" bottom="224">
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":I-Slice:0"
                start="576" length="7802" marker="disposable"
                left="0" right="352" top="0" bottom="80" />
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":I-Slice:1"
                start="8378" length="18565" marker="essential"
                left="0" right="352" top="81" bottom="224" />
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":I-Slice:2"
                start="26943" length="8293" marker="disposable"
                left="0" right="352" top="225" bottom="288" />
        </gBSDUnit>
        <gBSDUnit xsi:type="gbsdsa:gBSDFrameUnitType"
                syntacticalLabel=":P-Frame:1"                start="35236"
length="13000"
                left="0" right="352" top="81" bottom="224">
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":P-Slice:0"
                start="35236" length="2897" marker="disposable"
                left="0" right="352" top="0" bottom="80" />
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":P-Slice:1"
                start="38133" length="7243" marker="essential"
                left="0" right="352" top="81" bottom="224" />
            <gBSDUnit xsi:type="gbsdsa:gBSDSliceUnitType"
                syntacticalLabel=":P-Slice:2"
                start="45376" length="2860" marker="disposable"
                left="0" right="352" top="225" bottom="288" />
        </gBSDUnit>
    </dia:Description>
</dia:DIA>
```

3.3 Adaptation Engine

The adaptation engine performs the adaptation in two steps: first it adapts the bitstream description according to the user preferences and characteristics using an XSLT stylesheet. This adapted bitstream description is stored so that it can be reused when the same media is requested from the same (or sufficiently similar) user environment. This adapted gBSD is then used to generate the adapted bitstream, which is then delivered to the end user.

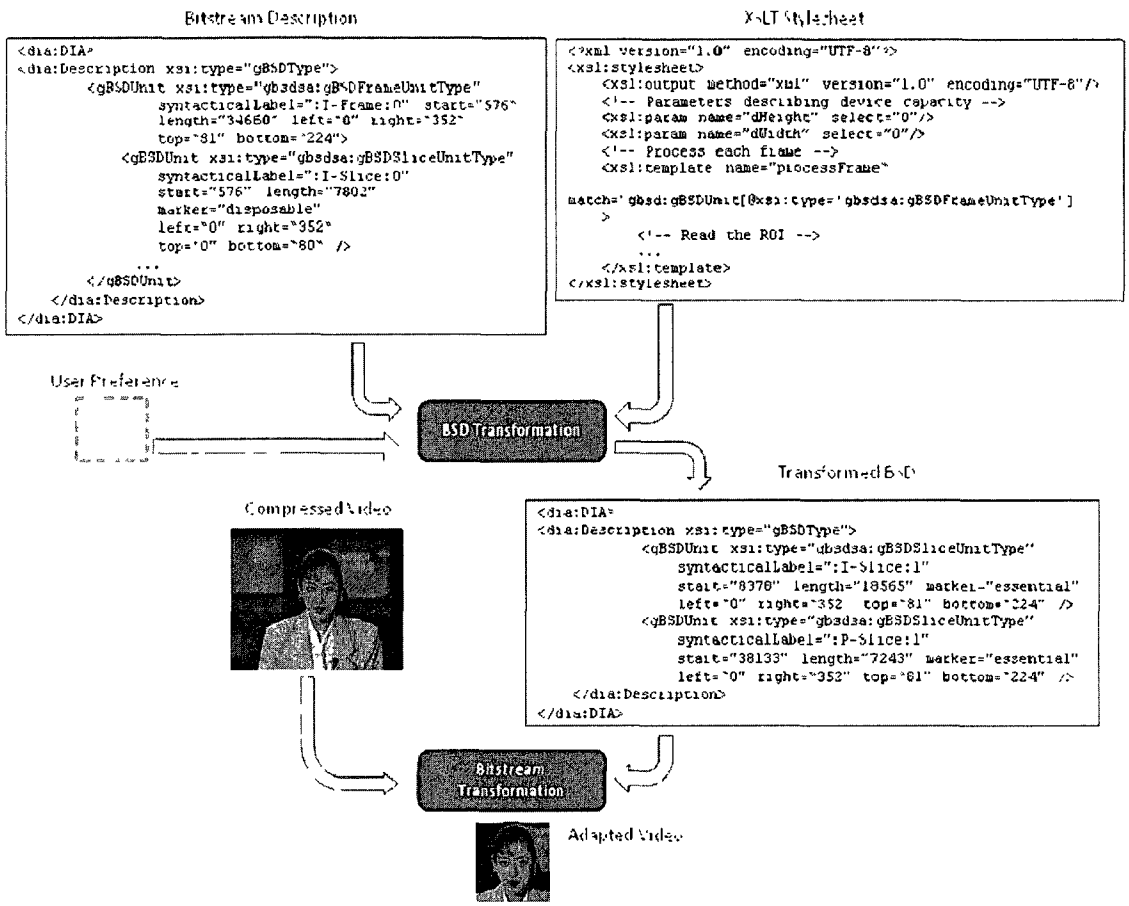


Figure 3.10: Adaptation Process

The XSLT stylesheet processes the gBSD frame by frame. For each frame, it reads the location and dimension of the ROI in the frame. From this dimension, and the requested dimension of the video from the user, it computes the effective region for the frame. The effective region is at least as wide and as high as the ROI in the frame, even if the requested dimension is smaller. However, if the requested dimension is larger than the ROI in the frame, then the effective region is expanded equally on each direction. The stylesheet then processes each slice in the frame and reads the dimension of the region contained in the slice. If any part of the slice belongs to the computed effective region, then the slice information is included in the adapted gBSD. Otherwise, the slice is omitted.

The XSLT stylesheet used to generate the adapted gBSD is presented in Table 3.9.

Table 3.9: Stylesheet for Spatial Adaptation

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet
    xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
    xmlns:gbsdsa="urn:mpeg:mpeg21:2010:01-DIA-SpatialAdaptation-NS"
    xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" version="1.0" encoding="UTF-8"/>

    <!-- Parameters describing device capacity -->
    <xsl:param name="dHeight" select="0"/>
    <xsl:param name="dWidth" select="0"/>

    <!-- Process each frame -->
    <xsl:template name="processFrame"
                match="gbsd:gBSDUnit[@xsi:type='gbsdsa:gBSDFrameUnitType']">
        <!-- Read the ROI -->
        <xsl:variable name="rLeft" select="@left" />
        <xsl:variable name="rRight" select="@right" />
        <xsl:variable name="rTop" select="@top" />
        <xsl:variable name="rBottom" select="@bottom" />
        <xsl:variable name="rHeight" select="$rBottom - $rTop" />
        <xsl:variable name="rWidth" select="$rRight - $rLeft" />

        <!-- Expand the left and right boundaries when possible -->
        <xsl:variable name="eLeft">
            <xsl:choose>
                <xsl:when test="$rWidth &lt; $dWidth">
                    <xsl:variable name="tmpLeft"
                                select="$rLeft - ($dWidth - $rWidth) div 2" />
                    <xsl:choose>
                        <xsl:when test="$tmpLeft &lt; 0">
                            <xsl:text>0</xsl:text>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:value-of select="$tmpLeft" />
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$rLeft" />
                </xsl:otherwise>
            </xsl:choose>
        </xsl:variable>
        <xsl:variable name="eRight">
            <xsl:choose>
                <xsl:when test="$rWidth &lt; $dWidth" >
                    <xsl:value-of select="$eLeft + $dWidth" />
```

```
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$rRight" />
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

    <!-- Expand the top and bottom boundaries when possible -->
    <xsl:variable name="eTop">
        <xsl:choose>
            <xsl:when test="$rHeight &lt; $dHeight">
                <xsl:variable name="tmpTop"
                          select="$rTop - ($dHeight - $rHeight) div 2" />
                <xsl:choose>
                    <xsl:when test="$tmpTop &lt; 0">
                        <xsl:text>0</xsl:text>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:value-of select="$tmpTop" />
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$rTop" />
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:variable name="eBottom">
        <xsl:choose>
            <xsl:when test="$rHeight &lt; $dHeight" >
                <xsl:value-of select="$eTop + $dHeight" />
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$rBottom" />
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

    <!-- Now, go over each slice -->
    <xsl:for-each select="gbsd:gBSDUnit[@xsi:type='gbsdsa:gBSDSliceUnitType']">
        <!-- Read the region belonging to the slice -->
        <xsl:variable name="sLeft" select="@left" />
        <xsl:variable name="sRight" select="@right" />
        <xsl:variable name="sTop" select="@top" />
        <xsl:variable name="sBottom" select="@bottom" />

        <!-- If this slice has any region inside the (possibly expanded)
             ROI, output it -->
        <xsl:if test="(($sLeft &gt;= $eLeft and $sLeft &lt; $eRight) or
                       ($sRight &gt;= $eLeft and $sRight &lt; $eRight)) and
                  (($sTop &gt;= $eTop and $sTop &lt; $eBottom) or
                     ($sBottom &gt;= $eTop and $sBottom &lt; $eBottom))" >
            <xsl:copy-of select="." />
```

```
            </xsl:if>

        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

The adapted gBSD contains only `gBSDUnits` describing the slices that belong to the region of interest. The generation of an adapted bitstream from the adapted gBSD is thus very fast, as it only requires very little XML parsing.

3.4 Bandwidth Cost of Slicing Strategy

Using any of the slicing strategies discussed in this chapter often requires more slices to encode a video than would otherwise be used. This can affect the compression quality of the generated video stream. If the slices are small in dimension, then there is a fewer number of macroblocks available to be used as references. As a result, there may be less compression. For this reason, the encoded video can have a larger size than a video encoded without using any of the slicing strategies. However, using slicing strategies makes it possible to deliver cropped videos to a client using a device with constrained capabilities. This cropped video can be of a smaller size than a full video encoded without any slicing strategy.

Let, the size of the encoded video without slicing strategies is `ES`. The size of the video encoded using a slicing strategy is `SS`. The size of the cropped video is `CS`. If `X%` of all users are delivered the cropped video, and the rest of the users are delivered the full video, then the average bandwidth use per user is:

$$\frac{X * CS + (100 - X) * SS}{100}$$

However, when no slicing strategy is used, the entire video is delivered to all users, which means the average bandwidth use for each user is:

$$ES$$

One of the motivations behind spatial adaptation is to reduce the resource usage. To determine if using the slicing strategies is going to be profitable, the average bandwidth usage per user should be reduced, i.e. in order to reduce the overall bandwidth usage on the server, we want:

$$\frac{X * CS + (100 - X) * SS}{100} < ES$$
$$\Rightarrow X * (SS - CS) > 100 * (SS - ES)$$
$$\Rightarrow X > 100 * \frac{SS - ES}{SS - CS}$$

This measurement is used in the following chapter to validate the profitability of the proposed spatial adaptation scheme in terms of bandwidth consumption.

In this chapter, a new spatial adaptation technique has been proposed and described in detail. The adaptation technique requires dividing each frame into a number of essential slices containing the ROI in the frame, and a number of disposable slices containing the rest of the regions in the frame. Various slicing strategies have been designed and explained in great detail that can be used for this purpose. The slicing strategy used for encoding is determined by the dimension and location of the ROI in the frame. The encoder also generates a gBSD description of the encoded video. The adaptation engine uses an XSLT stylesheet for adaptation that processes client request and the bitstream syntax description of the encoded video to determine the slices to drop and the slices to deliver to the client. This adapted video is then delivered to the client.

Chapter 4 : Validation of the Proposed Scheme

For this thesis, the proposed spatial adaptation framework was implemented in order to validate the framework by measuring its performance and its applicability.

4.1 Prototype

The implemented system consists of two separate modules: (1) the encoder, and (2) the adaptation server.

4.1.1 Encoder

Instead of writing an entire encoder from scratch, the spatial adaptation framework was added to an existing H.264 encoder. There are several encoders currently available. H.264/AVC JM Reference Software [24] is widely used for academic research purposes. However, the X264 encoder [25] is also very popular, and widely used by commercial products and services, such as ffmpeg [26], VLC Media Player [27], Google Video [28] etc. For this reason, X264 was chosen as the preferred encoder to work with for adding support for spatial adaptation.

X264 supported many of the features of the H.264 standard. However, some of the features were not yet supported. For example, a single frame could not have more than one slice in it. So for the purpose of the proposed spatial adaptation, support for multiple slices was added to the X264 encoder. There was no support for FMO in the encoder as well. In order to evaluate the proposed methods, support for FMO was also added to the encoder. In addition to that, a framework for detecting and refining the region of interest was added to the encoder. Finally, the logic for selecting slicing strategies and then computing the slice boundaries were implemented into the encoder.

- **Support for Multiple Slices**

X264 used to allow only one slice in each frame. However, for the proposed spatial adaptation scheme, it is essential to support multiple slices in a single frame. For this reason, support for multiple slices was added to X264. The encoder uses `x264_slices_write` procedure to write the slices in the compressed video stream. The sequence number of the first macroblock in a slice is denoted by `i_first_mb`, and the last macroblock in the slice is denoted by

i_last_mb. Instead of including all macroblocks in the frame into the same slice, i.e., instead of setting i_first_mb to 0 and i_last_mb to i_mb_count - 1 (i.e. the number of macroblocks in the frame), we use a routine x264_slices_next that looks up the computed slice regions returned by the selected slicing strategy, and decides the values of i_first_mb and i_last_mb for each slice in the frame.

- ***Region of Interest Detector and ROI Distiller***

As explained in the previous chapter, detecting the region of interest is not within the scope of this thesis. Any of the available algorithms for detecting the ROI in an image can be used with the proposed spatial adaptation scheme. Therefore, for these experiments, this ROI framework always selects the rectangular region in the middle of the frame as the refined region of interest. This refined region of interest has the same dimension as the desired dimension of the ROI.

