Enhancements & Optimizations to H.264/AVC Video Coding

by

Xiongwen Li

A doctoral thesis submitted in
partial fulfillment of the
requirements for the degree
of

Doctor of Philosophy

Department of Computer Science
Loughborough University
October 2007

Supervisors: Dr. E. A. Edirisinghe, Dr. H. E. Bez

Director of Research: Prof. Sameer Singh

Abstract

The H.264/AVC video coding standard offers enhanced performance compared to previous coding standards in terms of both rate-distortion (R-D) performance and functionality. In particular, its superior rate-distortion performance has resulted in a significant interest in its practical application in many different domains ranging from multimedia to security and surveillance. As a result in the recent past many successful research attempts have been made in further improving its efficiency and extending its application domains. This thesis provides two novel contributions: an object-based extension that is capable of extending H.264/AVC's effective use in video surveillance applications and a multi-objective optimization framework that can be used to enhance H.264/AVC's use in any general application area.

The first part of the thesis presents the design of a novel Shape Adaptive Integer Transform (SA-IT) and associated quantization procedures, to enable the coding of arbitrarily shaped video objects within H.264/AVC. The novel transform specifically enables maintaining the 16-bit integer arithmetic requirements of the standard. The thesis also presents the introduction of a novel binary shape coding strategy to H.264/AVC that is proved to be more efficient as compared to the shape coding scheme adopted by MPEG-4 visual. In addition, the slice group structure of the H.264/AVC is further extended and effectively used with flexible macroblock ordering (FMO) to provide support for object-based coding. The thesis shows that the proposed object-based CODEC provides the ability to selectively code images (video frames), enabling the ability to reconstruct important, pre-defined, foreground objects at high quality levels, leading to applications in the security & surveillance industry. Experimental results are provided to prove that the above functional enhancements come at no additional cost to the R-D performance.

The second part of the thesis provides a novel framework for the performance optimization of a standard H.264/AVC CODEC. The effect of different coding

parameters on the video quality, bit rate, computational complexity and memory utilization are initially investigated in detail, leading to the identification of significant coding parameters. This knowledge is subsequently used in developing a multi-objective optimization framework for a H.264/AVC CODEC. In particular the use of the proposed framework in the joint minimization of the distortion and computational complexity, in a memory and bandwidth constrained environment is presented. The framework produces a set of optimal or near optimal coding parameters (which can be used for optimizing the trade-off between complexity, memory, rate and distortion) that can be directly used in achieving the optimum performance setting of a H.264/AVC CODEC. This is the first attempt in the literature that has investigated the joint complexity-memory-rate-distortion (C-M-R-D) optimization in video coding.

The work presented in this thesis contributes in extending the use of H.264/AVC to new application domains such as CCTV surveillance. Further the proposed multi-objective optimization framework and the associated methodologies can be used generally for the performance optimization of any video coding standard.

Xiongwen Li

October 2007

Acknowledgements

I would like to take this opportunity to express my sincere thanks to all who helped me to successfully complete this work.

First, I would like to express my heartfelt gratitude to my supervisors, Dr Eran Edirisinghe and Dr Helmut Bez for their direction, advice and support. Especially to Dr Eran Edirisinghe, who provided high quality guidance, fullest support and helped review my thesis.

I would also like to express my thanks to the departmental technical staff member, Mr. Samra. My thanks are also due to my Digital Imaging colleagues, Dammike, Ken, Njad, Moi, Rupesh and Iffat. My special thanks also go to Hector and Usman, new members of the research group who helped in proof reading.

I would like to take this unique chance to thank my family members. I am profoundly thankful to my wife, for her love, faith and understanding during my study. I wish to thank my parents, who have always given me immeasurable and unconditional support. I also wish to thank my son, who is one year old and brought joy to my life during my final year of study.

Finally, special thanks also go to Dr Christos Grecos for his help and advice in the later part of my research.

Contents

List of Figures

List of Tables

Abbreviations

ADSL-2	-	Asymmetric Digital Subscribe Line Transceivers 2
AHBS	-	Adaptive Hexagon-based Search
AVC	-	Advanced Video Coding
ASO	-	Arbitrary Slice Ordering
BAB	-	Binary Alpha Block
BG	-	Background
B-VOP	-	Bidirectionally predicted Inter-coded VOP
CABAC	-	Context-based Adaptive Binary Arithmetic Coding
CAE	-	Context-based Arithmetic Encoding
CAVLC	-	Context-based Adaptive Variable-Length Coding
CBP	-	Coded Block Pattern
C-D	-	Complexity-Distortion
CIF	-	Common Intermediate Format, a video format
C-M-R-D	-	Complexity-Memory-Rate-Distortion
CPU	-	Central Processing Unit
C-R-D	-	Complexity-Rate-Distortion
DCT	-	Discrete Cosine Transform
DVD	-	Digital Versatile Disc
DWT	-	Discrete Wavelet Transform
EA	-	Evolutionary Algorithm
EPZS	-	Enhanced Predictive Zonal Search, a motion estimation algorithm of H.264
FG	-	Foreground
FME	-	Fast Motion Estimation
FMO	-	Flexible Macroblock Ordering
FPGA	-	Field Programmable Gate Array
FS	-	Full Search, a motion estimation algorithm of H.264
GA	-	Genetic Algorithm
H.264	-	A Video Coding Standard

HVS	-	Human Visual System
ISO/IEC	-	International Standards Organization, International Electrotechnical Commission
IT	-	Integer Transform
ITU-T	-	International Telecommunications Union, Telecommunication Standardization Sector
I-VOP	-	Intra-coded rectangular VOP, progressive video format
LPE	-	Low-Pass Extrapolation
MC	-	Motion Compensation
ME	-	Motion Estimation
MF	-	Multiplication Factor
MOEA	-	Multi-Objective Evolutionary Algorithm
MOGA	-	Multi-Objective Genetic Algorithm
MOO	-	Multi-Objective Optimization
MOOP	-	Multi-Objective Optimization Problem
MPEG	-	Motion Picture Experts Group, a Committee of ISO/IEC
MPEG-4	-	A Multimedia Coding Standard
MSE	-	Mean Squared Error
MVD	-	Motion Vector Difference
MVDS	-	Motion Vector Difference for Shape
MVPS	-	Motion Vector Predictor for Shape
MVS	-	Motion Vector for Shape
NAL	-	Network Abstraction Layer
NALU	-	Network Abstraction Layer Unit
NSGA	-	Non-Dominated Sorting Genetic Algorithm
P-D	-	Power-Distortion
PPS	-	Picture Parameter Set
P-R-D	-	Power-Rate-Distortion
PSNR	-	Peak Signal to Noise Ratio, an objective quality measure
P-VOP	-	Inter-coded rectangular VOP, progressive video format
QCIF	-	Quarter Common Intermediate Format
QP	-	Quantization Parameter

R-D	-	Rate-Distortion
RF	-	Rescaling Factor
RGB	-	Red/Green/Blue colour space
RMSE	-	Root Mean Square Error
RBSP	-	Raw Byte Stream Payload
ROI	-	Region of Interest
SA-DCT	-	Shape Adaptive Discrete Cosine Transform
SA-DWT	-	Shape Adaptive Discrete Wavelet Transform
SA-IT	-	Shape Adaptive Integer Transform
SIF	-	Source Input Format, a video format
SPS	-	Sequence Parameter Set
SQCIF	-	Sub Quarter CIF
SVC	-	Scalable Video Coding
UMHS	-	Unsymmetrical Multi-Hexagon Search, a motion estimation algorithm of H.264
SUMHS	-	Simplified Unsymmetrical Multi-Hexagon Search, a motion estimation algorithm of H.264
VCEG	-	Video Coding Experts Group
VCL	-	Video Coding Layer
VLC	-	Variable Length Coding
VO	-	Video Object
VOP	-	Video Object Plane

Chapter 1 Introduction

1.1 Problem Statement

The new video coding standard, H.264/AVC [1, 11] (hereafter referred to as H.264), offers enhanced performance compared to other previous video coding standards (e.g. MPEG-2, MPEG-4 Visual, H.263 etc.) in terms of both compression efficiency (approximately a 50% bit rate saving is achievable for equivalent perceptual quality [14]) and flexibility (provides for a much wider range of applications). However, one noteworthy functionality absence in H.264 specifically in comparison to MPEG-4 Visual (Part-2) [5] (hereafter referred to as MPEG-4) is the capability of coding arbitrarily shaped video objects. Such a capability can enable H.264 to enhance coding efficiency and flexibility by providing the means for coding selected arbitrarily shaped object(s)/region(s) at a higher quality level as compared to the visually non-important background. This is a highly desirable functionality for applications in highly bandwidth constrained mobile telephony and in CCTV surveillance systems where video quality is of utmost importance for post incidence analysis. In mobile telephony, the limited size of screen solicits the acceptability of encoding the background of a video frame at a lower quality as compared to the foreground, for e.g. representing speaker in a conversational application scenario, under bandwidth constraints. In CCTV applications, provided a pre-processing stage can separate important objects/regions (for the purpose of post incidence analysis), these areas can be coded at a much better quality level as compared to the background. Though region/object based coding has been well investigated under JPEG-2000, MJPEG-2000 and MPEG-4, no attempt has been made to include object-based coding to H.264. The inclusion of object-based coding concept within H.264 requires resolving many research challenges, particularly due to the need to maintain compliance with its complicated coding architecture, efficiency and integer arithmetic requirements.

Due to the high demand of applications requiring real-time video coding, mainly supported by capture, processing, transmission and display devices/mediums that are running under constraints, recently a significant amount of research momentum has been gathered in the area of optimizing video coding standards. With the recent standardization of H.264, the video optimization research has mainly been focused on this standard. In literature, the existing contributions to the optimization of H.264 have focused on reducing computational complexity, rate and distortion, either individually or pair wise. A single attempt has also been made at the joint complexity, rate and distortion optimization of H.263 [47]. These optimizations have focused on algorithmic enhancements or improvements, such as fast algorithms for motion estimation (ME), fast algorithms for coding mode selection and block skipping. A careful analysis has revealed that none of these optimizations has attempted to optimize a joint complexity, memory, rate and distortion (C-M-R-D) based on an optimal selection of H.264's numerous coding parameters. Further the existing optimizations have either been limited solely to the encoder or to the decoder. In other words no attempt has been made in optimizing an entire H.264 CODEC.

It is known that the performance of H.264 depends on a large set of coding parameters, including the choice of different fast motion estimation algorithms and rate-distortion optimization modes etc. A correct set (i.e. a combination) of coding parameters can enable a H.264 CODEC to achieve optimum performance. Therefore, the right selection of parameters is an open research question. However the choice of parameters will depend on the source video, coding objectives and system constraints. Therefore it is desirable to develop an optimization framework that yields the appropriate coding parameters to that can jointly optimize the trade-off between complexity, memory utilization, rate and distortion.

1.2 Aim and Objectives

The aim of the research presented in this thesis is to enhance the functionality and optimize the performance of a H.264 video CODEC.

To this effect two key objectives are to be met; (i) enhance functional flexibility of H.264 by introducing the ability of coding arbitrarily shaped object(s), and (*ii*) optimize the trade-off between computational complexity, memory utilization, rate and distortion by choosing the right coding parameters.

In order to fulfil the first objective, a novel DCT-based shape adaptive integer transform that is capable of efficiently coding the boundary blocks of an arbitrarily shaped object has to be defined. The said transform should be able to maintain integer arithmetic requirements of a standard H.264 CODEC during the transform-quantization stages. In addition to developing the above shape adaptive integer transform algorithm, a novel shape coding algorithm needs to be developed for coding shape information of video objects. In addition, the slice group structure of the H.264 needs to be modified and extended so that an object-based H.264 CODEC architecture can be achieved, which should provide the ability to selectively code images (video frames), enabling the ability to reconstruct important, pre-defined, foreground objects at high quality levels.

In order to achieve the second objective of this thesis, an H.264 CODEC's detailed performance needs to be first investigated. The effect of using different coding parameters, on computational complexity, memory utilization, rate and distortion should be analyzed. The coding parameters that have the most significant impact on the above objectives can then be used to obtain the fitness functions for each objective. These can then be utilized within a multi-objective optimization framework that can jointly optimize complexity-memory-rate-distortion.

The above research and development objectives are implemented based on the H.264 reference software model, JM 10 (referred to as H.264 CODEC within the context of this thesis) [51].

1.3 Thesis Contributions

The research and development work carried out in fulfilling the above research aims and objectives have resulted in a number of original contributions. They are:

- Proposing a shape adaptive integer transform (SA-IT) for coding the texture of arbitrarily shaped objects in H.264 video coding.

- Proposing a new shape coding algorithm based on the MPEG-4 shape coding methodology for coding shape information of video objects.

- Design, development and implementation of an object-based coding extension to the H.264 standard.

- A detailed investigation of the effects of different coding parameters on a H.264 CODEC's computational complexity, memory requirement, rate and distortion.

- The development of a multi-objective optimization framework for a H.264 video CODEC that is capable of jointly optimizing computational complexity, memory utilization, rate and distortion.

1.4 Organisation of the Thesis

For clarity of presentation, the thesis is organized as follows:

Chapter 2 provides fundamental background knowledge on video coding, an overview of the H.264 video coding standard and of arbitrarily shaped video object coding in MPEG-4 video coding standard.

Chapter 3 reviews different arbitrarily shaped object coding techniques. The advantages and disadvantages of each approach are critically compared and discussed. The chapter further provides an insight into video CODEC optimization methods proposed in previous literature and highlight their limitations.

Chapter 4 proposes a novel mathematical transform, the shape adaptive integer transform (SA-IT) that can be used to code the texture of arbitrarily shaped video objects.

Chapter 5 designs, develops and implements an object-based coding framework within a standard H.264 CODEC. The chapter further compares the performance of the object-based H.264 CODEC extension with that of a standard H.264 CODEC.

Chapter 6 investigates the effect of different coding parameters of a standard H.264 CODEC's computational complexity, memory requirements, rate and distortion. Further it concludes which coding parameters have the most significant effect on the four selected objectives.

Chapter 7 presents a multi-objective optimization framework for a H.264 CODEC that is capable of joint optimization of computational complexity, memory utilization, rate and distortion, collectively considering both the encoder and the decoder.

Finally, Chapter 8 concludes the thesis with a summary of contributions and future directions of research.

Chapter 2 Introduction to H.264/AVC and Object-based Video Coding

2.1 Introduction

With the widespread adoption of technologies such as digital television, Internet streaming video and DVD-Video, video coding (video compression) has become an essential component of broadcast and entertainment media [11]. Further the presence of a vast amount of CCTV cameras capturing real-time footage of public places has recently increased the use of video coding algorithms in video surveillance data storage and transmission. It is well known that raw or uncompressed digital videos require expensive resources for storage, transmission and processing of video data. For example, using a typical video resolution of 352 x 288 pixels with 3 bytes of colour data per pixel, playing at 30 frames per second, one second of the video requires 8.9 Megabytes of storage. At this rate, a 5.34 Gigabytes hard disk can only store 10 minutes of the video. Furthermore, if this video is to be transmitted in real time through the internet, it requires a channel bandwidth of 73 Mbps, which is 6 times the bandwidth (12 Mbps) of Asymmetric Digital Subscribe Line Transceivers 2 (ADSL2), the current broadband internet service. Moreover, the processing power needed to handle such massive amounts of data would make video processing hardware very expensive. For these reasons, compression technology is an essential requirement in digital video.

A large amount of statistical and subjective redundancy exists in digital video sequences. Video compression techniques are designed to reduce the size of data required for storage and transmission by removing both statistical and subjective redundancy. The compressibility of a video not only depends on the amount of redundancy in the source video sequence, but also on the compression technique used for coding [2]. Video compression techniques are primarily classified into lossless

coding and lossy coding. Lossless coding techniques usually exploit the statistical redundancy in image and video data so that the identical data can be reconstructed perfectly at the decoder. However, these techniques can only obtain a modest amount of compression. On the other hand, lossy coding methods operate by removing subjective redundancy in spatial, temporal and/or frequency domains to achieve a significant decrease in the file size (i.e. high compression ratio) at the expense of video quality.