```
/* Selects the region in the middle of the frame as the region
 * of interest. The selected region has the same dimension as the
 * desired dimension of the ROI 'dim'. The region is returned
 * in the 'region' variable.
 */
void x264_roi_retrieve(x264_t *h, x264_picture_t *pic,
        x264_dimension_t *dim, x264_region_t *region)
{
    int width = dim->width;
    int height = dim->height;

    region->left = (h->param->i_width - width) / 2;
    if (region->left < 0)
        region->left = 0;
    region->right = region->left + width;

    region->top = (h->param->i_height - height) / 2;
    if (region->top < 0)
        region->top = 0;
    region->bottom = region->top + height;
}
```

- **Slicing Strategies**

All the various slicing strategies described in Chapter 3 were implemented.

- **Generation of gBSD**

In order to generate the bitstream description of the encoded video-stream, I created a new output module for X264. X264 has several output modules, e.g. an *flv* module for creating flash video for the encoded stream. X264 provides an easy-to-use interface for writing such output modules, with the following signature:

```
typedef struct
{
    int (*open_file)( char *psz_filename, hnd_t *p_handle );
    int (*set_param)( hnd_t handle, x264_param_t *p_param );
    int (*write_nalu)( hnd_t handle, uint8_t *p_nal, int i_size );
    int (*set_eop)( hnd_t handle, x264_picture_t *p_picture );
    int (*close_file)( hnd_t handle );
} cli_output_t;
```

The operations defined by the interface are:

- **Set Parameters** `(set_param)`

  In this stage, X264 feeds the parameters provided by the user (i.e. the user encoding the video) to the output module. In order to enable the gBSD module, the user should use the `--gbsd <filename>` command-line parameter. The <filename> should be replaced by the desired name of the gBSD file, e.g.

  ```
  $ ./x264 --gbsd gbsd.xml ...
  ```

  The gBSD module is enabled when this parameter is used, and the filename is stored for use in subsequent steps by the module.

- **Open File** `(open_file)`

  In this stage, the output module creates two files: the output file for the encoded video stream, and the gBSD file. The output file is created in binary mode, while the gBSD file is created in text-mode using standard `fopen` function call.

- **Write NAL Units** `(write_nalu)`

  In this stage, the output module writes NAL units to the video stream. At the same time, for the gBSD description, the module first determines the type of data in the NAL unit, i.e. whether it is SPS (Sequence Parameter Set), PPS (Picture Parameter Set), SEI (Supplemental Enhancement Information) or a slice. In case of a slice, the module determines if this is the first slice in a frame. For the first slice in the frame, the

gBSDUnit tag of type gBSDFrameUnitType for the frame is first written. The tag includes left, right, top and bottom attributes, describing the region of interest in the frame. The output module then writes one gBSDUnit tag of type gBSDSliceUnitType for each slice in the frame. Each such tag for a slice also includes left, right, top and bottom attributes, describing the region in the frame belonging to this slice. Each time a slice is written, the output module increases the slice counter for the frame, so that it can be used for the syntacticalLabel attribute for the slice. The output module also updates the number of bytes written in the video-stream for each NAL unit, so that it can keep track of the starting byte position of a slice in the stream, and uses this count for the start attribute of the gBSDUnit tags. Standard fprintf function calls are used in order to output the gBSD data in the file opened during the open_file stage earlier.

- **End a frame** (set_eop)

  In this stage, the output module closes the gBSDUnit tag for a frame, and resets the slice-counter, so that the module can determine during the next *Write NAL Unit* step for a slice that it is the first slice for a frame. This step also increases the frame counter so that it can be used for the syntacticalLabel attribute of the gBSDUnit tags.

- **Close File** (close_file)

  This is the last step of the encoding process. In this step, the module closes the videostream. It also closes the dia:Description and dia:DIA tags for the gBSD description.

- **Flexible Macroblock Ordering**

Only support for FMO type 2 was added, where each slice-group occupies a rectangular region in the frame. To add support for FMO, the Picture Parameter Set (PPS) in the video-stream need to contain information about the number of slice-groups in the frame, and the rectangular regions in the frame belonging to each slice-group. The slice-groups within a frame are always created as non-overlapping rectangles. x264_pps_t::i_num_slice_groups is set to the number of slice groups – 1, as specified by the H.264 specification. In X264, the macroblocks in a frame are added to the stream sequentially from left to right until the right edge of the frame, and top to bottom. However, with FMO, that may not be the case. So the code was modified to ensure that when a macroblock at the right-edge of a slice-group is added to the stream, the

next macroblock added to the stream is the next macroblock in the slice-group, not the macroblock immediately on the right of the last written macroblock.

4.1.2 Adaptation Server

A standalone server has been created for receiving client requests and performing necessary video adaptation before serving the video stream to the users. Since most currently available clients do not have support for communicating the Usage Environment Description (UED) to the media server as described in MPEG-21, this requirement is relaxed and omitted from the implementation of the adaptation server. Instead, clients request for a video-stream using a URI, and the dimension of the video device is included in the URI as a query-string. So instead of processing the UED from the client, the adaptation server processes the query-string in the URI to determine the region of the video to crop.

The adaptation server has been written in C. The main features of this server are listed below:

- The adaptation server uses `libhttpd` [29] for receiving requests from clients over the HTTP protocol. The library provides a simple way for creating a web-server as follows:

```
httpd *server;
server = httpdCreate(host,port);

/* Setup some content for the server */
httpdAddCContent(server, "/", "video", HTTP_TRUE, NULL, serv_video);
```

  The callback function `serv_video` is executed whenever a request is made for the 'video' URL. The function processes three parameters in the query-string and sends back the adapted video stream:

  - `video`: the name of the video (e.g. *'coastguard'*, *'flower'* etc.)
  - `h`: the height of the video device (or the desired height of the video)
  - `w`: the width of the video device (or the desired width of the video)

  So, for example, a client with a `176x144` display requesting the *flower* video will use the following URI request:

  `http://server/video?video=flower&w=176&h=144`

If any of the required parameters is missing in the request-string, then a `404` HTTP response is sent to the client.

```
static void serv_video(httpd *server)
{
    httpVar *vvid, *vh, *vw;
    int height, width;

    vvid = httpdGetVariableByName(server, "video");
    vh = httpdGetVariableByName(server, "h");
    vw = httpdGetVariableByName(server, "w");

    if (!vvid || (!vh || sscanf(vh->value, "%d", &height) != 1) ||
            (!vw || sscanf(vw->value, "%d", &width) != 1))
    {
        httpdSetResponse(server,
                "404 Please provide all necessary parameters.");
        httpdOutput(server, "\n");
        return;
    }

    /* Adapt video and output to the client */
    adapt_video(server, vvid->value, vh->value, vw->value);
}
```

- Given the name of the requested video and the dimension of the video device, the adaptation server looks up the gBSD description of the requested video file, and the XSLT stylesheet to process the gBSD description. The adaptation server assumes that the gBSD description has the same name as the video file, but with `.gBSD` extension. The stylesheet file is saved stored as *SpatialAdaptation.xsl*. So, for the *flower* video, the adaptation server looks for a video file named *flower.h264*, a gBSD file named *flower.gBSD*. The server then uses `libxml2` [30] and `libxslt` [31] to processes the gBSD XML with the stylesheet to generate the adapted gBSD description.

```
static void
adapt_video(httpd *server, const char *video,
                            const char *height, const char *width)
{
```

```
    char filexsl[FILENAME_MAX];
    char filegBSD[FILENAME_MAX];
    const char *params[] = {"dHeight", height, "dWidth", width, NULL};

    /* Process the stylesheet */
    xsltStylesheetPtr cur;
    snprintf(filexsl, sizeof(filexsl), "SpatialAdaptation.xsl");
    cur = xsltParseStylesheetFile(filexsl);

    /* Read the gBSD file */
    xmlDocPtr doc;
    snprintf(filegBSD, sizeof(filegBSD), "%s.gBSD", video);
    doc = xmlParseFile(filexsl);

    /* Create the adapted gBSD description */
    xmlDocPtr res;
    res = xsltApplyStylesheet(cur, doc, params);

    ......