Over the last two decades, video coding has been a very active field of research and development, and many coding techniques have been proposed and developed by companies, researchers and international standardisation authorities. ISO Motion Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) are two formal organizations that have developed fully fledged video coding standards. These standards have been designed specifically for a variety of video applications. The ISO/IEC MPEG developed the MPEG series: MPEG-1 [3], MPEG-2 [4], MPEG-4 [5], MPEG-7 [6] and MPEG-21 [7]. They have addressed the problems of video storage, streaming and broadcasting through internet and mobile networks. The ITU-T video coding standards are named as the H.26x series (H.261 [8], H.262 [9], H.263 [10] and H.264 [1]), are have been specifically designed for applications such as video conferencing and video telephony [2].

For clarity of presentation this chapter is divided into several sections. Section 2.2 introduces the reader to the fundamentals of video coding. Section 2.3 presents an overview of the H.264 video coding standard, with particular emphasis to areas of research focus of this thesis. Section 2.4 introduces object-based coding in general, which is followed by Section 2.5 that presents object-based video coding as used in MPEG-4. Section 2.6 presents potential applications of object-based video coding. Finally, Section 2.7 concludes the chapter.

2.2 Fundamental Concepts of Video Coding

2.2.1 Terminology and Abbreviations

The basic terminology and abbreviations used generally in video coding can be listed as follows [2]:

- **Pixel:** A colour element at one position in a displayed image.
- **Luminance (or Luma):** A sample or array representing a video brightness signal, often symbolized as Y.
- **Chrominance (or Chroma):** A sample or array representing a blue or red video colour difference signal, often symbolized as C_b and C_r, or U and V.
- **Sample:** A luma or chroma component at one position in a video frame.
- **Frame:** A set of samples representing a single time instant of a progressive video signal. A video frame consists of one array of luma samples and two arrays of chroma samples.
- **Frame rate (frame frequency):** The number of frames or images that are projected or displayed per second. Frame rate is often expressed in frames per second (fps), or simply in hertz (Hz).
- **Resolution:** The dimensions of a video frame or an image, in pixels.
- **Macroblock (MB):** A 16 x 16 array of luma pixels (Y) and associated chroma pixels (U and V). In this thesis, the chroma components of a macroblock are assumed to each consist of 8 x 8 pixels (unless otherwise stated).
- **Block:** An M x N array of samples.

- **Picture:** In this thesis, a picture is defined as a coded video frame.

- **Slice:** A region of a coded picture, which is composed of a number of macroblocks.

- **Sequence:** A set of successive pictures representing a period of time of a video signal.

- **Motion vector (MV):** The offset between a macroblock or block and a matching area in a reference frame.

- **Motion estimation:** The process of finding optimal or near-optimal matching for a macroblock or block from one or more reference frame(s).

- **Motion compensation:** Computing the difference between a macroblock or block and a matching area in one or more reference frame(s).

- **Discrete Cosine Transform (DCT):** A transform converting a set of samples from spatial domain to frequency domain.

- **Quantization:** The process of mapping a signal with a range of values X to a signal with a reduced range of values Y.

- **Entropy coding:** The process of converting a series of symbols (e.g. transform coefficients, motion vectors, etc) into a compressed form.

- **Encoder:** Converts a series of video frames into a compressed form (coded video) prior to transmission and/or storage.

- **Decoder:** Decompresses coded video before display and/or storage.

- **CODEC:** An abbreviation of video encoder and decoder.

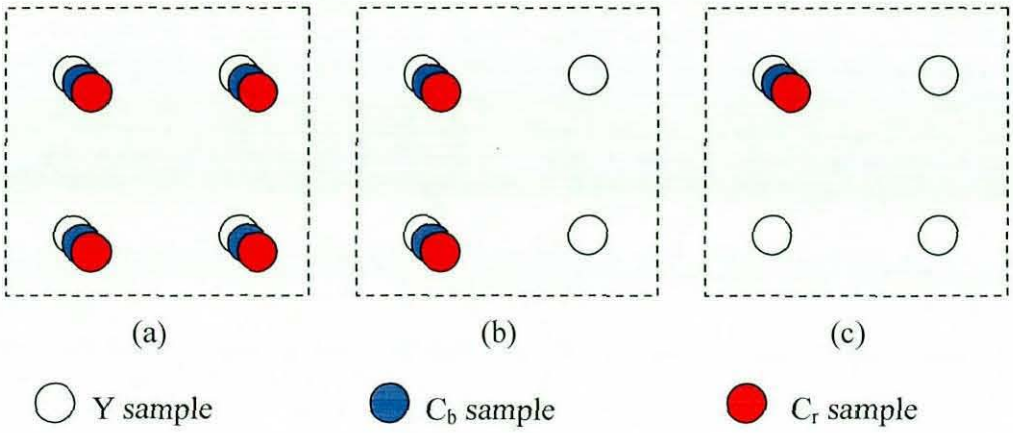


Figure 2-1 Sampling patterns (a) 4:4:4; (b) 4:2:2 and (c) 4:2:0

2.2.2 Colour Space and Sampling Formats

The method chosen to represent brightness (luminance) and colour of images and videos is described as a colour space. RGB and YC_bC_r are two commonly used colour spaces in image and video representation. In the RGB colour space, a colour image sample is represented with three colour components Red, Green and Blue. The components are equally important and thus are usually stored at the same resolution [11]. For example, a RGB colour image with resolution of 176 x 144, will require 1 byte (8 bits) per pixel, per colour. Thus the colour image requires a total of 74.25 Kbytes for storage. On the other hand, in the YC_bC_r colour space, a colour image may be represented more efficiently by reducing the resolution in colour (chroma) components, without having an obvious effect on visual quality due to the fact that the Human Visual System (HVS) is less sensitive to colour than luminance [11]. In the case of the above example, the storage requirement when using the YC_bC_r colour representation is less than 74.25 Kbytes. The actual size of the storage requirement depends upon the chroma sampling format (discussed in the next paragraph) used. For this reason, the YC_bC_r colour space is a popular way of efficiently representing colour images and videos.

Table 2-1 Common intermediate formats used in digital video

Format	Video Resolution (Hori. x Vert.)	Bits Per Frame (4:2:0, 8 bits per sample)
SQCIF	128 x 96	147456
QCIF	176 x 144	304128
CIF	352 x 288	1216512
4CIF	704 x 576	4866048
SIF	352 x 240	1013760

As mentioned above, chroma components may be represented with a lower resolution than the luma component without significantly reducing the overall visual quality. Hence, there are three sampling formats usually used in YC_bC_r colour space, 4:4:4, 4:2:2 and 4:2:0. In the 4:4:4 sampling (see Figure 2-1 (a)), the three components (Y, C_b and C_r) have the same resolution, which is similar to RGB colour space. In the 4:2:2 sampling (sometimes referred to as YUY2), the chroma components have the same vertical resolution as the luma but half the horizontal resolution as illustrated in Figure 2-1 (b). The number 4:2:2 means that for every four luma samples in the horizontal direction there are two C_b and two C_r samples. In the 4:2:0 sampling (sometimes referred to as YV12), U and V components have half resolution of Y in both horizontal and vertical directions as illustrated in Figure 2-1 (c). In other words, a 4:2:0 YC_bC_r video requires exactly half as many samples as 4:4:4 or RGB video. In the above example, the image requires only 37.13 Kbytes space. Hence, 4:2:0 sampling is widely used for consumer applications such as video conferencing, digital television, DVD storage and this thesis as well.

2.2.3 Video Format (Resolutions)

The most widely used, standard video resolutions are tabulated in Table 2-1. The Common Intermediate Format (CIF), 352 x 288 pixels, is commonly used to

standardize the horizontal and vertical resolutions of YC_bC_r colour video sequences. It was originally proposed as a part of the H.261 [8] video coding standard. The CIF is the basis for converting into many other standard resolutions as depicted in Table 2-1. Different resolutions are used in different applications catering for the needs of different screen sizes such as in mobile phones, video conferencing applications etc. The experiments conducted in this thesis use video sequences belonging to a number of different resolutions defined in the table.

2.2.4 Bitrate

In video processing, bitrate (sometimes written bit rate or simply as rate) is the number of bits that are conveyed or processed per unit of time [13]. Bit rate is quantified using the "bit per second" (bit/s or bps) unit, often in conjunction with a metric prefix such as kilo- (kbit/s or kbps) and is synonymous to data rate and digital bandwidth [13]. In this thesis, all bit rates of coded videos are calculated in terms of the equation below:

$$R = \frac{S \times FR}{N} \tag{2.1}$$

where R indicates the bit rate, S is the size of a coded video in bits. FR and N are the frame rate (fps) of the coded video and the length of the video (i.e., the number of frames in the video) respectively.

2.2.5 Mean Squared Error (MSE)

Mean Squared Error (MSE) is one of the popular objective video quality measurement methods. It calculates the average of squared difference between the original video sequence (V_o) and reconstruction video sequence (V_r) as shown by Equation (2.2). It is noted that N is the total number of pixels in a single frame.

$$MSE = \frac{\sum_{i=1}^{N} (V_o - V_r)^2}{N} \tag{2.2}$$

2.2.6 Peak Signal to Noise Ratio (PSNR)

Peak Signal to Noise Ratio (PSNR) is another popular objective video quality metric. It is measured in a logarithmic scale and depends on the Mean Squared Error (MSE) of a video frame as shown by Equation (2.3). In this thesis, the averaged PSNR of the luminance and chrominance components is used to measure video quality (8 bits per pixel).

$$PSNR = 10\log_{10}\left(255^2 / MSE\right) \tag{2.3}$$

2.3 Introduction to H.264

H.264 [1, 14], known as MPEG-4 Part 10 or Advanced Video Coding (AVC), was developed by the Joint Video Team (JVT) (consisting of IUT-T VCEG and ISO MPEG) in 2003. The main focus of the standardisation of H.264 was to enhance coding efficiency as compared to previous video coding standards and to provide flexibility for effective use over a broad variety of network types and application domains. The basic video coding design in H.264 is based on conventional block-based motion-compensated hybrid video coding concepts, but with some important differences relative to prior standards such as enhanced prediction capability, small block-size exact-match integer transform, and enhanced entropy coding methods and so on. The enhanced algorithms utilized within H.264 enable it to provide approximately a 50% extra bit rate savings (at an equivalent perceptual quality) when compared to the performance of prior standards [14].

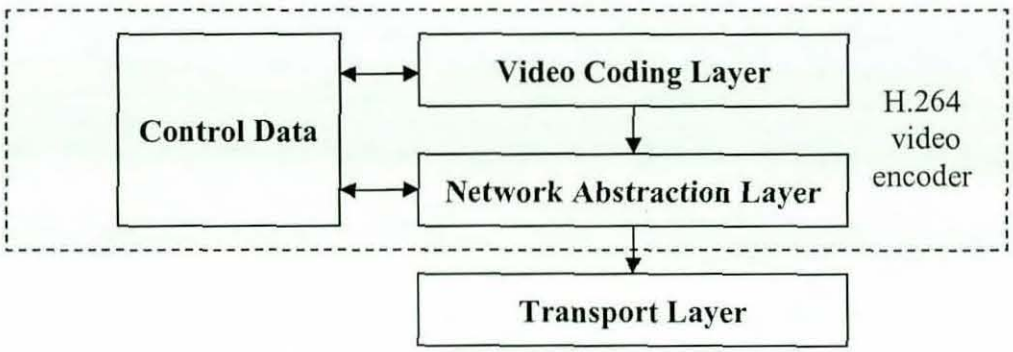


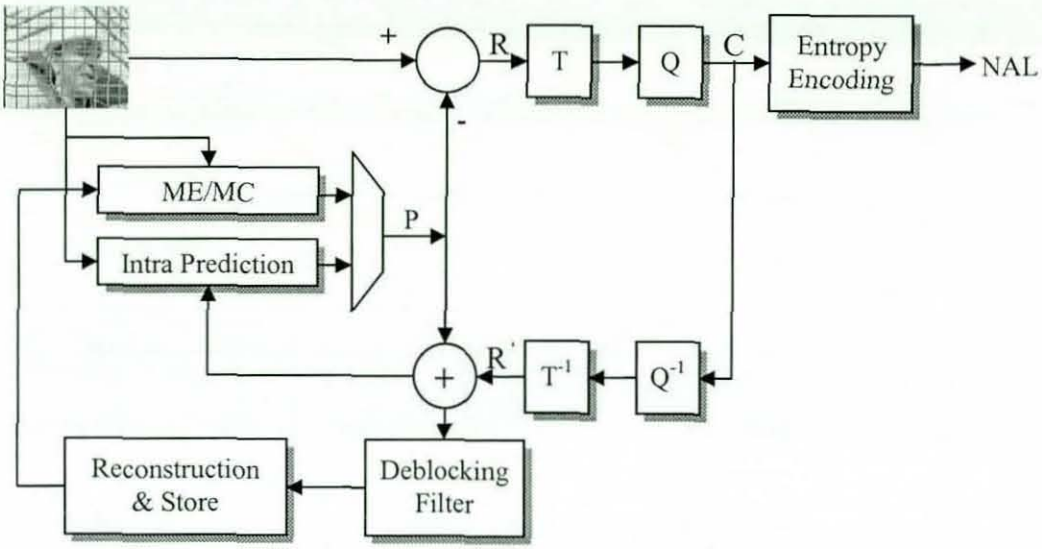
Figure 2-2 Layer structure of H.264 video encoder

2.3.1 Structure of H.264

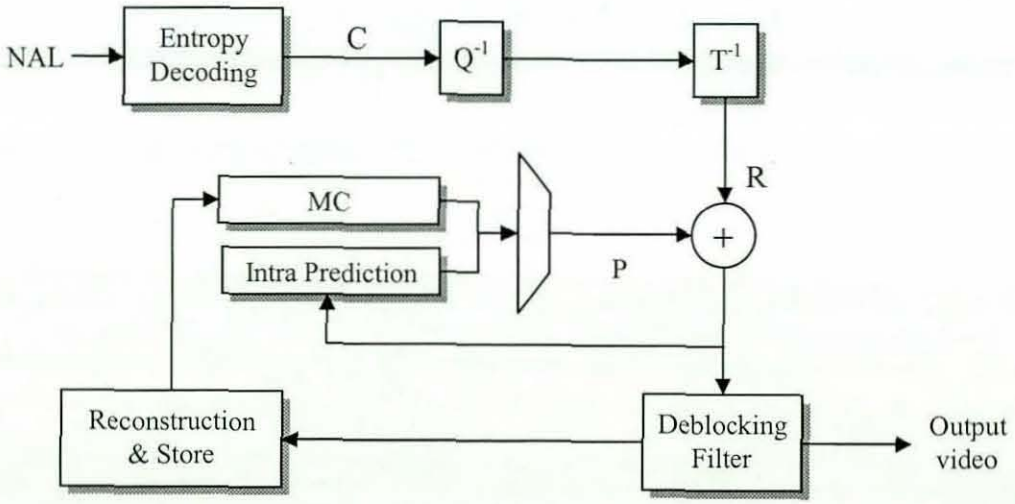
To address the need of flexibility and customizability, the H.264 design is separated into a video coding layer (VCL), which achieves a generic video compression similar in spirit to other standards such as MPEG-2 [4], and a network abstraction layer (NAL), which formats the VCL data together with some control data and provides header information for effective transmission by the transport layers or storage media (see Figure 2-2) [14].

2.3.2 H.264 CODEC

Figure 2-3 (a) and (b) show block diagrams of a H.264 encoder and decoder respectively. The H.264 CODEC structure has some similarities to those of the prior standards but some important changes have been made to improve coding performance. Enhancements will be explained in Section 2.3.4. The data flow within the CODEC is briefly discussed below.