    /* Clean up */
    xsltFreeStylesheet(cur);
    xmlFreeDoc(res);
    xmlFreeDoc(doc);
}
```

This adapted gBSD is then processed to generate the adapted video stream. In order to generate the adapted video stream, a generic program was written that reads the addressing information, i.e. offset and length of each slice to be delivered from the adapted gBSD description, and outputs only these portions of the bitstream from the original video stream to the client. It should be noted that standard processors such as gBSDtoBin can also be used to generate the adapted video stream from the adapted gBSD and the original video stream. This adapted video stream is then served to the client.

The server can store the adapted gBSD in the filesystem, and avoid processing the stylesheet to generate the adapted gBSD for each subsequent request. However, for simplicity, the server implemented for this thesis processes the stylesheet and adapts

the gBSD description for each request. This scenario, where the adaptation server has to adapt the gBSD description for each client request, is the worst case scenario, and allows measuring the worst cast performance of the adaptation system.

A client does not interact with the encoder; rather, the client interacts with the HTTP module of the adaptation system. Figure 4.1 shows the sequence diagram of the client requests and server response of the implemented system.

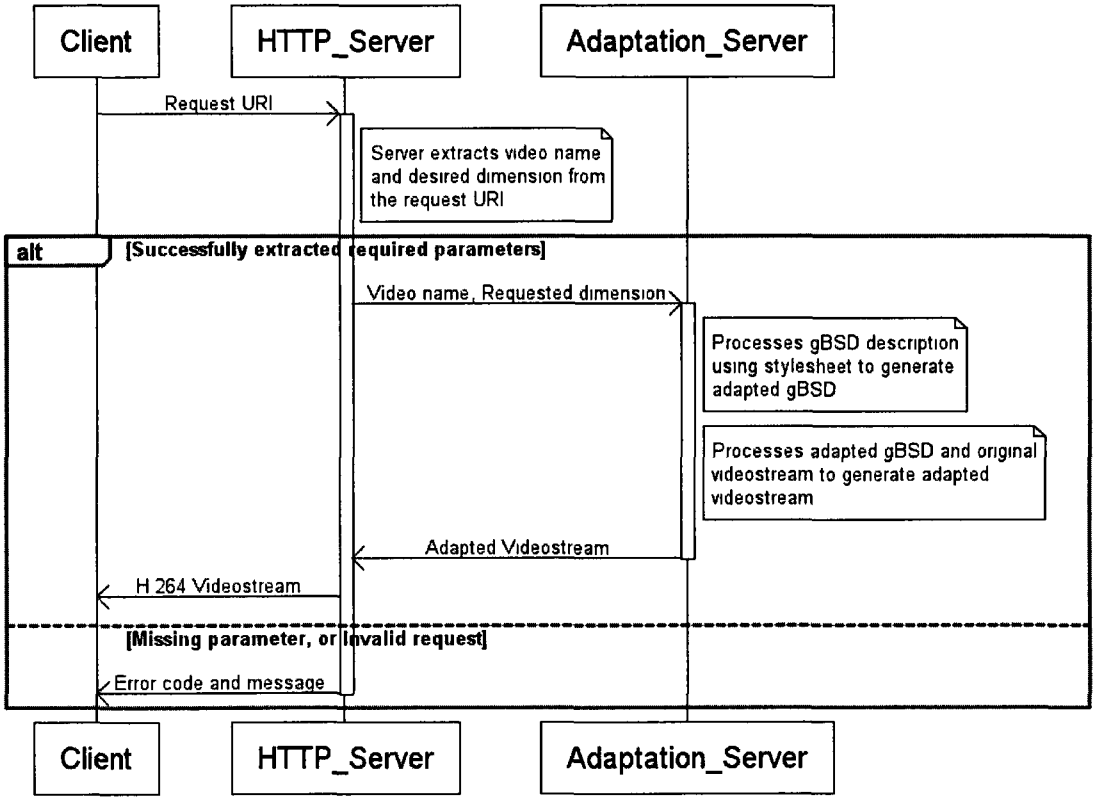


Figure 4.1: Sequence Diagram for the Adaptation System

4.2 Evaluation and Results

To evaluate the spatial adaptation system, several standard videos are encoded using the adapted X264 encoder. The adaptation system is then used to generate adapted videos of various dimensions. The results from these tests help in evaluating the performance of the system

4.2.1 Test Environment Setup

Several video files of varying resolutions were used for testing the adaptation system. They are listed below:

- CIF (352x288): Akiyo, Flower, Hall, News, Silent
- 4CIF (704x576): Ice, Crew[1], Soccer
- HD Sequence (1280x720): Parkrun

Each of these videos has 250 frames, at 30 fps. For encoding, IDR interval of 100 frames and quantization parameter (QP) of 25 were used.

For encoding and adaptation, the following resources were available for the system:

- CPU : Intel(R) Pentium(R) Dual CPU T3200 @ 2.00GHz
- Memory : 2 GB
- Operating System : GNU/Linux (2.6.27)

There can be a large number of possible dimensions for the refined ROI in a video. For this testing, however, a fixed dimension for RROI is used; the dimension of RROI is:

- QCIF (176x144) for CIF videos
- CIF for 4CIF videos
- 4CIF for HD videos

Each slicing strategy is deliberately used for each of these videos, regardless of the size of the RROI, to help evaluate the system better

4.2.2 Encoding Performance

Table 4.1 and Figure 4.2 present the encoding performance regarding run-time, and Table 4.2 and Figure 4.3 present the performance regarding the size of the encoded video file with various slicing strategies for all the videos. Each value in the tables represents the average run-time of five executions with the same parameters for the same video.

[1] The Crew video had only 36 frames.

Table 4.1: Encoding Performance (runtime in seconds)

		Wide-Strategy	Block-Strategy		O-Strategy		No Strategies
			Normal	Using FMO	Normal	Using FMO	
CIF (352x288)	**Akiyo**	38.75	38.95	41.44	38.85	40.09	38.64
	Flower	73.89	74.04	85.54	74.83	87.53	87.92
	Hall	84.36	75.82	99.89	80.47	95.93	88.46
	News	56.63	59.21	58.25	59.89	63.77	59.19
	Silent	64.40	68.34	73.66	67.54	73.69	63.27
4CIF (704x576)	**Ice**	257.66	257.52	263.64	267.02	271.43	261.03
	Crew	60.21	56.92	63.36	62.25	65.84	58.74
	Soccer	381.07	382.04	402.29	389.07	405.18	387.10
HD (1280x720)	**Parkrun**	1103.02	1101.04	1186.71	1086.28	1172.95	1127.12

Table 4.2: Encoding Performance (file-size in KB)

		Wide-Strategy	Block-Strategy		O-Strategy		No Strategies
			Normal	Using FMO	Normal	Using FMO	
CIF (352x288)	**Akiyo**	141	335	139	236	148	132
	Flower	2062	2330	2065	2189	2073	2051
	Hall	525	735	522	632	532	511
	News	340	543	337	442	347	328
	Silent	424	631	422	529	432	413
4CIF (704x576)	**Ice**	1036	1489	1037	1283	1044	1020
	Crew	363	462	363	416	362	358
	Soccer	2830	3435	2821	3157	2848	2801
HD (1280x720)	**Parkrun**	26234	27326	26200	27138	26247	26185

Figure 4.2 illustrates that among the slicing strategies, the O strategy causes the most increase in the encoding run-time. However, the increase from normal encoding time (i.e. when no slicing strategy is used) is very little. This goes to show that the proposed slicing strategies in the

encoding process are very efficient, and are suitable for use for commercial purposes. Since the search region for reference macroblocks is the largest when no slicing strategy is used, the run time can be more than when a slicing strategy is used. For example for the video *Parkrun* the run time when no strategy is used is more than the cases when several slicing strategies such as Wide, Block (non-FMO) and O (non-FMO) are used.

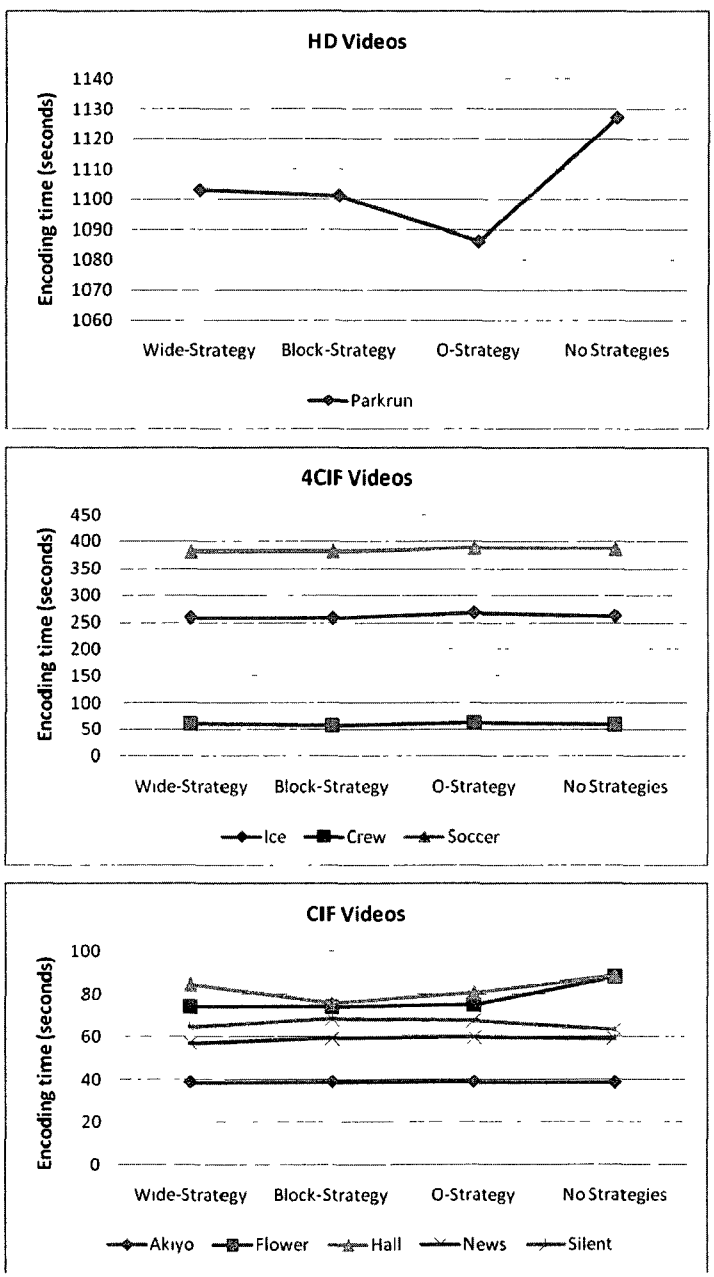


Figure 4.2: Encoding Performance (run-time in seconds)

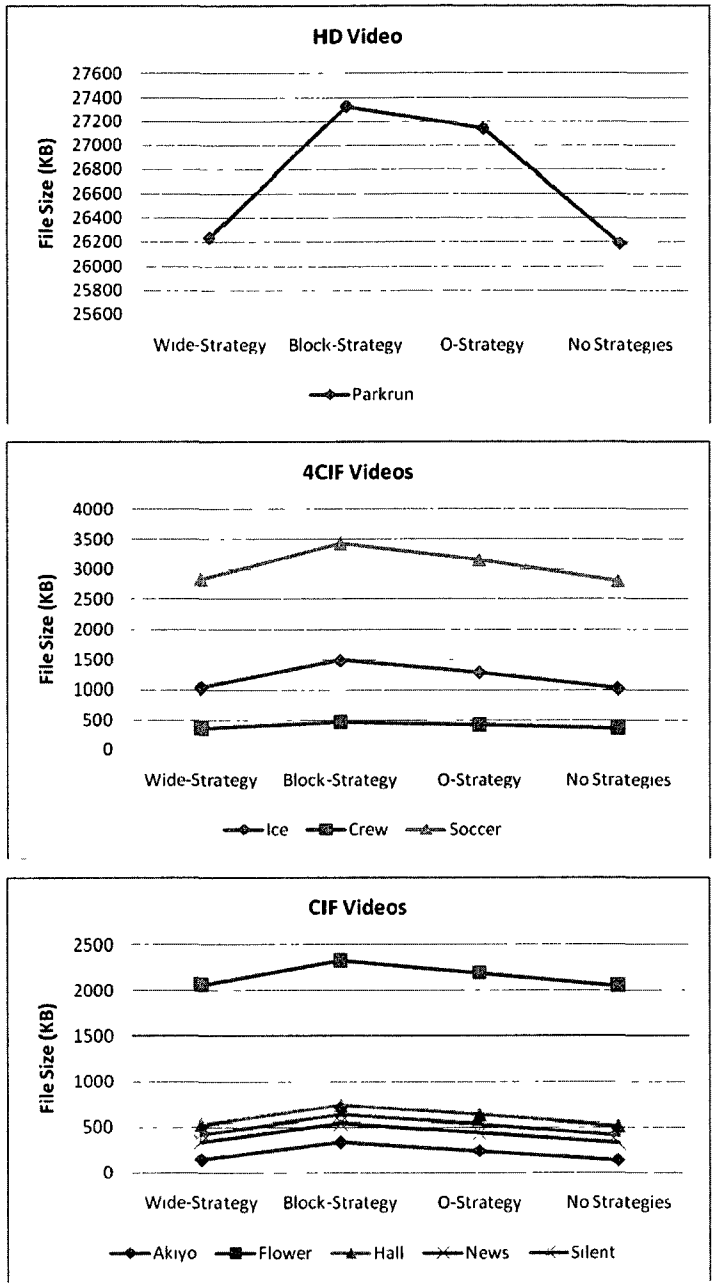


Figure 4.3: Encoding Performance (file-size in KB)

Figure 4.3 illustrates that among the slicing strategies, the Block strategy causes the largest increase in the file-size of the encoded video. This is predictable, because the Block strategy creates more slices for each frame than the other slicing strategies. Having an increased number of slices implies that there are fewer macroblocks available for reference in each slice, and that reduces the opportunities for compression in the output video stream. However, as Table 4.2

shows, when FMO is used with Block and O strategies, the slices have a larger region, and hence a larger number of macroblocks. As a result, the increase in file-size for FMO-encoded videos is much smaller. The Wide-strategy also creates a small number of slices, thus having the least effect on the compression, subsequently causing very small increase in file-size in the output video. This is confirmed by the data in Table 4.2 and Figure 4.3.

4.2.3 Adaptation Performance

Table 4.3 presents the performance of the adaptation process. The adaptation times for FMO-encoded videos are omitted from this table since the adaptation times are identical to the non-FMO videos.

Table 4.3: Adaptation Performance (runtime in seconds)

		Wide-Strategy	Block-Strategy	O-Strategy
CIF (352x288)	**Akiyo**	0.004	0.060	0.016
	Flower	0.000	0.028	0.020
	Hall	0.004	0.032	0.024
	News	0.004	0.032	0.024
	Silent	0.004	0.032	0.036
4CIF (704x576)	**Ice**	0.004	0.04	0.01
	Crew	0.004	0.004	0.004
	Soccer	0.004	0.08	0.02
HD Sequence (1280x720)	**Parkrun**	0.004	0.08	0.12

In order to compare this performance with other proposed adaptation schemes, I compare the adaptation time of each frame of my proposed method with the method described in [16]. The comparison is presented in Table 4.4.

Table 4.4: Comparison of Processing Time in Intermediary Nodes (in seconds)

Video Sequence	MB SKIP Mode Method [16]	Block Strategy	Wide Strategy	O Strategy
Flower	0.371	0.00028	0.00028	0.00016
Coastguard	0.460	0.000208	0.000209	0.000144

As the data shows, the adaptation process is extremely fast in generating the adapted video from the gBSD description and the encoded video, and the performance is a remarkable improvement above the existing methods.

4.2.4 Bandwidth Performance

To measure the bandwidth performance of the slicing strategies, we use the formula we deduced in Chapter 3.

Table 4.2 presents the file sizes of encoded videos with and without slicing strategies (i.e. SS and ES respectively). The file sizes of the cropped videos (CS) after spatial adaptation are presented in Table 4.5.

Table 4.5: Cropped File Size (in KB)

		Wide-Strategy	Block-Strategy		O-Strategy	
			Normal	Using FMO	Normal	Using FMO
CIF (352x288)	**Akiyo**	94	111	118	118	89
	Flower	1069	1200	943	1014	483
	Hall	343	301	352	317	259
	News	254	208	259	270	205
	Silent	271	257	328	301	219
4CIF (704x576)	**Ice**	659	708	659	422	333
	Crew	224	234	224	153	134
	Soccer	1568	1487	1261	822	710
HD Sequence (1280x720)	**Parkrun**	19220	15958	15483	11910	11523

If no spatial adaptation is available, then the entire encoded video needs to be delivered to the client. Consequently, a lot of bandwidth is wasted since the client is not interested, or incapable of utilizing the entire video. We demonstrate the usefulness of using a slicing strategy proposed in this thesis for spatial adaptation by comparing the size of a video-stream after adaptation with the size of the video-stream when no adaptation is used. The comparison is presented in Figure 4.4. As the figure clearly demonstrates, the size of the bitstream is always decreased, sometimes drastically, when adaptation is used compared to the size when spatial adaptation is not performed on the video.

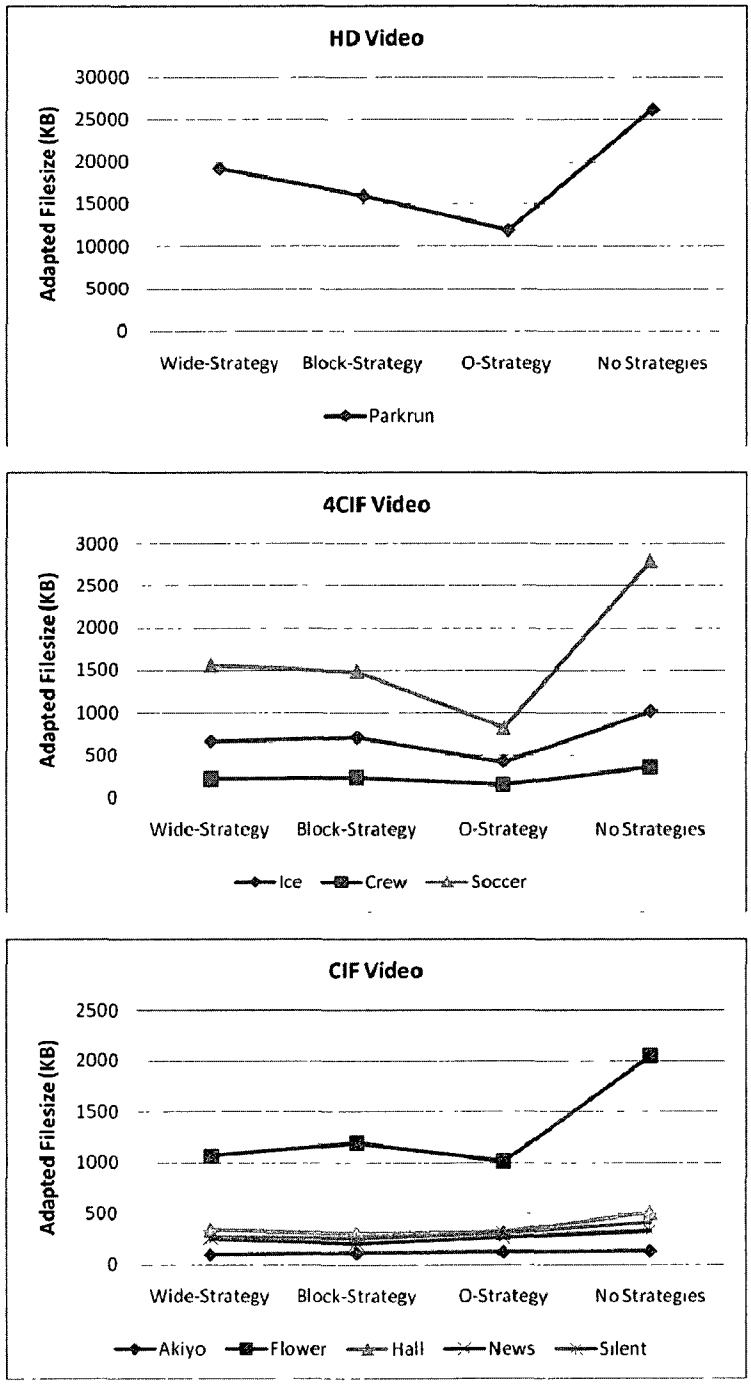


Figure 4.4: Comparison between Adapted Filesize and Non-adapted Filesize

Table 4.5 presents the CS values of the videos. From these values, the minimum demand for cropped videos required to make the spatial adaptation process profitable in terms of

bandwidth can be computed using the equation provided in the previous chapter. This result is presented in Table 4.6.

Table 4.6: Required Demand for Cropped Video for Profitable Adaptation

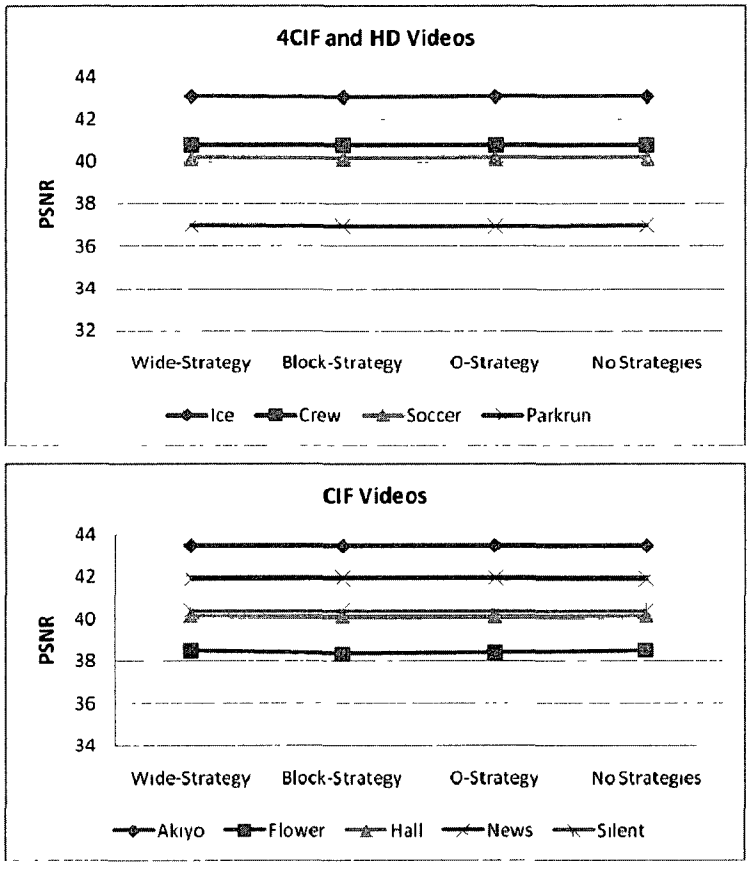
		Wide-Strategy	Block-Strategy		O-Strategy	
			Normal	Using FMO	Normal	Using FMO
CIF (352x288)	**Akiyo**	19.15%	84.23%	15.56%	73.24%	29.63%
	Flower	0.56%	12.48%	0.71%	6.59%	1.11%
	Hall	3.25%	34.95%	2.57%	22.49%	4.79%
	News	4.88%	47.88%	3.7%	32.76%	7.51%
	Silent	3.33%	40.6%	2.74%	26.67%	5.62%
4CIF (704x576)	**Ice**	1.7%	33.62%	1.8%	22.12%	2.53%
	Crew	1.86%	28.26%	1.86%	18.01%	1.49%
	Soccer	1.06%	18.98%	0.73%	11.62%	1.71%
HD Sequence (1280x720)	**Parkrun**	0.19%	4.19%	0.06%	3.52%	0.24%