(a) H.264 encoder



(b) H.264 decoder

Figure 2-3 Block diagram of a H.264 CODEC

The H.264 encoder comprises of two parts: encoding and reconstruction. At the encoding part, each picture of a video is initially partitioned into one or more slices. Each slice consists of a number of 16 x 16 macroblocks of the luma component and 8 x 8 blocks of each of the two chroma components. Each luma or chroma macroblock is either spatially or temporally predicted (The macroblock or block may be subdivided into sub-blocks for improved efficiency of prediction). The resulting prediction block (marked '*P*' in Figure 2-3) is subsequently subtracted from the current block to produce a residual (difference) block (marked '*R*' in Figure 2-3). The residual block is transformed using integer transform (*T*), and the transform coefficients are quantized (*Q*) to generate '*C*', a set of quantised transform coefficients which are finally encoded by entropy coding. The resulting entropy-encoded data, together with additional information required to decode each block within the macroblock (prediction modes, quantizer parameter used, motion vector information, etc.) form the compressed bitstream which is passed to the NAL for transmission or storage. At the reconstruction part, the coefficients "*C*", undergo inverse quantisation (Q^{-1}) and inverse transform (T^{-1}) to produce a residual block (*R'*). The residual block is added to the prediction block (*P*) to create a reconstructed block which is filtered to reduce the effects of blocking artefacts [11]. This reconstructed block is saved as a part of a reconstructed frame which is used as a reference frame in the prediction of macroblocks of subsequent frames.

The decoder receives the NAL data and initially uses entropy-decoding to obtain the quantized coefficient, "*C*". This data then follows a path similar to that described in the reconstruction part of the encoder, to finally obtain the reconstructed frame.

2.3.3 Profiles and Levels

In H.264, *Profiles* and *Levels* specify the conformance points. These conformance points are designed to facilitate inter-operability between various applications of the H.264 standard. A *profile* defines a set of coding tools or algorithms that can be used in producing a compliant bitstream, whereas a *level* places limits on parameters of the bitstream such as frame size, memory requirements and coded bit rate.

H.264 defines seven *Profiles* (Baseline, Main, Extended, High, High 10, High 4:2:2, and High 4:4:4). Each profile specifies a subset of algorithmic features and limits as shown in Table 2-2. The associated potential application domains of each profile are well described in [13]. For example, Baseline profile is primarily used for lower-cost applications with limited computing resources like videoconferencing and mobile applications. The details of coding tools used in Baseline profile are described in detail, from Section 2.3.5 to Section 2.3.9 since all the contributions of this thesis are based on this profile. For information on other profiles, the readers are referred to [15].

Table 2-2 Specification of H.264 Profiles

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

2.3.4 H.264 Enhancements

H.264 contains a number of new features that allow it to compress video much more effectively than previous standards. Some highlighted features relevant to Baseline profile are listed as follows (for more information, see [14]).

Enhancements to prediction methods:

- **Variable block-size motion compensation with small block sizes:** Supports flexible motion compensation block sizes (minimum size 4 x 4).

- **Quarter-sample-accurate motion compensation:** Introduces quarter-sample-accurate motion vector prediction from MPEG-4 Visual (part 2) [5], but with reduced the complexity of the interpolation processing.

- **Multiple reference picture motion compensation:** Allows using multiple reference frames (up to 16) for motion compensation to enhance coding efficiency.

- **Directional spatial prediction for intra coding:** Adds spatial prediction to intra coding to improve the quality of the prediction signal.

- **In-loop deblocking filtering:** The deblocking filter (see Section 2.3.8) is placed within the motion-compensated prediction loop so that the resulting improvement in quality of the reference frame can be used in inter-picture prediction to improve the ability to predict other frames.

Improvements to coding efficiency:

- **Small block-size integer transform:** The use of 4 x 4 integer transform (in comparison with 8 x 8 DCT in floating point arithmetic in other standards) reduces "ringing" (artefacts) distortion (faint patterns along the edges of objects, caused by the 'break through' of DCT basis patterns in a decoded image). The integer transform (see Section 2.3.7) also

reduces the computational complexity from 32-bit processing to 16-bit arithmetic and avoids inverse transform mismatch problem as well.

- **Context-adaptive entropy coding**: Both entropy coding methods used, i.e., CAVLC (context-adaptive variable-length coding) and CABAC (context-adaptive binary arithmetic coding) use context-based adaptivity to improve performance relative to pervious standards.

Enhancements to error resilience and flexibility of transmission:

- **Parameter set structure:** The parameter set design provides for robust and efficient conveyance of header information.

- **NAL unit syntax structure:** The NAL unit syntax structure provides more robustness and flexibility than that provided in prior standards.

- **Flexible macroblock ordering (FMO):** A novel ability to partition the picture into flexible regions called slice groups, each of which consists of a number of macroblocks and can be decoded independently. The ability significantly enhances the robustness to data losses in transmission.

- **Arbitrary slice ordering (ASO):** Enables sending and receiving the slices of the picture in any order relative to each other (since each slice can be decoded independently). This capability can improve end-to-end delay (e.g., out-of-order delivery) in real-time applications.

2.3.5 Intra Prediction

In intra prediction mode, each prediction block *P* (see Figure 2-3 (a)) is generated from spatially neighbouring samples of already coded blocks in the same slice. H.264 provides two classes of intra coding types, i.e. INTRA-4x4 and INTRA-16x16 for the

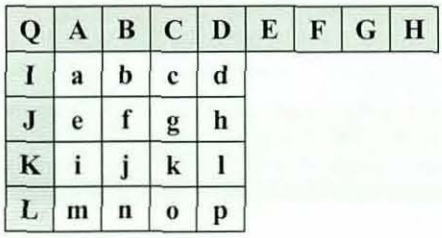


Figure 2-4 Neighbouring samples of INTRA-4x4 mode

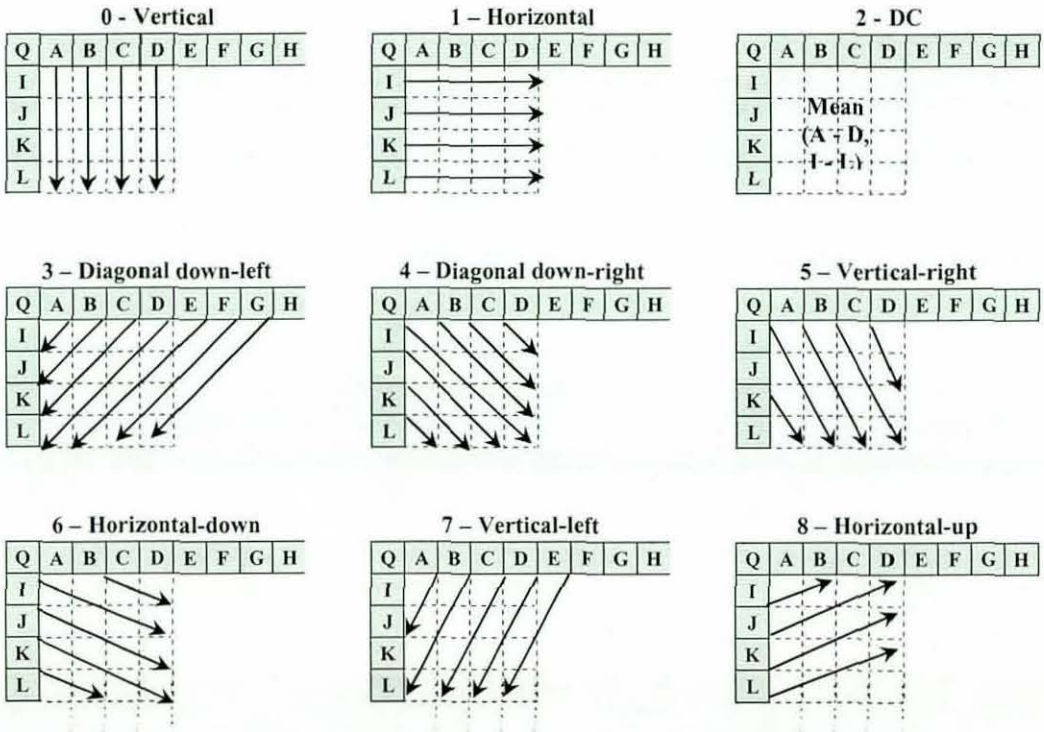


Figure 2-5 INTRA-4x4 prediction modes

luma components. Figure 2-4 [11] illustrates neighbouring samples of INTRA-4x4 mode. For INTRA-4x4 mode, a total of nine possible prediction modes (see Figure 2-5 [11]) are used for prediction. For the INTRA-16x16 mode, which is the typical prediction mode used in smooth image areas, four types of prediction modes are supported. For detailed information on the INTRA-16x16 prediction modes, the

readers are referred to [16]. Each 8 x 8 chroma block of a macroblock is predicted following a prediction technique similar to that of an INTRA-16x16 luma macroblock. It is noted that in order to maintain the independent nature of the individual slices and slice groups, intra prediction (and all other forms of prediction, such as inter prediction) across slice boundaries is restricted.

2.3.6 Inter Prediction

Inter prediction is also known as motion compensation, and is usually used to reduce the temporal redundancy in moving pictures. H.264 adopts block-based motion estimation and compensation for removing the redundancy between frames. Within this approach, each M x N block in the current frame is compared with blocks of similar size within a predefined search region of the reference pictures, in order to obtain the "best" match (the technique of searching the "best" match is known as motion estimation (ME) which is discussed in later this section). This "best" match block is then subtracted from the current block to produce a residual block R that is encoded and transmitted along with the corresponding motion vector difference (MVD) describing the residual between the current motion vector and a predicted motion vector.

In addition to the intra macroblock coding types, H.264 also supports various prediction modes for P-slices (P frames). Each luma macroblock in a P-slice may be partitioned into one of eight block shapes (as illustrated in Figure 2-6) used for motion-compensated prediction. 1/2 and 1/4 sub-samples are used in H.264 for the accuracy of motion compensation. Since these prediction values at 1/2 and 1/4 sample positions do not exist in the reference frame, they are obtained using interpolation from nearby integer pixel location samples. Each chroma block in a macroblock is segmented in a manner similar to that of the luma component. The prediction values at fractional positions for the chroma component are generated by bilinear interpolation.

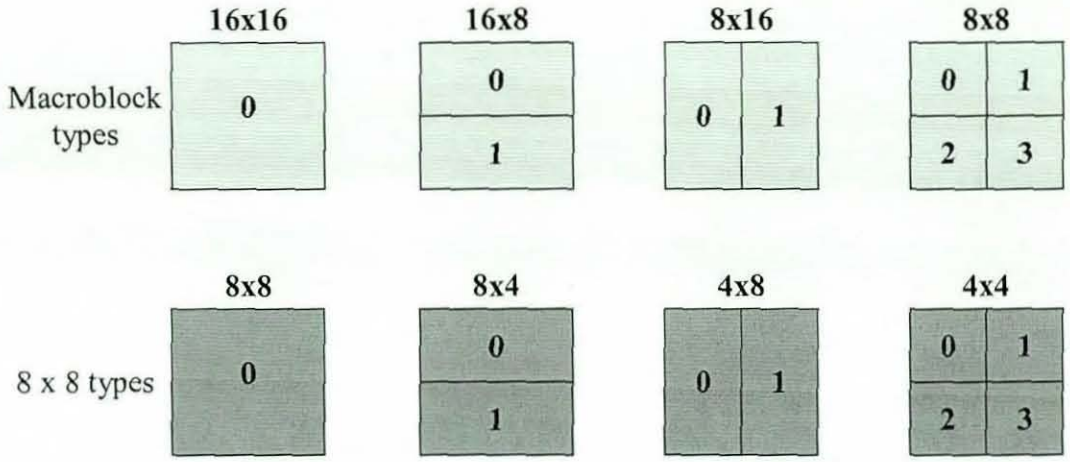


Figure 2-6 Segmentations of the macroblock for motion compensation

It is noted that motion vector of a block partition can be predicted from motion vectors of nearby previously coded partitions due to the high correlation between neighbouring partitions. The predicted motion vector, MVp, depends on the motion compensation partition size and on the availability of nearby vectors [11].

ME technique is used for seeking the "best" match block from the reference frame within a certain search range. The current H.264 reference software JM [51] provides a Full Search (FS) ME algorithm and several valuable Fast Motion Estimation (FME) algorithms [68-71]. The FME algorithms include Unsymmetrical-cross Multi-Hexagon Search (UMHS) [68-69], Simplified Unsymmetrical-cross Multi-Hexagon Search (SUMHS) [70] and Enhanced Predictive Zonal Search (EPZS) [71]. These FME algorithms were designed for reducing the ME time (by only searching specified positions within a certain search range and by adopting early termination schemes) while maintaining almost the same PSNR that FS could achieve (see [62] or Section 6.4.1 and Section 6.4.3).

2.3.7 Transform and Quantization

After intra or inter prediction, the residual block R is transformed and quantized. H.264 uses a 4 x 4 integer transform [17] for the residual block instead of the 8 x 8 DCT transform used in other standards. For INTRA-16x16 mode, an extra 4 x 4 transform is applied to the 4 x 4 DC coefficients of the luma components. For chroma blocks, an additional 2 x 2 transform is performed for the four DC coefficients.

The integer transform is based on a standard 4 x 4 DCT but includes additional features. The inverse transform mismatch problem caused by using floating point transforms is avoided since the transform is computed in integer arithmetic. Furthermore, the 4 x 4 integer transform is performed in 16-bit arithmetic by the use of simple additions and shifts (i.e., without the need of multiplications), thus minimizing computational complexity. In addition, a scaling multiplication (part of the transform) is combined into the quantizer so that the total number of multiplications can be reduced. Finally, by using quantization tables, the need for divisions at quantizer is avoided.

2.3.7.1 Forward Transform

A 4 x 4 forward DCT is given as follows [11]:

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \tag{2.4}$$

where $a = 1/2$, $b = \sqrt{1/2}\cos(\pi/8)$, $c = \sqrt{1/2}\cos(3\pi/8)$, A and A^T are the transform matrix and its transpose respectively, X is a matrix of samples and Y is the resulting coefficients.

The A and A^T can be factorised [18] to the following equivalent representation:

$$Y=\left(CXC^T\right)\otimes E_f=\left(\begin{bmatrix}1&1&1&1\\1&d&-d&-1\\1&-1&-1&1\\d&-1&1&-d\end{bmatrix}\begin{bmatrix}X\end{bmatrix}\begin{bmatrix}1&1&1&d\\1&d&-1&-1\\1&-d&-1&1\\1&-1&1&-d\end{bmatrix}\right)\otimes\begin{bmatrix}a^2&ab&a^2&ab\\ab&b^2&ab&b^2\\a^2&ab&a^2&ab\\ab&b^2&ab&b^2\end{bmatrix} \tag{2.5}$$

where $\otimes$ indicates element-by-element multiplication rather than matrix multiplication. E_f is a matrix of scaling factors, which can be integrated into the quantization process as discussing later. CXC^T is a "core" 2-D transform. The value of d (=0.414213...) is approximated by 1/2 so that the matrix multiplication can be computed with fixed-point arithmetic [18]. Thus, b is also modified accordingly in order to maintain orthogonality of A, i.e., $A^TA=I$. Therefore, finally, $a=1/2$, $b=\sqrt{2/5}$, $c=\sqrt{1/10}$ and $d=1/2$.

After modifications of b, c and d, the 2nd and 4th rows of C and the 2nd and 4th columns of C^T can be further factorised, thus Equation (2.5) becomes:

$$\begin{aligned}Y&=\left(CXC^T\right)\otimes E_f\\&=\left(\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\begin{bmatrix}X\end{bmatrix}\begin{bmatrix}1&2&1&1\\1&1&-1&-2\\1&-1&-1&2\\1&-2&1&-1\end{bmatrix}\right)\otimes\begin{bmatrix}a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\\a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\end{bmatrix}\end{aligned} \tag{2.6}$$

The "core" transform CXC^T in Equation (2.6), therefore, can be implemented by additions and shifts. Since the maximum sum of absolute values in any row of C is 6, the maximum dynamic range gain increase for a 2-D transform is $log_2(6^2)$ = 5.17. That is, the output of CXC^T after the 2-D transform will need 6 more bits than the input X (prediction residuals) which have a 9-bit range for 8-bit pixel data with a 1-bit signed symbol. Thus, the "core" transform can be computed with 16-bit arithmetic.

2.3.7.2 Inverse Transform

For the inverse transform, the equation is given by:

$$X' = C^T (Y \otimes E_i) C$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \left(\begin{bmatrix} \\ Y \\ \\ \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

(2.7)

where Y is the transformed coefficients from Equation (2.6) and E_i is a matrix of scaling factors, which is combined in quantization stage. X' is the output of the inverse transform. The factors $\pm 1/2$ are implemented as right shifts which introduce small errors, but can be compensated by a larger dynamic range for the data at the input of the inverse transform [18].

2.3.7.3 Quantization

A general representation of quantization is given by [11]:

$$Z_{ij} = round\left(Y_{ij} / Qstep\right)$$

(2.8)

where Y_{ij} is a transformed coefficient from Equation (2.6), Z_{ij} is a quantized, approximated (to the nearest integer) coefficient. *Qstep* indicates a quantizer step which is indexed by a quantization parameter (QP). H.264 provides a large range of *Qstep* from 0 to 51, which enables flexibility in bit rate and quality control. It is noted that *Qstep* doubles in size for every increment of 6 in QP.

Each integer coefficient of $W = CXC^T$ and the corresponding scaling factor in E_f are input to the quantizer, so Equation (2.8) becomes [11]:

$$Z_{ij} = round\left(W_{ij} \cdot \frac{PF}{Qstep} \right)$$

(2.9)

where PF is a^2, $ab/2$ and $b^2/4$ depending on the position (i, j). The factor $(PF/Qstep)$ is then changed into the following form [11]:

$$PF/Qstep = MF/2^{qbits}, \quad qbits = 15 + floor(QP/6).$$

MF is a multiplication factor. Thus, Equation (2.9) can be implemented as follows:

$$\begin{aligned} |Z_{ij}| &= (|W_{ij}| \cdot MF + f) \gg qbits \\ sign(Z_{ij}) &= sign(W_{ij}) \end{aligned} \tag{2.10}$$

where f is a dead-zone control parameter, which is set to $2^{qbits}/3$ for intra and $2^{qbits}/6$ for inter frames. The symbol $\gg$ indicates a right shift.

2.3.7.4 De-quantization

The basic de-quantization formula is:

$$Y'_{ij} = Z_{ij} Qstep \tag{2.11}$$

The scaling factors of E_i are multiplied by a constant scaling factor 64 (avoiding rounding errors), and then are integrated into the above equation:

$$W'_{ij} = Z_{ij} V_{ij} \cdot 2^{floor(QP/6)} \tag{2.12}$$

where W_{ij} is a de-quantized coefficient and $V = Qstep \cdot PF \cdot 64$. The factor produces the output increase by a factor of two for every increment of six in QP. H.264 defines MF and V as quantization and de-quantization tables respectively. Only the first six values ($0 \le QP \le 5$) are defined. The final output of the inverse transform are divided by 64 to remove the scaling factor (implemented by a right shift) introduced in Equation (2.12).

2.3.8 Deblocking Filter

The 'blocking' artifacts, i.e., visible block edges in a decoded picture, are common problems that appear in block-based video coding. H.264 adopts an adaptive deblocking filter within the motion-compensated prediction loop to improve video quality. Moreover, this improvement in quality positively contributes towards inter prediction as it reduces the residual errors.

2.3.9 Entropy Coding

Two methods of entropy coding: CAVLC and CABAC are supported in H.264. CAVLC is simpler compared with CABAC which is more efficient in coding. Readers interested in further details of these techniques are referred to the H.264 standardization documents [1].

2.4 Video Object-based Coding

In addition to the conventional frame-based video coding approaches, object-based coding (known as region-based coding, first proposed by Musmann *et al.* [19]) has been considered in second generation coding techniques/standards such as MPEG-4 Visual. Object-based coding separates a video frame into component objects, which are subsequently coded as arbitrary shaped objects. It is noted that after segmenting a frame into objects, the shape, texture and motion information are coded and transmitted independently.

2.5 Arbitrary Shaped Video Object Coding in MPEG-4

The concept of a video object (VO) was introduced in early versions of video coding standards such as MPEG-2 [4] and H.263 [10]. However, in these standards the definition of video objects was restricted to rectangular shaped areas within a spatial frame that was time invariant with respect to size and position. The latter version, MPEG-4 (i.e., Part 2: Visual) [5] was the first video coding standard that treats a video sequence as a collection of one or more two-dimensional (2-D) video objects

and describes a video object to be of arbitrary shape. Furthermore, the shape, size, and position of the video object may vary from one frame to the next [20].

2.5.1 Video Object Planes

A video object (VO) is an area of a video scene that may be an arbitrarily shaped region such as a person, ball or tree and may vary with time. A snapshot of a video object taken at a given sample time is regarded as a video object plane (VOP). A VOP is essentially a rectangular area that completely contains the video object but with the minimum number of macroblocks contained within it as shown in Figure 2-7 (a). Each VOP defines the video object's texture (luminance and chrominance samples) and shape information. To achieve the coding of arbitrarily shaped objects, MPEG-4 uses two extra tools, i.e. MPEG-4 encodes the shape information first using a binary shape encoder (Section 2.5.2) and subsequently uses two different discrete cosine transform (DCT)-based algorithms: DCT and shape-adaptive DCT (Section 2.5.3) for the interior texture coding and boundary texture coding respectively.

2.5.2 Binary Shape Coding

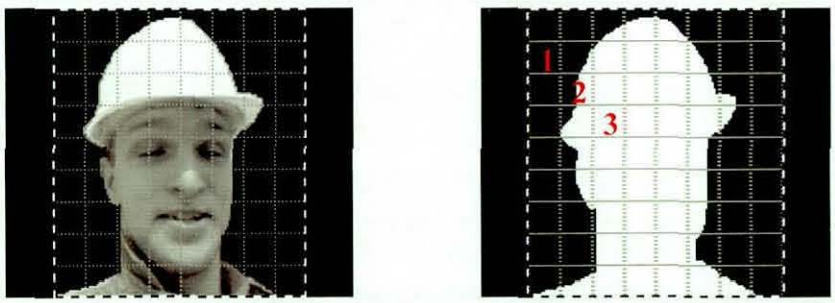
In MPEG-4, the contour information of an arbitrarily shaped object is described by a map of the same dimension as the luminance signal. There are two types of maps supported by MPEG-4 in terms of level of transparency: binary alpha map and grey-scale alpha map. The binary alpha map defines all pixels either as opaque (1, inside a video object) or as transparent (0, outside a video object). This map is subsequently coded using binary shape coding. Alternatively, the grey-scale alpha map employs 8 bits to represent the transparency level (between 0, transparent, and 255, opaque) of each pixel. The compression of the map is achieved using grey-level alpha coding which is not discussed in this thesis (For further details see the original MPEG-4 [5]).

In a binary alpha map, the binary object is enclosed in a tightest bounded VOP that is made up of a number of 16 x 16 blocks as shown in Figure 2-7 (b). Each such binary alpha block is referred to as a BAB that is encoded by the binary shape encoder (or BAB encoder). Figure 2-8 illustrates a simplified block diagram of the BAB encoder.

All BABs are classed into three categories: transparent, boundary and opaque. These categories are marked with "1", "2" and "3" in Figure 2-7 (b).

2.5.2.1 BAB Coding

A BAB is first input to the BAB encoder and may be encoded using one of several ways listed in Table 2-3. It is the task of the mode decision block to select an efficient encoding method for the BAB. Mode "2" and "3" are the simplest and most efficient methods to compress a BAB as it is a transparent or opaque BAB. Mode "0"



(a) (b)

Figure 2-7 The Foreman object. (a) Texture object. (b) Binary alpha object.

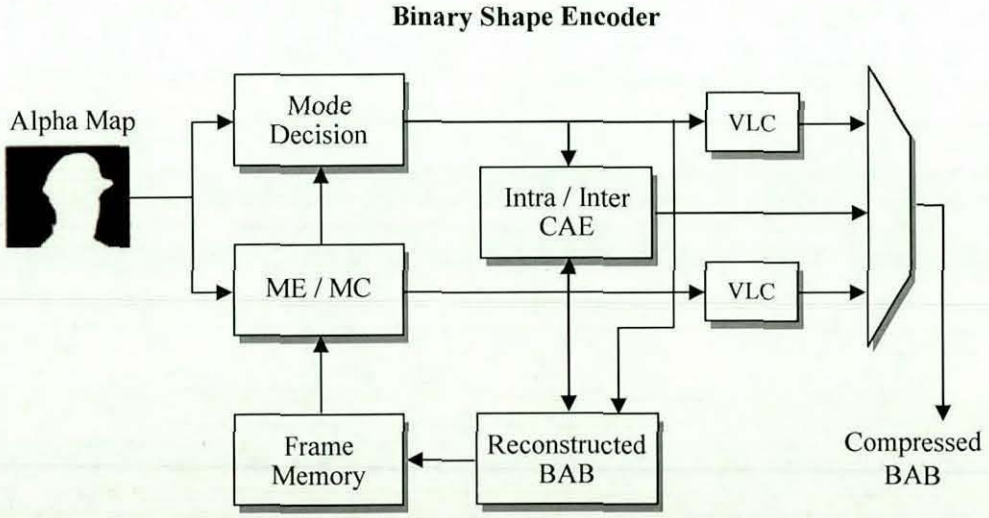


Figure 2-8 Block diagram of MPEG-4 binary shape encoder

or "1" is a "no update" mode which means the BAB is reconstructed by motion compensation alone. All other BABs (boundaries) are coded by intra- or inter-context-based arithmetic encoding (CAE). Finally, the coded BAB, coding mode and motion vector (for inter coded blocks) are either stored or transmitted to the decoder.

2.5.2.2 Context-based Arithmetic Encoding (CAE)

For a given BAB, CAE encoding is performed pixel by pixel. Each pixel is coded using a template that consists of a number of previously coded pixels. The intra template is formed by ten surrounding pixels (see Figure 2-9 (a), *X* is the current pixel to be coded) of the current BAB. The inter template is formed by 9 pixels as illustrated in Figure 2-9 (b) and (c), *c0-c3* are obtained from the current BAB and spatially related to the pixel *X*, and *c4-c7* coming from the motion compensated BAB with *c6* fully aligned with *X* [20]. From the template, a context number is computed [21] and is subsequently used to access a probability (the probability that the pixel is zero) table provided with the MPEG-4 specification [22]. The accessed probability and the pixel value (0 or 1) are then used to drive an arithmetic encoder [23].

Table 2-3 BAB coding modes represented by the BAB type

BAB type	Type	Used in	Description
0	No update, no MVD	P-,B-VOPs	The decoded block is obtained through motion compensation without correction. No motion vector difference is given, i.e. it is set to zero.
1	No update, with MVD	P-,B-VOPs	The decoded block is obtained through motion compensation without correction. A motion vector difference is given.
2	Transparent	I-, P-, B-VOPs	The decoded block is completely transparent.
3	Opaque	I-, P-, B-VOPs	The decoded block is completely opaque.
4	Intra CAE	I-, P-, B-VOPs	The decoded block is obtained through intra CAE decoding.
5	Inter CAE, no MVD	P-, B-VOPs	The decoded block is obtained through inter CAE decoding. No motion vector difference is given, i.e. it is set to zero.
6	Inter CAE, with MVD	P-, B-VOPs	The decoded block is obtained through inter CAE decoding. A motion vector difference is given.

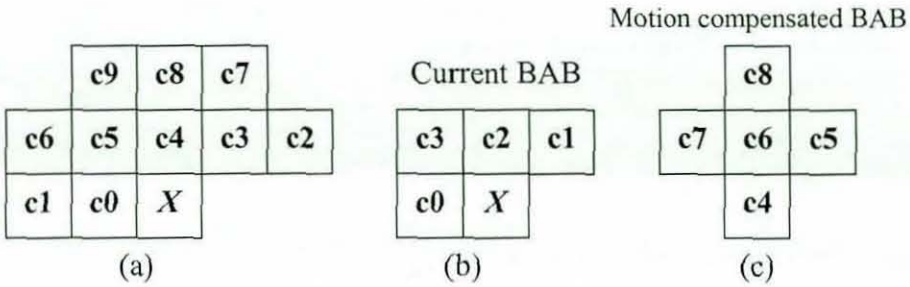


Figure 2-9 The templates. (a) Intra template; (b) and (c) Inter template

2.5.2.3 Mode Coding

Once the coding mode (i.e., BAB type) for the current BAB has been decided, the coding mode is further coded by using VLC tables provided by MPEG-4 [22]. For I-VOP, the VLC table for the coding mode of the current BAB is selected by an index computed by:

$$27\text{x}A_{mode} + 9\text{x}B_{mode} + 3\text{x}C_{mode} + D_{mode}$$

where A_{mode}, B_{mode}, C_{mode} and D_{mode} relate to the coding modes of those BAB's adjacent to the current BAB as illustrated in Figure 2-10. For P-, and B-VOP's, the chosen VLC table relies on the coding mode of the BAB that has the same location as the current BAB in the reference VOP [20].

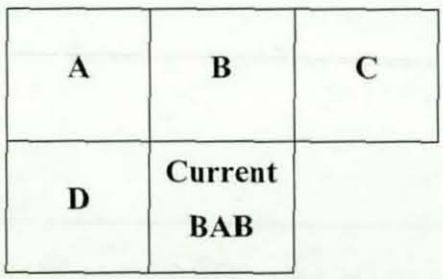


Figure 2-10 The adjacent BAB's of the current BAB in I-VOP

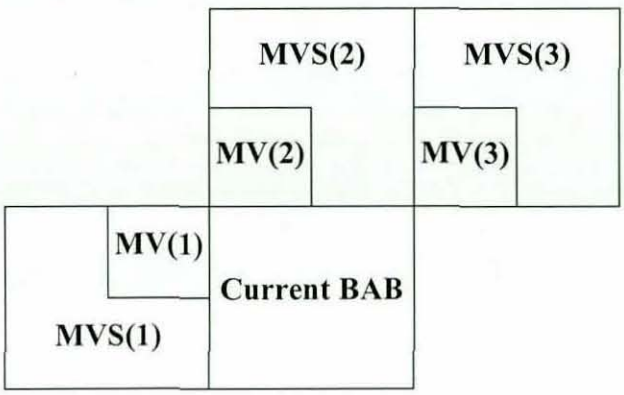


Figure 2-11 The current BAB and related candidate motion vectors.

2.5.2.4 Motion Vector Coding

Motion vector for shape (MVS) information is used especially for inter coded BAB's in P- and B-VOP's, which is computed from:

$$MVS\_x = MVPS\_x + MVDS\_x$$
$$MVS\_y = MVPS\_y + MVDS\_y$$

where *MVPS* is the motion vector predictor for shape and *MVDS* is the motion vector difference for shape. The *MVDS* is finally coded by a motion vector difference VLC table and is transmitted to the decoder.

The motion vector predictor, *MVPS* is selected from a list of candidate motion vectors (see Figure 2-11) that includes three shape motion vectors *MVS(i)* and three texture motion vectors *MV(i)* from the three nearby BAB's and texture blocks of the current BAB. The *MVPS* is set to the first valid motion vector encountered when following the candidate motion vectors in a predefined order {*MVS(1)*, *MVS(2)*, *MVS(3)*, *MV(1)*, *MV(2)*, *MV(3)*}.

2.5.3 Shape-Adaptive DCT

The Shape-Adaptive DCT (SA-DCT) was first proposed by Sikora [24] in 1995. Subsequently Kaup [25] solved the inherent problem of non-normalized (but orthogonal) basis functions. SA-DCT is used for dealing with an arbitrary region of a block efficiently, which is based on pre-defined sets of one-dimensional DCT basis functions. The SA-DCT is applicable to 8 x 8 blocks within a boundary BAB. Figure 2-12 gives an example of the application of the forward SA-DCT algorithm on an 8 x 8 image block that fully encloses an arbitrarily shaped object (shaded grey). Firstly, the length $N(j)$ ($1 \le N(j) \le 8$) of every column j ($1 \le j \le 8$) of the opaque pixels are calculated. The opaque pixels (grey) of each column in the initial block (Figure 2-12 (a)) are then shifted up to the upper border of the block (Figure 2-12 (b)). For a column of opaque pixels of length $N(j)$, the associated DCT transform matrix $A_{N(j)}$ is given by [24]:

$$A_{N(j)}(p,k) = c_0 \cdot \cos\left[\frac{(2k+1)\cdot p \cdot \pi}{2N(j)}\right], \quad p,k = 0,1,\ldots,N(j)-1 \tag{2.13}$$

Here $c_0 = \sqrt{1/2}$ if $p = 0$, and $c_0 = 1$ otherwise.