The data in the table shows that in most cases, the wide strategy can be profitable even if only 5% of the clients request the cropped video. The table also shows that in order for Block-strategy to be profitable in terms of bandwidth, the cropped video will need to be delivered to a larger ratio of the clients. However, if FMO is used, then Block and O-strategies can also be equally profitable as the Wide slicing strategy. While bandwidth consumption is not the only measure to decide whether spatial adaptation can be profitable, this measurement can be useful for video providers to determine whether availing spatial adaptation to clients is going to be profitable in terms of bandwidth consumption.

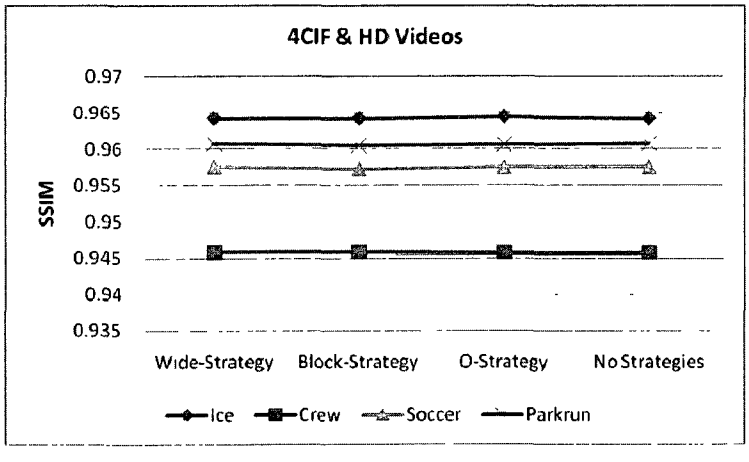
4.2.5 Video Quality Performance

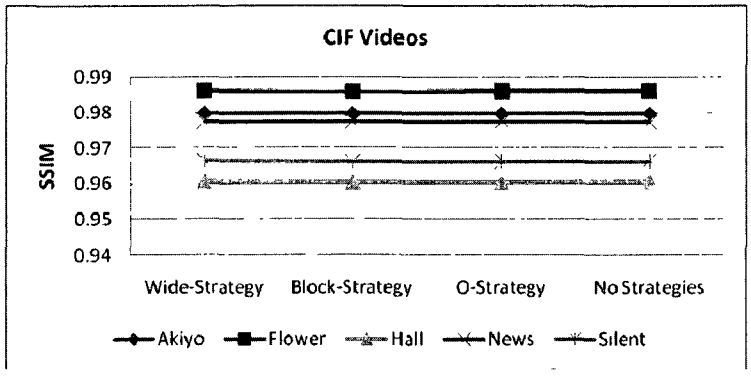
One of the major concerns of the spatial adaptation scheme is to not have a detrimental effect on the video quality. Peek Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM) index are generally used for measuring the perceptual quality of encoded videos [32]. For this reason we compute the PSNR and SSIM of videos encoded using different slicing strategies and no slicing strategy. These values are presented in Figure 4.5. As the figure clearly demonstrates, the

change in both PSNR and SSIM when one of the slicing strategies is used is very little, and can be considered negligible.



(a) Effect of Slicing Strategies on PSNR





(b) Effect of Slicing Strategies on SSIM

Figure 4.5: Quality of Adaptable Videos

4.3 Live Adaptation

An adaptation system that allows spatial adaptation on live streaming video can be especially useful for video conferencing, and make it possible for users to participate even when they have only handheld devices like iPhone etc. available to them. Commercial news providers can also provide a spatially adapted version of a live news broadcast video for commuting viewers.

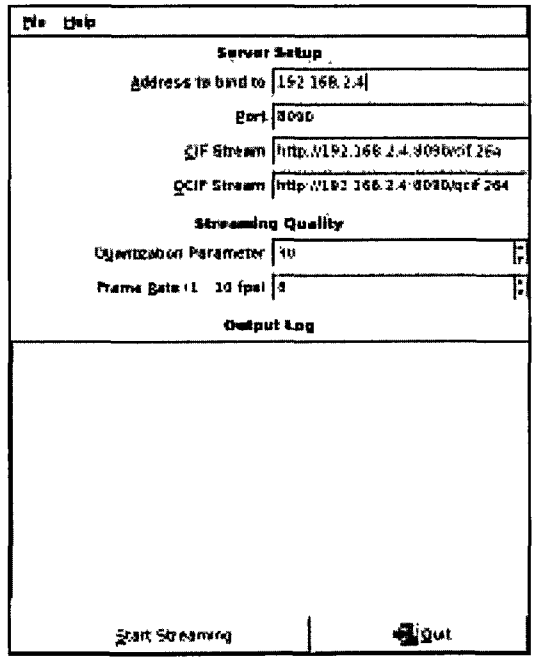


Figure 4.6: Live Adaptation Server

For this reason, a live adaptation system was developed as part of the thesis to demonstrate the efficiency and applicability of the proposed spatial adaptation scheme in real world commercial

systems. For the live adaptation system, `FFmpeg` was used to capture live video from creative laptop intregated webcam [26]. For serving the streaming video to clients, `FFserver` of `FFmpeg` suite was used. The entire system was integrated into a graphical user interface (GUI) using python programming language [33] and GTK+ toolkit [34]. The GUI is presented in Figure 4.6.

The GUI allows specifying the server address and the port number to use for the streaming video. Clients connect to the server at this port to view the video. Clients can request either a CIF stream or a QCIF stream using different names for the video. The name of the streams, i.e. the URLs clients will use, can also be configured in the GUI. As explained in Chapter 3, Usage Environment Descriptions are not supported by the popular media players. So the users have to explicitly request for either the CIF or QCIF version of the video depending on their respective device capabilities. To allow finer control of the quality of the video, it is possible to change the *Quantization Parameter (QP) and the frame rate for the video from the GUI. Both the CIF and* QCIF streams are served simultaneously, and multiple clients can connect to either stream and receive the video at the same time.

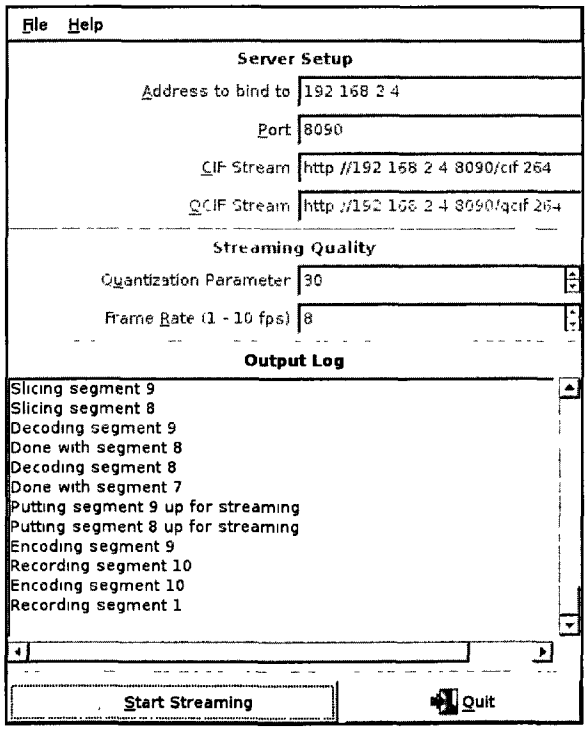


Figure 4.7: Live Adaptation in Progress

When the streaming is started, by clicking on the 'Start Streaming' button (or from 'File ➔ Play' menu), ffmpeg is used to capture the raw video in CIF (i.e. 352x288) format. The captured raw video is encoded into H.264 using the specified QP and frame-rate. O-slicing strategy is used when encoding the video, where the ROI distiller is set to always return the center of the video of QCIF dimension (i.e. 176x144). Clients requesting the full video are delivered this unadapted compressed video. For clients who request the spatially adapted video, spatial adaptation scheme proposed in this thesis is used to generate the cropped video stream from the CIF video stream and then delivered to clients. As can be seen from the data in Table 4.1 and Table 4.3, the time required to generate the QCIF video by cropping a compressed CIF video is much faster than to encode the video in QCIF from the raw video. Therefore, this approach at spatial adaptation is very suitable for practical use.

In this chapter of the thesis, I presented details about an implementation of the proposed spatial adaptation mechanism. The performance of the adaptation scheme was presented and compared with mechanisms proposed by others in the literature. The results show that the spatial adaptation scheme proposed in this thesis outperforms existing methods, and is suitable for use in commercial products. A system for adapting live video streams using the proposed adaptation method was also presented.

Chapter 5 : Conclusion and Future Work