Y_j, the vertical (1-D column) DCT-coefficients of column j resulting from the opaque pixels, X_j, can thus be obtained by using the following formula:

$$Y_j = \sqrt{\frac{2}{N(j)}} \cdot A_{N(j)} \cdot X_j \quad , \quad 1 \le j \le 8, \quad 1 \le N(j) \le 8 \tag{2.14}$$

After applying a 1-D column DCT, the DC coefficients (denoted by ■ mark in Figure 2-12 (c)) for each column are located along the upper edge of the 8x8 block. Next, the rows are horizontally shifted to the left border of the 8x8 block (Figure 2-12 (d)) and a horizontal 1-D DCT is performed for each row of intermediate coefficients Y_i

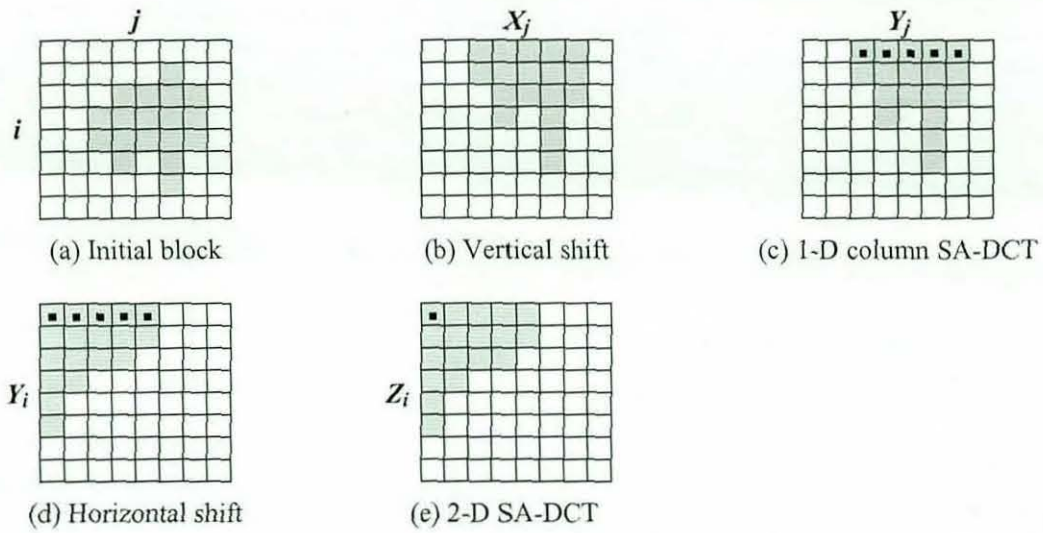


Figure 2-12 Example of forward SA-DCT processing

by using Equation (2.14). Finally, the resulting 2-D SA-DCT coefficients Z_i of the block are illustrated in Figure 2-12 (e). Note that the final DC coefficient (denoted by ■) for the whole boundary block is located in the upper left border of the block. The remaining coefficients are concentrated around the DC coefficient depending on the actual shape of the arbitrarily shaped object.

The inverse SA-DCT can be obtained with the use of the following equation:

$$X_i^* = \sqrt{\frac{2}{N(i)}} \cdot A_{N(i)}^T \cdot Z_i \tag{2.15}$$

in both horizontal and vertical directions. Here Z_i denotes forward transformed coefficients, X_i^* denotes the inverse-transformed data and i refers to the length of the data set.

2.6 Potential Applications of Object-based Coding

MPEG-4's object-based coding can be used for a wide range of interactive and content-related applications, such as high bandwidth constrained mobile telephony and surveillance systems where video quality is of utmost importance for post incident analysis. In mobile telephony, the limited size of screen implies the acceptability of encoding the background of a video frame at a lower quality as compared to the foreground, for e.g. representing speaker in a conversational application scenario, under bandwidth constraints. In surveillance applications, provided a pre-processing stage can separate important objects/regions (for the purpose of post incidence analysis), these areas can be coded at a much better quality level as compared to the background.

2.7 Summary and Conclusions

In this chapter, the fundamental concepts for video coding were initially introduced. The baseline profile of the latest video coding standard, H.264 was then discussed in detail, giving particular emphasis to the presence of enhanced features and novel coding tools, when compared to those present in traditional video coding standards. The applications that can benefit from including object-based coding functionality in H.264 were introduced. The video object-based coding concepts used in MPEG-4 were presented as an insight to the basics of the proposed H.264 video object-based coding strategy presented in Chapter 4 and Chapter 5. It is concluded that inclusion of object-based coding in H.264 requires specific design of the object-based coding concepts, tailored to the requirements of the standard.

Chapter 3 Literature Review

3.1 Introduction

In Chapter 2, the popular video coding standard, H.264 was introduced. Further the basic concepts of object-based video coding were presented in relation to MPEG-4. As mentioned in Chapter 1, the motivation of this thesis is to introduce the functionality of coding arbitrarily shaped video objects within the H.264 standard (Chapter 4 and Chapter 5) and to present a framework for optimizing a typical H.264 CODEC's performance based on multiple objectives (Chapter 6 and Chapter 7). Therefore, this chapter critically reviews the existing state-of-the-art in coding the texture of arbitrarily shaped video objects. Further the chapter provides a comprehensive review of the state-of-the-art in optimization approaches used within video coding techniques/standards.

For clarity of presentation the chapter is organized as follows. Further to the brief introduction to the chapter presented in this section, Section 3.2 reviews the coding methods used within the MPEG-4 standard for arbitrarily shaped video objects. Section 3.3 discusses a number of performance optimization methods that have been proposed for the benefit of video CODECs. Section 3.4 finally concludes with a summary and a conclusion.

3.2 Arbitrarily Shaped Video Object Coding Techniques

Texture coding within the fundamental coding units, i.e. blocks, that lie along the boundary of an arbitrarily shaped object provides the fundamental challenge in arbitrarily shaped object coding. In the last two decades, a number of arbitrarily shaped video object coding techniques have been proposed in literature. These approaches can be basically classified into two categories, extrapolation methods and

shape-adaptive transforms. Several prominent techniques are reviewed in the following sections.

3.2.1 Extrapolation Methods

The extrapolation methods provide means for extrapolating the object (opaque) pixels to completely fill a block's non-object area (i.e., where the pixels are transparent) to provide a so-called padded block that can be coded using a conventional block-based DCT. A sample border block is illustrated in Figure 3-1 (a), opaque and transparent pixels are marked as grey and white respectively.

3.2.1.1 Zero Padding

Zero padding is the simplest padding method used in which the transparent pixel values are assigned values of zeros as illustrated in Figure 3-1 (b). After padding, the block can use a normal 2-D DCT transform to convert its pixel domain representation into a frequency domain representation. This approach is recommended in MPEG-4 to be used for inter frame coding of arbitrarily shaped objects. It is obvious that the zero padding method has very low complexity and gives reasonable rate-distortion results if prediction error (residual signal) is small. However in the presence of large prediction errors, this approach usually results in discontinuities at the boundary between opaque and padded pixels. Applying the DCT to these discontinuities leads to a considerable number of nonzero high-frequency coefficients that negatively impact the coding efficiency.

3.2.1.2 Repetitive Padding

Repetitive padding is used in MPEG-4 [22] for motion compensation, in which transparent pixels in each border block are padded horizontally and vertically from opaque pixels as illustrated in Figure 3-1 (c). First of all, transparent samples of each row are extrapolated horizontally from the nearest opaque sample of the same row, both towards left and/or right directions. If the transparent samples in a particular row are enclosed on both sides by opaque pixels, the average value of the two nearest

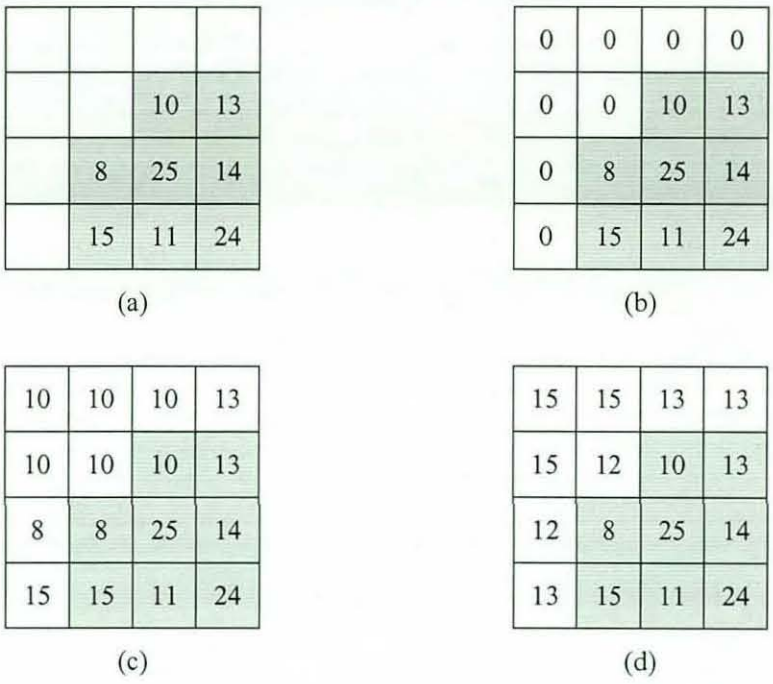


Figure 3-1 (a) Initial border block; (b) zero padding; (c) repetitive padding; (d) LPE padding.

opaque samples on both sides is extrapolated to the transparent samples. Subsequently the remaining unfilled transparent pixels, such as the first row of pixels of the block illustrated in Figure 3-1 (c) are padded, using a padding process similar to horizontal repetitive padding but in the vertical direction. This approach is very simple and its use in the coding of boundary blocks of arbitrarily shaped video objects can decrease the discontinuity related problems highlighted above (see Section 3.2.1.1). However, this relative advantage could significantly diminish as the prediction error becomes small.

3.2.1.3 Low-Pass Extrapolation

Low-pass extrapolation (LPE) was proposed by [26] and has been selected for inclusion within the MPEG-4 reference software [27]. The technique can be summarized as follows. At the start, the transparent samples of a border are replaced by the mean value of the opaque pixels on the block. Then each padded sample is

repeatedly filled by the mean value of the surrounding four samples. If one or more of the four samples are outside the block, the corresponding samples are not considered for the averaging operation. Figure 3-1 (d) shows the result of applying low-pass extrapolation to a 4 x 4 border block. The low-pass extrapolation method has been proven to provide an improved performance particularly in the intra-frame coding mode, as compared with zero padding and repetitive padding. However, its benefit in inter-frame coding, is marginal. Hence its application within the MPEG-4 video coding standard is limited to intra-frame coding. Moreover, the computational complexity of LPE is slightly higher than that of zero and repetitive padding methods.

3.2.1.4 Extension Interpolation

Extension interpolation (EI) was originally proposed by [28] and uses an interpolation approach. The idea is as follows. An M-point 1-D DCT is first executed for each column or row (of length M) in a boundary block to get M transform coefficients. Subsequently, $N - M$ zeros are filled in the rear of the DCT coefficient vector (N is the length of block size). Finally, an N-point inverse DCT is performed on the new transformed coefficient vector. In fact, these three steps can be implemented together by a multiplication matrix of dimension N x M in the pixel domain as derived in [28]. The EI presents a good rate-distortion performance which is very close to that of the smart padding (discussed in Section 3.2.1.5) and SA-DCT. Nevertheless, the EI provides the highest complexity among these approaches. In addition to this, EI involves both spatial- and frequency-domain operations for signal extrapolation, which is not very desirable from a computational point of view.

3.2.1.5 Smart Padding

A further padding method, named *smart padding*, was introduced by [29]. For a given N x N boundary block, the padding (the padded values are dependant on DCT transform matrix and opaque pixels, details on the padding approach in [29]) along the vertical (or horizontal) direction is first performed. A 1-D DCT is subsequently applied to each column of pixels. The second directional padding, i.e. in the horizontal direction is carried out next. Finally, 1-D DCT is performed on rows. The

smart padding method is comparable with EI and SA-DCT in rate-distortion performance. It is marginally better in terms of computational complexity as compared to SA-DCT approach that will be discussed in the next section. The smart-padding technique has two drawbacks, i.e. it can only be implemented in coding intra frames, and uses a joint spatial- and frequency-domain that increases its computational cost.

3.2.2 Shape-Adaptive Transforms

Two shape adaptive transform approaches [24, 30] have been proposed in the recent past for coding the texture of arbitrarily shaped video objects. They are the shape-adaptive DCT [24] (SA-DCT) discussed in detail in this thesis and the shape adaptive discrete wavelet transform (SA-DWT) [30] which is not discussed in this thesis due to its non-suitability for use in the block-based coding approach adopted by H.264. SA-DCT has been introduced in detail in Section 2.5.3. A previous attempt [31] of experimental comparison of SA-DCT versus extrapolation methods for the coding of arbitrarily shaped object has proved that the shape-adaptive method outperforms extrapolation method in terms of rate-distortion characteristics, especially for high-quality video coding. However, SA-DCT is more complex compared to simple extrapolation methods such as zero padding, repetitive padding and LPE, and its complexity is similar to that of the smart approach.

3.2.3 Summary

The state-of-the-art techniques for coding texture of arbitrarily shaped video object have been introduced in Section 3.2.1 and 3.2.2. As a result of this review, it can be concluded that each algorithm has its inherent advantages and disadvantages. Table 3-1 provides a summary of comparisons of these algorithms in terms of their rate-distortion performance and computational complexity. The rate-distortion performance is divided into five levels: *Low*, *Low-Mid.*, *Mid.*, *Mid.-High* and *High*, whereas the computational complexity is divided into three levels: *Low*, *Mid.* and *High*. The results illustrate that the three extrapolation approaches, i.e., zero padding, repetitive padding and LPE, provide lower complexity than the others. Moreover,

rate-distortion performance is the same level as other approaches in inter-frame coding mode at low bit rates. On the other hand, they offer a rather poor rate-distortion performance at high-quality video coding and low-quality video coding at intra-frame mode. EI yields good rate-distortion results at low bitrates but with high cost of complexity. Smart method seems to be very good, however, only the intra-frame mode is implemented so far. Among these algorithms in the table, SA-DCT offers the best results in terms of computational complexity versus rate-distortion performance.

As a result of the above analysis, it was concluded that SA-DCT is the most suitable approach for texture coding of arbitrarily shaped video objects. Unfortunately, further investigations revealed that the possible use of SA-DCT within the extended H.264 CODEC was critically limited due to the need of maintaining integer arithmetic operations. In addition to this reason, in the original design of SA-DCT, the non-integer scaling factors (see Section 4.3) were not integrated within the quantization process. Hence in Chapter 4, a novel shape-adaptive integer transform (SA-IT) is proposed, which conforms to a standard H.264 CODEC's integer mathematics requirements.

Table 3-1 Comparison of methods of coding arbitrarily shaped object

Method	**Rate-distortion**				**Complexity**
	Low Bitrate		**High Bitrate**		
	Intra-mode	**Inter-mode**	**Intra-mode**	**Inter-mode**	
Zero	Low	High	Low	Mid. - High	Low
Repetitive	Low - Mid.	High	Low	Low - Mid.	Low
LPE	Mid. - High	High	Mid. - High	Mid. - High	Low
EI	High	High	Mid. - High	Mid. - High	High
Smart	High	-	High	-	Mid.
SA-DCT	High	High	High	High	Mid.

3.3 Optimization methods for Video Coding

In the last decade, optimization of video compression algorithms has received significant research interest. Many optimization methods have been proposed in the literature, which can be broadly classified into two categories, algorithm-based optimizations and parameter-based optimizations. The algorithm-based optimization methods focus on the direct performance optimization of a given algorithm. Alternatively, parameter-based optimization methods optimize given objectives through the optimal selection of coding parameters.

3.3.1 Algorithm-based Optimization

A significant amount of research has been carried out on the optimization of video coding algorithms and associated sub-functions [32-47]. They can be broadly classified into single-, two- and three-objective optimization problems.

Single-objective Optimization:

In [45], Kannangara *et al.* proposed a complexity reduction algorithm for a H.264 encoder. The algorithm was developed based on the R-D optimization modes of a H.264 encoder. It reduces computational complexity through a process that identifies whether the coding of a macroblock should be skipped prior to dealing with the macroblock data. Ji *et al.* [32] proposed a memory optimization technique for a H.264 video decoder, which effectively uses the preloading mechanism for sub-macroblocks (which can be loaded early on frame reconstruction), and algorithmic improvements in the motion compensation module and the variable length decoding module, to reduce memory accesses.

Two-objective Optimization:

The most popular optimization procedure adopted in video coding is rate-distortion (R-D) optimization, which can be classified as a two-objective optimization problem. It evaluates the cost of using every possible coding mode and the corresponding motion vectors (for inter-coded frame), in obtaining the best tradeoff between the

distortion and the bit-rate, i.e. the number of bits consumed. A significant portion of optimization research in video coding has concentrated on R-D optimization procedures [33-40]. In 2001, an encoding framework for MPEG-2 and H.263 was developed by Ismaeil *et al.* in [48], which optimizes multiple coding modules (ME, DCT, quantization and mode selection) to yield desirable joint complexity-distortion (C-D) optimization. A joint optimization of power consumption (for mobile devices) and video quality was carried out by Pu *et al.* [46]. This power-distortion (P-D) model was designed for a H.264 encoder. For a given distortion constraint, the procedure minimizes the overall power consumption of the encoder.

Three-objective Optimization:

Further a number of methods have been proposed in literature to extend the traditional R-D optimization methodology by including another dimension, which is either computational complexity or power consumption. In 2003, Zhang *et al.* [42] proposed a fast motion estimation algorithm, based on an adaptive hexagon-based search (AHBS) pattern to achieve a joint optimization in complexity-rate-distortion (C-R-D) for H.264 video coding. In addition, Jesper *et al.* [43] in 2004 proposed C-R-D optimization method by using a modified EPZS (enhanced predictive zonal search) motion estimation algorithm in H.264. The modified EPZS includes three early-stop criteria, which determines if motion estimation should be stopped or not, after 16 x 16 and 8 x 8 block level sub-divisions. Moreover, Yu *et al.* [44] proposed a similar algorithm in which an alternative procedure was used to determine whether motion search into smaller block sizes should be continued or not. The similarity of the above three optimization methods is the skipping of unnecessary search modes (which thus result in a decrease in computational complexity) after obtaining the best motion vector through the C-R-D cost function. A further three-objective optimization framework, that considers joint power-rate-distortion (P-R-D), was developed by Chen *et al.* [47]. The framework was designed for H.263 video encoding on mobile devices to optimize the rate-distortion under the constraints of the power consumption.

3.3.2 Parameter-based Optimization

Kwon *et al.* [41] proposed a parameter-based method for the joint optimization of computational complexity and distortion (C-D) in H.263 video coding. Initially, three control (coding) parameters, i.e., search window size, full- or sub-pixel ME and one of four different DCT coefficient pruning options (i.e. 2 x 2, 4 x 4, 6 x 6 and full 8 x 8 DCT), are selected for a comprehensive C-D analysis of the encoder. The complexity data is subsequently obtained by calculating the operations required for the ME and DCT modules, with different combinations of the three parameter values. The distortion is estimated by averaging the PSNR over five test sequences. Finally, based on the computational complexity and distortion (averaged PSNR) data gathered, a Lagrangian method is used to find out the Lagrangian multiplier (using the corresponding combination of control parameters) that yields the optimal C-D, under a given computational complexity constraint.

3.3.3 Summary

A comprehensive literature review on optimization of video coding has been presented in Section 3.3.1 and Section 3.3.2. Table 3-2 provides a comparison of these methods giving particular emphasis to the summarization of objectives/parameters considered. It should be noted that for simplicity of presentation, the optimization objective related to power consumption [46, 47] has been represented by the computational complexity objective for all algorithms that consider power consumption. This is deemed to be reasonable since the power consumption can be shown to be directly proportional to computational complexity [49]. It is noted that most optimization research works have focused on algorithmic enhancements/improvements as compared to only one study that has focused on parameter-based optimization. In addition, the table reflects that most studies have focused on R-D optimization, whereas only a comparatively smaller number of studies have focused on joint C-D and/or C-R-D optimization.

Table 3-2 Comparison of existing optimization methods on video coding

Published year	Authors	Method	C	M	R	D
1996	Wiegand *et al.*	Algorithm-based			√	√
1997	Yang *et al.*	Algorithm-based			√	√
1998	Sullivan *et al.*	Algorithm-based			√	√
2001	Ismaeil *et al.*	Algorithm-based	√			√
2001	He *et al.*	Algorithm-based			√	√
2001	Wiegand *et al.*	Algorithm-based			√	√
2002	Stockhammer *et al.*	Algorithm-based			√	√
2003	Zhang *et al.*	Algorithm-based	√		√	√
2003	Takagi *et al.*	Algorithm-based			√	√
2003	Kwon *et al.*	Parameter-based	√			√
2004	Jesper *et al.*	Algorithm-based	√		√	√
2005	Ma *et al.*	Algorithm-based			√	√
2005	Kannangara *et al.*	Algorithm-based	√			
2005	Chen *et al.*	Algorithm-based	√		√	√
2006	Ji *et al.*	Algorithm-based		√		
2006	Yu *et al.*	Algorithm-based	√		√	√
2006	Pu *et al.*	Algorithm-based	√			√

Note: C – computational complexity, M – Memory utilization, R – bit rate, and D – distortion.

A number of optimization studies [32, 39, 40, 42-46] have also focused attention particularly on H.264. Though these methods have been well developed and have provided valuable insights into furthering the state-of-the-art, they have mainly focused on proposing algorithmic improvements/enhancements to enable optimum performance of the encoder or decoder of a H.264 CODEC. Given a H.264 CODEC, either complying with the standard or modified/enhanced version (i.e. in particular the encoder), a large number of coding options are available through the selection of various combinations of a large number of coding parameters. Therefore an obvious problem that needs solving is, "given a video sequence, what combination of coding parameters should be used so as to achieve the optimum performance of the CODEC". The performance could be largely compromised due to the selection of ill-suited parameter values. Thus, the choice of the right parameter set is of utmost

importance. The parameter-based optimization of a H.264 video CODEC can provide a solution for this problem. Although a parameter based optimization approach for H.263 video has been proposed in the literature, the method only focuses on the joint optimization of complexity and distortion. Other important aspects such as bit-rate and memory usage have not been considered in this optimization model.

In order to bridge the above gap in the state-of-the-art in video CODEC optimization research, this thesis proposes a parameter-based, multi-objective optimization framework for H.264 video coding (see Chapter 6 and Chapter 7). In particular the thesis focuses on the development of a framework where a joint complexity-memory-rate-distortion (C-M-R-D) optimization of H.264 video encoding/decoding can be achieved. An important aspect of the proposed framework is that it jointly considers the optimization of multiple objectives in both the encoder and decoder. Further an Evolutionary Algorithm (EA) that is better suited for addressing multi-objective optimization problems is selected as the optimization algorithm rather than the frequently used Lagrange multiplier.

3.4 Conclusions

The aim of this chapter was to review different background technologies adopted within the context of the main research focus of this thesis. It first provided a review of research in texture coding of arbitrarily shaped video objects, which focused on a detailed discussion of existing approaches to video object coding and their underlying principles/methods. Particular emphasis was given to the analysis of their performance, advantages and disadvantages. It was shown that SA-DCT used in MPEG-4 is the most efficient approach that can be used for dealing with the texture coding of arbitrarily shaped objects, when considering computational complexity and rate-distortion performance. However it was pointed out that due to the need of maintaining integer arithmetic within the integer transform (IT) and quantization stages in H.264, the direct adaptation of SA-DCT principles is not possible in introducing object scalability to H.264. Therefore, a novel SA-IT (see Chapter 4) based on the principles of SA-DCT will be designed and implemented within a

H.264 CODEC (see Chapter 5) to achieve the goals of the research presented in the first part of this thesis.

The chapter proceeded to provide an overview of optimization approaches proposed for video coding during the past decade. It categorized the state-of-the-art optimization techniques into two groups, namely algorithm-based and parameter-based. It was revealed that despite its practical importance in video coding; only one such attempt has been made in the past at parameter-based optimization. The said attempt was limited to the joint C-D optimization of a H.263 encoder. It was shown that the multi-objective optimization of an entire H.264 CODEC (i.e. both the encoder and decoder) that includes the consideration of further important coding objectives, such as rate and memory utilization, is of utmost importance for state-of-the-art in video coding research. Chapter 6 and Chapter 7 provide details of this research contribution

Chapter 4 Shape Adaptive Integer Transform and Quantization

4.1 Introduction

The approaches used in MPEG-4 for video object-based coding have been discussed in Chapter 2 , in particular the shape adaptive discrete cosine transform (SA-DCT) [24] based approach for coding the texture of arbitrarily shaped video objects. In this chapter a novel Shape Adaptive Integer Transform (SA-IT) is derived with particular attention given to maintaining integer divisions during the subsequent quantization step. In Chapter 5, we use the theory presented in this chapter to provide arbitrary shaped object coding in the H.264 standard.

The SA-IT considerably differs from the SA-DCT and therefore calls for novel design and implementation considerations based on combining those merits of both SA-DCT and IT [17] algorithms. As a mathematical transform used in MPEG-4, the SA-DCT has its advantages: adaptability at object edges, low complexity and block-based DCT. However, all such previous work adopted a floating point arithmetic design and implementation for SA-DCT, which is not suitable to be used in conjunction with the integer arithmetic-based IT used in H.264 texture coding. Although the use of IT allows increased decoding speed and reduced complexity of the decoders, it has the limitation that if arbitrarily shaped objects are to be coded, pixels outside the object will have to be considered in filling the boundary blocks of the object before being transformed as a block. This would result in a waste of computing power and processing time. The above limitation has been the key motivation behind the development of the SA-IT theory presented in this chapter.

The organization of this chapter is as follows. Section 4.2 provides an overview of the proposed algorithm. Section 4.3 provides the theoretical derivation of the

proposed SA-IT. A method for incorporating the transformed scaling factors within the quantization process is described in Section 4.4. An example of the process of SA-IT is given in Section 4.5. The 1-D SA-IT used as a solution to solve the sub-coefficients problem caused by the proposed SA-IT (which is essentially a 2-D transform, i.e., 2-D SA-IT) is presented in Section 4.6. Some preliminary simulation results of using 1-D and 2-D SA-ITs in video coding and an analysis are presented in Section 4.7. Finally a chapter conclusion is provided in Section 4.8 with an insight into Chapter 5.

4.2 An Overview

The basic idea of the proposed SA-IT is to transform a 4 x 4 boundary block of an arbitrarily shaped image object by cascading column and row DCT transforms. The transform is separated into two parts: (i) an integer part (i.e., core part) which is implemented with 16-bit integer arithmetic using only additions/shifts [Note: this avoids the traditional transform's mismatch problem between the decoded data in the encoder and the decoder, which arises from the fact that the inverse transform in traditional DCT is not fully specified in integer arithmetic [17]]; (ii) scaling factors (floating point numbers) which are produced by factorizing the direct and inverse transformation matrices (see $E'_{M(i)}$ and $E'^{T}_{M(i)}$ of Equation (4.13) and (4.14) respectively). To conform to the transform and quantization used in H.264 standard, the scaling factors, i.e., the post- and pre-scaling factors, are incorporated into the quantizer, reducing the total number of scaling multiplications. The post- and pre-scaling factors are normalizied into two separate quantization and reconstruction tables by different quantization parameters (QPs). The quantization and reconstruction tables are designed to avoid divisions and/or floating point arithmetic at the encoder and the decoder, and to ensure that data can be processed in 16-bit arithmetic.

A block diagram representing the process of 2-D SA-IT and its associated quantization stages is illustrated in Figure 4-1. Note that the 2-D SA-IT is essentially carried out as two cascaded 1-D SA-ITs. The input residual boundary block X, first

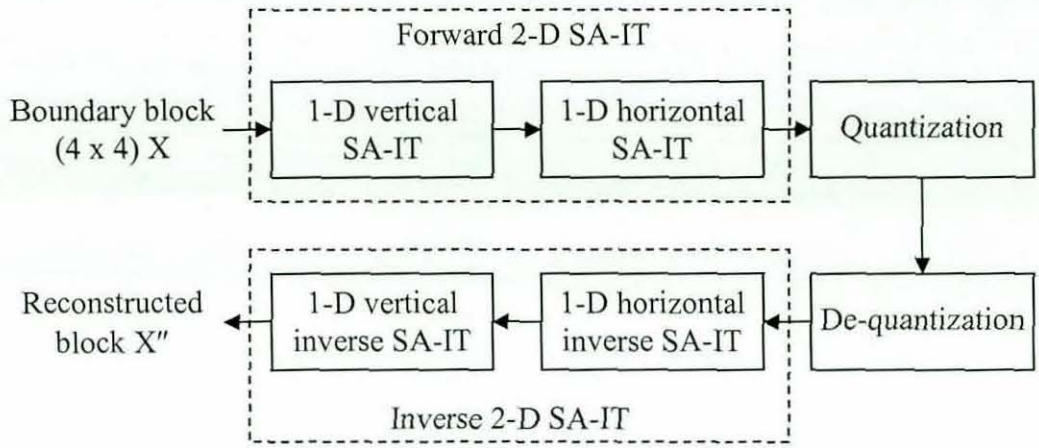


Figure 4-1 Flow diagram of 2-D SA-IT and its associated quantization

goes through forward 1-D vertical and horizontal SA-ITs before quantization. At the decoder end the inverse processes (i.e. of quantization and transforms) are executed in the reverse order. Finally, outputs the reconstructed block X''.

For clarity of explanation, an example of the forward SA-IT algorithm on a 4 x 4 image block that fully encloses an arbitrarily shaped object is illustrated in Figure 4-2. Figure 4-2(a) shows the division of the pixels within the said block into two groups, namely; foreground (shaded grey) and background (white) pixels. The foreground pixels are encoded with SA-IT by first applying a vertical one-dimensional transform, followed by a horizontal one-dimensional transform on the resulting vertically transformed foreground object. This is done as follows: firstly, the length $N(j)$ ($1 \leq N(j) \leq 4$) of every column j ($1 \leq j \leq 4$) of the foreground pixels X_j are calculated. Then, each column is shifted up and finally aligned with the upper border of the block as shown in Figure 4-2(b). After applying SA-IT in vertical direction, the DC coefficients (denoted by ■ in Figure 4-2(c)) for each column are found along the upper edge of the block. Next, the rows are shifted to align at the left border of the block (see Figure 4-2(d)) and a horizontal one-dimensional SA-IT transform is performed on each row of coefficients Y_i. Finally, the resulting transformed coefficients Z_i within the 4 x 4 boundary block are shown in Figure 4-2(e). Note that the final DC coefficient (denoted by ■) for the whole boundary block is located in the

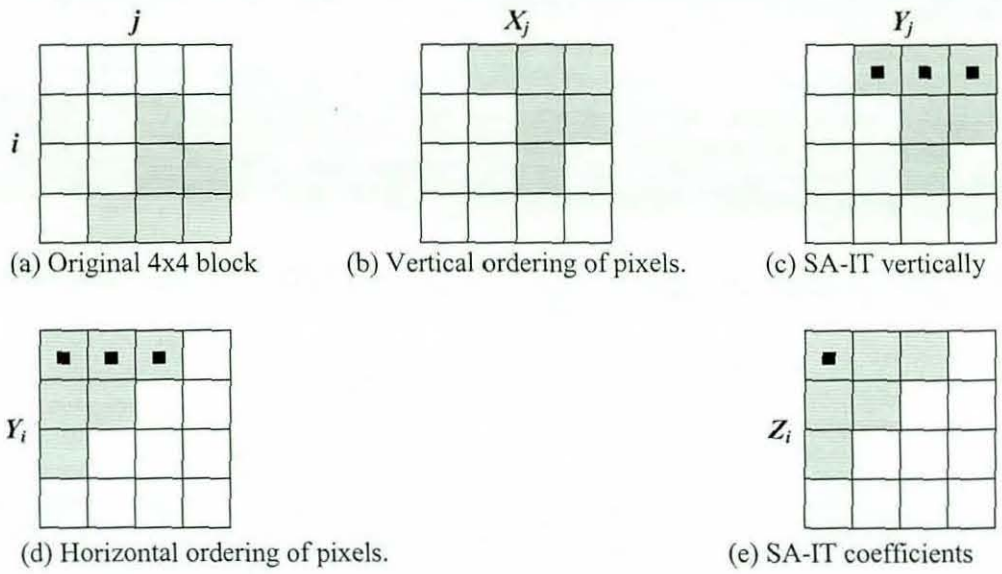


Figure 4-2 Example of forward SA-IT in a 4x4 block with arbitrary shape

upper left border of the block. The remaining coefficients are concentrated around the DC coefficient depending on the actual shape of the arbitrarily shaped object. The detailed design of the SA-IT is presented in the following sections.