With the advent of a deluge of handheld and mobile devices with video playing capabilities, spatial adaptation of digital video content has gained significant importance for serving the desired content in a feasible and efficient way. This thesis presents a very efficient adaptation technique that can crop a video before delivering it to the consumer. The adaptation process takes the capability of the client device into account and produces video optimized for that specific device. So each client receives video suitable for the specific device the client is using. The adaptation technique eliminates the need for cascaded transcoding or multiple pre-processed video streams in the media server or the intermediary nodes. The final output of the adaptation scheme is compliant to the H.264 standard, and any standard client is able to play the video. The client is not required to transcode the video or perform other operations on it before it can play the video. So the adaptation scheme is completely transparent to the client and does not impose any additional dependencies on it.

5.1 Summary of Contribution

The adaptation technique proposed in this thesis has two steps: first is the encoding of the video, and second is the adaptation of the encoded video. While encoding a video, ROI detection algorithms are used to find the region of interest in the video. This detected ROI is then distilled intelligently so that the distilled ROI can be viewed by the client devices of limited capabilities. Each frame in the video is then divided into slices so that the distilled ROI is encoded into a number of essential slices, and the rest of the regions are encoded into a number of disposable slices. A number of intelligent slicing strategies have been proposed in this thesis. The slicing strategy to be used to encode a frame depends on the detected location and dimension of the ROI in the frame. The essential slices, i.e. the slices containing the distilled ROI are always delivered to the clients, while the disposable slices, i.e. the slices that do not contain the ROI, are delivered to a client only if the client requested for that region and the client device is capable of displaying the region contained in the slice. The encoder also generates the bitstream syntax description using the generic Bitstream Syntax Schema (gBS) as described in MPEG-21 framework. This gBSD contains information about the dimension and location of the distilled ROI in a frame, the dimensions of the essential and disposable slices and the offsets and

lengths of the slices in the encoded bitstream. The generated bitstream is completely H.264 standard compliant.

The second step of the adaptation process happens at the Adaptation Engine. Depending on the client request and the capabilities of the client device, the adaptation engine can crop disposable slices out of the bitstream and deliver only slices that belong to the requested region in the frame to the client. The adaptation engine is able to efficiently and quickly drop the undesired slices from the bitstream by processing the gBSD description generated by the encoder, and using an XSLT style sheet presented in this thesis. The adapted video stream generated this way by dropping slices is also completely H.264 standard compliant.

Multipoint adaptation is possible using the adaptation technique proposed in this thesis by updating the offsets of the slices in the adapted bitstream description. The adapted bitstream created as described in this thesis conforms to the gBS Schema. Preserving this adapted bitstream description and the adapted video-stream in an intermediary node makes it possible to apply farther spatial adaptation to the stream in the node when required. The proposed scheme in this thesis thus leads to a multipoint adaptation design.

This thesis also details an implementation of the proposed adaptation technique. The prototype developed is used to perform adaptation on a number of video streams of different categories, e.g. CIF, 4CIF, HD videos with both low and high motions. The encoding time of a video for different slicing strategies, adaptation time of dropping slices from an encoded video, the bitstream size of the video before and after adaptation, perceptual video quality etc. were measured. The measurements show that the adaptation technique is very efficient and has negligible effect on the processing time and quality of the video. Some results were also compared with performance metric of one existing adaptation technique. The comparison results show that the proposed adaptation scheme in this thesis is competitive and performs better than the existing methods.

Live adaptation is challenging because the adaptation process has to be very fast so that it does not hinder the spontaneity of the live video-stream. Since the spatial adaptation technique presented in this thesis performs the adaptation in compressed domain, it is very fast, and

incurs only negligible delay. For this reason, the challenge of spatial adaptation of live video streams is now achievable, applicable and successful.

5.2 Future Work

Only one type of FMO, namely Type 2, is explored in this thesis for the various slicing strategies. Other types of FMO, especially Type 3, can also be very suitable for spatial adaptation. In addition to the slicing strategies presented in this thesis, additional video-genre-specific slicing strategies may be devised. For example, O-strategy is very suitable for news broadcasts, but it may not perform very well for sports broadcasts. A specific strategy may be developed that optimizes performance for sports broadcasts, such as racing, golf etc. FMO Type 6 may be useful for devising such genre-specific strategies.

A number of different ROI detection algorithms can be used with the various slicing strategies described in this thesis to determine the effect of the ROI algorithms on the adaptation performance. An interesting research project could involve studying these effects and more closely integrate the ROI detection algorithm with the slicing strategies.

References

1. C.-S. Li, R. Mohan and J. R. Smith, "Multimedia Content Description in the InfoPyramid," IEEE Proceedings of Int. Conf. Acoust., Speech, Signal Processing (ICASSP) , Seattle, WA, Special Session on Signal Processing in Modern Multimedia Standards, May 1998.
2. E. Kasutani, "Investigation Report on Universal Multimedia Access," EPFL-ITS Technical Report 04-04, January 2004.
3. ISO/IEC JTC1/SC29/WG11, Overview of Scalable Video Coding, April 2008.
4. ISO/IEC JTC1/SC29/WG11N5525, MPEG-7 Overview v.9, March 2003.
5. ISO/IEC JTC1/SC29/WG11/N5231, MPEG-21 Multimedia Framework, October 2002.
6. ISO/IEC 21000-7, Information Technology — Multimedia Framework (MPEG-21) — Part 7: Digital Item Adaptation, October 2004.
7. T. Wiegand, G.J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560- 576, July 2003.
8. Encoding.com, http://blog.encoding.com/
9. I. E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia," Wiley, ISBN. 0-470-84837-5, 2004.
10. P. Lambert, W. De Neve, Y. Dhondt, and R. Van de Walle, "Flexible macroblock ordering in H.264/AVC," Journal of Visual Communication and Image Representation, vol. 17, no. 2, pp. 358- 375, April 2006.
11. P. Zhang, Y. Lu, Q. Huang, and W. Gao, "Mode Mapping Method for H.264/AVC Spatial Downscaling Transcoding," Proceedings of International Conference on Image Processing (ICIP '04), vol. 4, pp. 2781-2784, Singapore, October 2004.
12. H. Shen, X. Sun, F. Wu, H. Li and S. Li, "A Fast Downsizing Video Transcoder for H.264/AVC with Rate-Distortion Optimal Mode Decision," Proceedings of IEEE International Conference on Multimedia Expo (ICME), vol. 1, pp. 2017-2020, Toronto, Canada, July 2006.
13. J. Cock, S. Notebaert, K. Vermeirsch, P. Lambert, R. Walle, "Efficient Spatial Resolution Reduction Transcoding for H.264/AVC," Proceedings of IEEE International Conference on Image Processing (ICIP), pp. 1208-1211, 2008.

14. H. Li, Y. Wang, and C. Chen, "An Attention-Information-Based Spatial Adaptation Framework for Browsing Videos via Mobile Devices," EURASIP Journal on Advances in Signal Processing, vol. 2007, no. 25415, 2007.
15. Y. Wang, H. Li, Z. Liu, and C. Chen, "Attention Information Based Spatial Adaptation Framework for Browsing Videos Via Mobile Devices," Proceedings of Pacific Rim Conference on Multimedia (PCM), pp. 788-797, Hangzhou, China, 2006.
16. H. K. Arachchi, S. Dogan, H. Uzuner and A. M. Kondoz, "Utilizing Macroblock SKIP Mode Information to Accelerate Cropping of an H.264/AVC Encoded Video Sequence for User Centric Content Adaptation", Proceedings of Intl. Conf. on Automated Production of Cross Media Content for Multi-Channel Distribution, 2007.
17. P. Lambert, D. De Schrijver, D. Van Deursen and W. De Neve, "A Real-Time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC," Lecture Notes in Computer Science, Advanced Concepts for Intelligent Vision Systems (ACIVS), pp. 442–453, 2006.
18. P. Lambert and R. Van de Walle "Real-time Interactive Regions of Interest in H.264/AVC," Journal of Real-Time Image Processing, pp. 67–77, 2009.
19. Y. F. Ma and H. J. Zhang, "A Model of Motion Attention for Video Skimming," Proceedings of IEEE International Conference on Image Processing, pp. 129-132, 2002.
20. Y. F. Ma, L. Lu, H. J. Zhang and M. Li, "A User Attention Model for Video Summarization," Proceedings of ACM Multimedia, pp. 533-542, 2002
21. C. C. Ho, W. H. Cheng, T. J. Pan and J. L. Wu, "A User Attention Based Focus Detection Framework and its Applications," Proceedings of Pacific-Rim Conference on Multimedia, Singapore, 2003.
22. W. H. Cheng, W. T. Chu and J. L. Wu, "A Visual Attention Based Region of Interest Determination Framework for Video Sequences," IEICE - Transactions on Information and Systems, pp. 1578-1586, 2005.
23. J. W. Chen, M. J. Chen and M. C. Chi, "Region of Interest Video Coding Based on Face Detection," Proceedings of the Third IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia Information Processing, pp. 1201-1211, 2002.
24. H.264/AVC JM Reference Software, http://iphome.hhi.de/suehring/tml/
25. x264 - A Free H.264/AVC Encoder, http://www.videolan.org/developers/x264.html
26. FFmpeg, http://www.ffmpeg.org/

27. VLC Media Player, http://www.videolan.org/vlc/
28. Google Video, http://video.google.com
29. LibHTTP, http://www.hughes.com.au/products/libhttpd/
30. The XML C Parser and Toolkit of Gnome, http://xmlsoft.org/
31. The XSLT C Library for GNOME, http://xmlsoft.org/xslt/
32. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, April 2004.
33. Python Programming Language, http://www.python.org/
34. The GTK+ Project, http://www.gtk.org/