4.3 Transform Design

4.3.1 Forward 1-D vertical Transform

For a given 4 x 4 block such as the one illustrated in Figure 4-2, for a column of foreground pixels, X_j (marked with grey), of length $N(j)$ ($1 \le N(j) \le 4$), the associated DCT transform matrix $A_{N(j)}$ is given by Equation (2.13) which is re-used as follows:

$$A_{N(j)}(p,k) = c_0 \cos\left[\frac{(2k+1)p\pi}{2N(j)}\right] \qquad p,k = 0 \rightarrow N(j)-1 \tag{4.1}$$

where $c_0 = \sqrt{1/2}$ if $p = 0$, and $c_0 = 1$ otherwise, and p, k denote the p^{th} row and k^{th} column DCT basis element respectively. Therefore the vertical SA-IT coefficients of column j, Y_j can be obtained by the following formula (similar to Equation (2.14)):

$$Y_j = \sqrt{2/N(j)}\,A_{N(j)}X_j, \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4 \tag{4.2}$$

Equation (4.2) may be written as follows:

$$Y_j = B_{N(j)}X_j, \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4$$
$$B_{N(j)} = \sqrt{2/N(j)}\,A_{N(j)} \tag{4.3}$$

where $B_{N(j)}$ is a $N(j)$ x $N(j)$ scale transform matrix. According to Equation (4.3), the scaled transform matrices for different $N(j)$'s can be obtained as follows:

When:

$N(j) = 1$,

$$B_1 = \sqrt{2}A_1 = \sqrt{2}\left(\sqrt{\frac{1}{2}}\cos 0\right) = 1.$$

$N(j) = 2$,

$$B_2 = \sqrt{\frac{2}{2}}A_2 = \begin{bmatrix} \sqrt{1/2} & \sqrt{1/2} \\ \sqrt{1/2} & -\sqrt{1/2} \end{bmatrix}.$$

$N(j) = 3$,

$$B_3 = \sqrt{\frac{2}{3}}A_3 = \begin{bmatrix} \sqrt{1/3} & \sqrt{1/3} & \sqrt{1/3} \\ \sqrt{1/2} & 0 & -\sqrt{1/2} \\ \sqrt{1/6} & -\sqrt{2/3} & \sqrt{1/6} \end{bmatrix}.$$

$N(j) = 4$,

$$B_4 = \sqrt{\frac{2}{4}}A_4 = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{1/2}\cos\frac{\pi}{8} & \sqrt{1/2}\cos\frac{3\pi}{8} & -\sqrt{1/2}\cos\frac{3\pi}{8} & -\sqrt{1/2}\cos\frac{\pi}{8} \\ 1/2 & -1/2 & -1/2 & 1/2 \\ \sqrt{1/2}\cos\frac{3\pi}{8} & -\sqrt{1/2}\cos\frac{\pi}{8} & \sqrt{1/2}\cos\frac{\pi}{8} & -\sqrt{1/2}\cos\frac{3\pi}{8} \end{bmatrix}.$$

In order to implement integer arithmetic, $B_{N(j)}$ can be factorized in the following form:

$$B_{N(j)} = C_{N(j)} \otimes E_{N(j)}, \qquad \text{i.e.,}$$

$$B_1 = [1] \otimes [1]$$

$$B_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} a & a \\ a & a \end{bmatrix}$$

$$B_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -2 & 1 \end{bmatrix} \otimes \begin{bmatrix} b & b & b \\ a & a & a \\ c & c & c \end{bmatrix}$$

$$B_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a^2 & a^2 & a^2 & a^2 \\ d & d & d & d \\ a^2 & a^2 & a^2 & a^2 \\ d & d & d & d \end{bmatrix}$$

(4.4)

where $C_{N(j)}$ is an integer transform matrix consisting of constant integers (Note that $C_{N(j)}$ is designed to retain the smallest possible integers. As discussed subsequently in Section 4.3.2, it minimizes the possible increase of the dynamic range of the transformed coefficients, thus resulting in an improved compression performance), and $E_{N(j)}$ is a matrix of scaling factors that comprises fractional elements such as a, b, c and d. $a = \sqrt{1/2}$, $b = \sqrt{1/3}$, $c = \sqrt{1/6}$, $d = \sqrt{1/10}$. The symbol $\otimes$ indicates that each element of matrix on the left is multiplied by the scaling factor in the corresponding position in the matrix on the right (i.e., the symbol $\otimes$ denotes Hadamard scalar product rather than matrix multiplication [11]). The scaled transform matrices B_1, B_2 and B_3 in Equation (4.4) can be easily factorized as described above. However, the factorization of B_4 in a similar manner is more complex. In order to factorize B_4, we first represent it as follows:

$$B_4 = \begin{bmatrix} e & e & e & e \\ f & g & -g & -f \\ e & -e & -e & e \\ g & -f & f & -g \end{bmatrix}$$

(4.5)

here

$$e = \frac{1}{2} \qquad f = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \qquad g = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right).$$

Thereafter, Equation (4.5) can be factorized as follows:

$$B_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & h & -h & -1 \\ 1 & -1 & -1 & 1 \\ h & -1 & 1 & -h \end{bmatrix} \otimes \begin{bmatrix} e & e & e & e \\ f & f & f & f \\ e & e & e & e \\ f & f & f & f \end{bmatrix} \tag{4.6}$$

where

$$h = g/f \approx 0.414.$$

To simplify the implementation of the transform, h is approximated by 0.5 following a strategy similar to that used in IT [11, 17]. In order to ensure that the matrix remains orthogonal, f also needs to be modified to $\sqrt{2/5}$. Thus, Equation (4.6) can be represented as:

$$B_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \otimes \begin{bmatrix} e & e & e & e \\ f & f & f & f \\ e & e & e & e \\ f & f & f & f \end{bmatrix} \tag{4.7}$$

Since the matrix on the left of Equation (4.7) can be further factorized so that the matrix contains integers only, Equation (4.7) can be written as follows:

$$B_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} e & e & e & e \\ f/2 & f/2 & f/2 & f/2 \\ e & e & e & e \\ f/2 & f/2 & f/2 & f/2 \end{bmatrix} \tag{4.8}$$

Finally, since $e = a^2$ and set $d = f/2$, the above equation may be re-written in a form similar to that of Equation (4.4), as follows:

$$B_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a^2 & a^2 & a^2 & a^2 \\ d & d & d & d \\ a^2 & a^2 & a^2 & a^2 \\ d & d & d & d \end{bmatrix}$$

Thus, after factorizing these scaled transform matrices $B_{N(j)}$, Equation (4.3) may be modified as follows:

$$Y_j = \left(C_{N(j)} \otimes E_{N(j)}\right) X_j = \left(C_{N(j)} X_j\right) \otimes E_{N(j)} \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4 \tag{4.9}$$

where $C_{N(j)}$ and $E_{N(j)}$ are the same definitions as above Equation (4.4). The symbol $\otimes$ within the context of this thesis represents Hadamard scalar product as described above. Note that the result of the SA-IT will not be identical to the 4 x 4 SA-DCT because of the change to factor d.

4.3.2 Forward 1-D horizontal Transform

As a result of performing the vertical SA-IT along columns as depicted above, the resulting coefficients are left aligned as shown in Figure 4-2(d). Subsequently following a strategy similar to that used in obtaining Equation (4.9), the formula for one-dimensional horizontal SA-IT along rows could be obtained:

$$Z_i = \left(C_{M(i)} Y_i\right) \otimes E_{M(i)} \qquad 1 \le i \le 4, \quad 1 \le M(i) \le 4 \tag{4.10}$$

where $M(i)$ refers to the length of row i (i.e. the number of the transformed coefficients of row i). $C_{M(i)}$ and $E_{M(i)}$ are defined similar to $C_{N(j)}$ and $E_{N(j)}$ respectively. Y_i is the ith row's coefficients that were obtained by Equation (4.9). Z_i is the ith row's

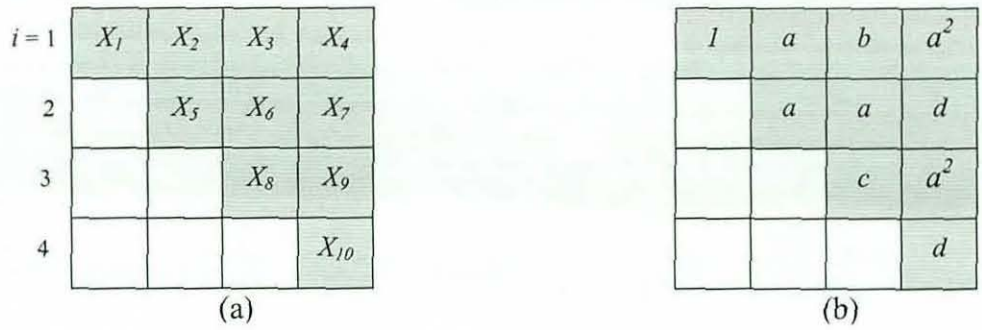


Figure 4-3 Results of vertical transformed coefficients (a) and scaling factors (b)

coefficients after horizontal transformation. There exists a problem that arises from the fact that Y_i in Equation (4.10) not only contains integers, but also fractions (scaling factors shown in Equation (4.4)). Thus, Equation (4.10) is required to be factorized further. However, since different lengths, *N(j)* result in different scaling factors in Y_i, it is impossible to decompose the Y_i into the form of an integer and a fraction. For instance, the following Figure 4-3 shows the results of coefficients and scaling factors after vertical transformation. According to Equation (4.10), when $i = 3$, the corresponding horizontal transform can be written as follows:

$$Z_3 = \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} cX_8 \\ a^2 X_9 \end{bmatrix} \right) \otimes \begin{bmatrix} a \\ a \end{bmatrix}$$
$$= \begin{bmatrix} cX_8 + a^2 X_9 \\ cX_8 - a^2 X_9 \end{bmatrix} \otimes \begin{bmatrix} a \\ a \end{bmatrix} \qquad (4.11)$$

From the above equation, it is obvious that it can not be split into an integer and a fraction due to the different scaling factors c and a^2. In order to solve the above problem, it is desirable to decompose Equation (4.9) (i.e. the vertical transform) into the following form:

$$Y_j = \sum_{k=1}^{N(j)} \left(C_{N(j)(k)} X_{j(k)} \right) \otimes E_{N(j)(k)} \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4$$

(4.12)

where $C_{N(j)(k)}$ is the kth column of $C_{N(j)}$. $E_{N(j)(k)}$ represents the scaling factors in the kth column of scaling matrix $E_{N(j)}$. $X_{j(k)}$ represents the kth internal pixel (i.e. grey pixel) of X_j as in Figure 4-2. Accordingly, the corresponding equation for horizontal transform may be modified to:

$$Z_i = \sum_{k=1}^{M(i)} \left(C_{M(i)(k)} Y_{i(k)} \right) \otimes E'_{M(i)(k)} \qquad 1 \le i \le 4, \quad 1 \le \mathrm{M}(i) \le 4$$

(4.13)

$C_{M(i)(k)}Y_{i(k)}$ is the core two-dimensional transform. $E'_{M(i)(k)}$ is the k^{th} column of the matrix of scaling factors in Equation (4.4) multiplied by the k^{th} scaling factor of row i resulting from Equation (4.12). Therefore, Equation (4.11) can be re-written as follows:

$$\begin{aligned} Z_3 &= \sum_{k=1}^{2} \left(C_{2(k)} \cdot Y_{2(k)} \right) \otimes E'_{2(k)} = C_{2(1)} \cdot Y_{2(1)} \otimes E'_{2(1)} + C_{2(2)} \cdot Y_{2(2)} \otimes E'_{2(2)} \\ &= \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot X_8 \right) \otimes \begin{bmatrix} ac \\ ac \end{bmatrix} + \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot X_9 \right) \otimes \begin{bmatrix} a^3 \\ a^3 \end{bmatrix} \end{aligned}$$

Table 4-1 Scaling factors of constituting elements of $E'_{M(i)}$.

	1	**a**	**b**	**c**	**d**	**a^2**
1	1	a	b	c	d	a^2
a	a	a^2	ab=c	ac	ad	a^3
b	b	ab	b^2	bc	bd	a^2b=ac
c	c	ac	bc	c^2	cd	a^2c
d	d	ad	bd	cd	d^2	a^2d
a^2	a^2	a^3	a^2b	a^2c	a^2d	a^4

From the above equation, we may notice that the scaling factors in $E'_{M(i)}$ are different from $E_{N(j)}$ which consists of one or more of six scaling factors (i.e, 1, a, b, c, a^2, and d), while $E'_{M(i)}$ is made up of eighteen scaling factors as depicted in Table 4-1 highlighted in grey for clarity.

As we mentioned at the beginning of this chapter, the new transform can be computed in 16-bit arithmetic that conforms to H.264 standard. In the integer transform matrices C_1 to C_4 in Equation (4.4), the maximum sum of absolute values in any row of these matrices is in C_4 and equals 6. If the maximum pixel value of a 4 x 4 image block equals to A, then the maximum value of direct transformed coefficient is 6A, i.e., the transform has a dynamic range gain of 6. Thus, the maximum dynamic range gain increase for a 2-D transform is $\log_2(6^2) = 5.17$, i.e., storage of transformed coefficient needs only six more bits than initial 8-bit pixel data or 9-bit residual data. So the transform is implemented within a 16-bit arithmetic.

It is desirable to note that the integer transform matrices C_3 and C_4 are orthogonal but do not have the same norm. However, that can be easily compensated for in the quantization process, as we discuss in Section 4.4.

4.3.3 Inverse Transform

In relation to Equations (4.13) and (4.12) respectively, the inverse transforms in horizontal and vertical directions respectively and their factorizations can be summarized by the following equations:

$$Y_i = \sum_{k=1}^{M(i)} C^T_{M(i)(k)} \left(Z_{i(k)} \otimes E'^T_{M(i)(k)} \right) \qquad 1 \le M(i) \le 4,\ 1 \le i \le 4 \tag{4.14}$$

$$X^*_j = \sum_{k=1}^{N(j)} C^T_{N(j)(k)} Y_{j(k)} \qquad 1 \le N(j) \le 4,\ 1 \le j \le 4 \tag{4.15}$$

here, $Z_{i(k)}$ and $Y_{j(k)}$ are respectively the k^{th} forward horizontal transform coefficient of row i of the SA-IT transformed block and the k^{th} inverse horizontal transform vector of column j. $C^T_{M(i)(k)}$, $C^T_{N(j)(k)}$ and $E'^T_{M(i)(k)}$ represent the transposes of $C_{M(i)(k)}$, $C_{N(j)(k)}$ and $E'_{M(i)(k)}$ respectively. Finally, the reconstructed pixels X^*_j are replaced in the initial positions of the block as in Figure 4-2(a).

However, combined rounding errors arise from the inverse transform and reconstruction. In order to minimize the errors, we need to reduce the dynamic range gain in the 2-D inverse transform. The problem is in the odd-symmetric basis functions of C_4, whose peak value is two. We scaled the odd-symmetric basis functions by 1/2; that is, using Equation (4.7) as the inverse transform matrix. Thus, the maximum sum of absolute values of C_4 in Equation (4.7) now equals 4, which reduces the dynamic range gain for the 2-D inverse transform from 6^2 to 4^2. Since $\log_2(4^2) = 4$, the increase in dynamic range is reduced from 6 bits to 4 bits. The factors $\pm 1/2$ in the inverse transform matrix (Equation (4.7)) can be implemented by 1-bit right shifts [17]. Although small errors would be caused by the right shifts, [17] has shown that the errors can be compensated by the 2-bit gain in the dynamic range of the input to the inverse transform.

4.4 Quantization Design

Due to the need of conformability with H.264 standard, the quantization design procedure adopted in our proposal is similar to the H.264 standard's quantization design [17] procedure. Thus, the proposed quantization design must fulfil the following requirements.

- The post- and pre-scaling factors, i.e., $E'_{M(i)}$ and $E'^T_{M(i)}$, need to be integrated into forward and inverse quantizers individually;
- The need for divisions should be avoided;
- Implementation should be done using16-bit arithmetic.

In Equation (4.13), the output of the forward transform consists of two parts, (i) integer part, $W_{M(i)(k)} = C_{M(i)(k)}Y_{i(k)}$; (ii) a non-integer part (i.e. it consists of the post-scaling factors) $E'_{M(i)(k)}$. To achieve the first requirement, the $E'_{M(i)(k)}$ is incorporated into the forward quantization process. Thus, the integers, transformed coefficients, $W_{M(i)(k)}$ are quantized and scaled by a single operation as follows:

$$Q_i = round\left(\sum_{k=1}^{M(i)} W_{M(i)(k)} \otimes E'_{M(i)(k)} / Qstep\right) \tag{4.16}$$

where Q_{step} is a quantizer step size indexed by QP in the range of 0 to 51, inclusive. Q_{step} and QP have been introduced in Section 2.3.7.3. The rounding operation here approximates towards smaller integers. Q_i represents the quantized coefficients of row i.

A disadvantage of the above quantization formula is that it requires integer divisions. To avoid divisions, following the approach used within H.264 reference model software [51], we apply the factor $(E'_{M(i)(k)}/Q_{step})$ in Equation (4.16), as a multiplication by a quantization factor MF and a right shift, thus avoiding actual division operations. i.e., Equation (4.16) can be re-written as follows:

$$Q_i = round\left(\sum_{k=1}^{M(i)} W_{M(i)(k)} \otimes \frac{MF_{M(i)(k)}}{2^{qbits}}\right) \tag{4.17}$$

where

$$\frac{MF_{M(i)(k)}}{2^{qbits}} = \frac{E'_{M(i)(k)}}{Qstep}, \quad qbits = 15 + floor(QP/6).$$

In integer arithmetic, Equation (4.17) can be implemented as follows:

$$|Q_i| = \sum_{k=1}^{M(i)} \left(\left| W_{M(i)(k)} \right| \otimes MF_{M(i)(k)} + F_{M(i)} f \right) \gg qbits = \sum_{k=1}^{M(i)} R_{M(i)(k)}$$
$$sign(Q_i) = sign\left(\sum_{k=1}^{M(i)} W_{M(i)(k)} \right)$$

(4.18)

where the symbol (>>*qbits*) indicates the *qbits*-bit right shift that is equivalent to a division by 2^{qbits}. $F_{M(i)}$ is a column vector of size *M(i)* x 1 , in which all elements are one. The so-called dead-zone control parameter f is set to $2^{qbits}/3$ for intra blocks or $2^{qbits}/6$ for inter blocks by the encoder in our implementation. $R_{M(i)(k)}$ is the sub-quantization coefficient(s) of each $W_{M(i)(k)}$, and Q_i represents the quantized coefficients of row *i*.

The pre-scaling factor $E'^{T}_{M(i)(k)}$ for the inverse transform in Equation (4.14) is incorporated into inverse quantization (reconstruction), together with a constant scaling factor of 64 to avoid rounding errors [11], thus the corresponding reconstruction formula that we proposed is:

$$\begin{aligned} W_i^* &= Q_i \otimes \sum_{k=1}^{M(i)} \left(E'^{T}_{M(i)(k)} \cdot Qstep \cdot 64 \right) \\ &= \sum_{k=1}^{M(i)} R_{M(i)(k)} \otimes E'^{T}_{M(i)(k)} \cdot Qstep \cdot 64. \end{aligned}$$

(4.19)

W_i^* are de-quantized coefficients of row *i*. The reconstruction factor $E'^{T}_{M(i)(k)} \cdot Qstep \cdot 64$ is replaced by reconstruction factor, $RF_{M(i)(k)}$. Thus, the reconstruction formula may be re-written as follows:

$$W_i^* = \sum_{k=1}^{M(i)} \left(R_{M(i)(k)} \otimes RF_{M(i)(k)} \right) \ll floor(QP/6)$$
$$RF_{M(i)(k)} = round\left(E'^{T}_{M(i)(k)} Qstep \cdot 64 \right)$$

(4.20)

where the symbol << denotes a binary left shift. Note that finally the values at the output of the inverse transform are divided by 64 to remove the scaling introduced in Equation (4.19). This is achieved using a right shift operation.

In the proposed SA-IT quantization process design, the quantization and reconstruction factors, *MF* and *RF* are obtained via lookup table as illustrated in Table 4-2 and Table 4-3 respectively (the definition of the values of *MF* and *RF* are discussed later). Note that only the first six values of *MF* and *RF* are used by the proposed SA-IT due to the fact that for every increase of "six" in QP, the denominator 2^{qbits} in Equation (4.18) doubles, but the factors *MF* remain unchanged. We use a total of 18 scaling factors that depend on the actual arbitrarily shaped object within the block. Since the scaling factors in $E'_{M(i)}$ are unpredictable, a question of obtaining the factors in our implementation (discussed in Chapter 5) arises. To avoid using floating point values of the scaling factors to look up the quantization table, it is desirable to create integer indices to the array of scaling factors as illustrated in Table 4-2 and Table 4-3. We first use a number between 0 to 17 to *index* each row of Table 4-2 and Table 4-3 factor to pick up the associated QP values. Further we use 1, 2, 3, 6 and 7 as a *code* to represent the five basic factors, 1, *a*, *b*, *c*, *d* respectively, resulting in all 18 scaling factors being representable by unique codes. The maximum valued unique code is 49, which is for d^2, and the corresponding relationship of *index* (i.e. 16) and *code* is indicated in Table 4-2 and Table 4-3.

As we mentioned in Section 4.3, C_3 and C_4 are orthogonal but do not have the same norm. In the quantization process it is necessary to compensate for the different row norms (3, 2 and 6 in C_3; 4, 10, 4 and 10 in C_4). The scaling factors needed for compensation are depicted in Table 4-4. Note that since the odd-rows of C_4 are scaled by 1/2 in the inverse transform (i.e., to reduce the dynamic range gain (see Section 4.3.3)), the compensated values for all the scaling factors that involves *d*, i.e., *d*, *ad*, *bd*, *cd*, a^2d and d^2, are halved.

Table 4-2 Quantization factor MF for $0 \le QP \le 5$

Factor	Index	Code	QP for MF					
			0	1	2	3	4	5
a	0	2	37449	33825	28340	26214	23301	20560
b	1	3	30393	27962	23302	21845	18893	16644
c	2	6	21845	19418	16644	15197	13443	12052
d	3	7	16777	14980	12710	11984	10485	9118
a^2	4	4	26214	23831	20165	18724	16384	14563
ac	5	12	14564	13443	11650	10922	9709	8322
ad	6	14	11651	10486	9118	8389	7232	6553
a^3	7	8	18725	16384	14563	13107	11398	10485
bc	8	18	12945	11651	9709	8962	7767	6853
bd	9	21	9321	8738	7358	6990	6079	5377
cd	10	42	6991	6355	5377	4993	4112	3679
a^2c	11	24	10923	9709	7944	7943	6721	5825
a^2d	12	28	8066	7490	6553	5825	5242	4559
l	13	1	52429	47662	40329	37449	32768	29127
b^2	14	9	17924	15534	13706	12264	11096	9709
c^2	15	36	8322	8322	6473	6472	5296	4854
d^2	16	49	5243	4660	4194	3813	3226	2995
a^4	17	16	13107	11915	10082	9362	8192	7281

Table 4-3 Rescaling factor (RF) for $0 \le QP \le 5$

Factor	Index	Code	QP for MF					
			0	1	2	3	4	5
a	0	2	28	31	37	40	45	51
b	1	3	23	25	30	32	37	42
c	2	6	16	18	21	23	26	29
d	3	7	25	28	33	35	40	46
a^2	4	4	20	22	26	28	32	36
ac	5	12	12	13	15	16	18	21
ad	6	14	18	20	23	25	29	32
a^3	7	8	14	16	18	20	23	25
bc	8	18	9	10	12	13	15	17
bd	9	21	15	16	19	20	23	26
cd	10	42	10	11	13	14	17	19
a^2c	11	24	8	9	11	11	13	15
a^2d	12	28	13	14	16	18	20	23
l	13	1	40	44	52	56	64	72
b^2	14	9	13	15	17	19	21	24
c^2	15	36	7	7	9	9	11	12
d^2	16	49	8	9	10	11	13	14
a^4	17	16	10	11	13	14	16	18

Table 4-4 Compensations of scaling factors

Scaling Factor	**Compensated Value (CV)**
a	2
b	3
c	6
d	5
a^2	4
ac	12
ad	10
a^3	8
bc	18
bd	15
cd	30
a^2c	24
a^2d	20
1	1
b^2	9
c^2	36
d^2	50
a^4	16

Since in the standard [16] only the decoder is specified, the *RF* may be computed from the reconstruction Equation (4.20), which is depicted in Table 4-3. To follow a strategy similar to that in [17], the *MF* and *RF* were designed to maximize dynamic range and to satisfy a similar expression to that in [16], namely:

$$MF_{M(i)(k)} RF_{M(i)(k)} CV_{M(i)(k)} \cong 2^{21} \tag{4.21}$$

where $CV_{M(i)(k)}$ represent the corresponding compensation values. Thus, according to the above equation, *MF* can be calculated easily and shown in Table 4-2.

With the transform design discussed in Section 4.3 and the quantization design above, we see that all operations can be computed in 16-bit arithmetic, for input data with 9-bit dynamic range. It is because the inputs (8-bit pixel data (0-255)) to the transform

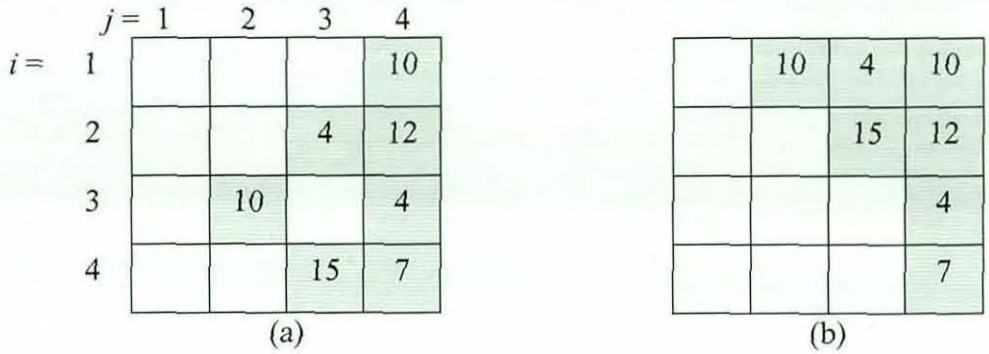


Figure 4-4 (a) Initial block; (b) After shifting to upper border

are prediction residuals/errors that are in a -255 to 255 (9-bit) dynamic range. However, there is one exception in the quantization Equation (4.18), i.e., the product $|W_{M(i)(k)}| \otimes MF_{M(i)(k)}$ has a 32-bit dynamic range, but the final quantized value is guaranteed to fall within a 16-bit range.

4.5 Example of 2-D SA-IT's Process

This section illustrates an example of applying the proposed 2-D SA-IT on a selected arbitrarily shaped block. Figure 4-4 (a) shows an initial 4 x 4 block with arbitrarily shaped pixels highlighted with grey and Figure 4-4 (b) illustrates these pixels being shifted to upper border in preparation for a forward vertical transform. The detailed operation steps are shown in the following statements.

Step 1: The forward 1-D vertical transform is executed by Equation (4.12) and the corresponding results for each column are as follows.

$$Y_1 = 0,$$

$$Y_2 = (C_1 \otimes E_1) \cdot X_2 = \sum_{k=1}^{1} \left(C_{1(k)} \cdot X_{2(k)} \right) \otimes E_{1(k)} = \left(C_{1(1)} \cdot X_{2(1)} \right) \otimes E_{1(1)} = 10,$$

$i=$ \ $j=$	1	2	3	4
1		10	19	33
2			-11	14
3				1
4				-13

(a)

	1	a	a^2
		a	d
			a^2
			d

(b)

Figure 4-5 (a) Resulting coefficients of the vertical transform and (b) corresponding scaling factors

$$Y_3=(C_2\otimes E_2)\cdot X_3=\sum_{k=1}^{2}\left(C_{2(k)}\cdot X_{3(k)}\right)\otimes E_{2(k)}=\left(C_{2(1)}\cdot X_{3(1)}\right)\otimes E_{2(1)}+\left(C_{2(2)}\cdot X_{3(2)}\right)\otimes E_{2(2)}$$
$$=\left(\begin{bmatrix}1\\1\end{bmatrix}\cdot[4]\right)\otimes\begin{bmatrix}a\\a\end{bmatrix}+\left(\begin{bmatrix}1\\-1\end{bmatrix}\cdot[15]\right)\otimes\begin{bmatrix}a\\a\end{bmatrix}=\begin{bmatrix}4+15\\4-15\end{bmatrix}\otimes\begin{bmatrix}a\\a\end{bmatrix}=\begin{bmatrix}19\\-11\end{bmatrix}\otimes\begin{bmatrix}a\\a\end{bmatrix},$$

$$Y_4=(C_4\otimes E_4)\cdot X_4=\sum_{k=1}^{4}\left(C_{4(k)}\cdot X_{4(k)}\right)\otimes E_{4(k)}$$
$$=\left(C_{4(1)}\cdot X_{4(1)}\right)\otimes E_{4(1)}+\left(C_{4(2)}\cdot X_{4(2)}\right)\otimes E_{4(2)}+\left(C_{4(3)}\cdot X_{4(3)}\right)\otimes E_{4(3)}+\left(C_{4(4)}\cdot X_{4(4)}\right)\otimes E_{4(4)}$$
$$=\left(\begin{bmatrix}1\\2\\1\\1\end{bmatrix}\cdot[10]\right)\otimes\begin{bmatrix}a^2\\d\\a^2\\d\end{bmatrix}+\left(\begin{bmatrix}1\\1\\-1\\-2\end{bmatrix}\cdot[12]\right)\otimes\begin{bmatrix}a^2\\d\\a^2\\d\end{bmatrix}+\left(\begin{bmatrix}1\\-1\\-1\\2\end{bmatrix}\cdot[4]\right)\otimes\begin{bmatrix}a^2\\d\\a^2\\d\end{bmatrix}+\left(\begin{bmatrix}1\\-2\\1\\-1\end{bmatrix}\cdot[7]\right)\otimes\begin{bmatrix}a^2\\d\\a^2\\d\end{bmatrix}=\begin{bmatrix}33\\14\\1\\-13\end{bmatrix}\otimes\begin{bmatrix}a^2\\d\\a^2\\d\end{bmatrix}.$$

After the vertical transform, the resulting Y_j and $E_{N(j)}$ are shown in Figure 4-5.

Step 2: The forward 1-D horizontal transform is then performed by applying Equation (4.13) as follows:

$$Z_1=\sum_{k=1}^{3}\left(C_{3(k)}\cdot Y_{1(k)}\right)\otimes E'_{3(k)}=\left(C_{3(1)}\cdot Y_{1(1)}\right)\otimes E'_{3(1)}+\left(C_{3(2)}\cdot Y_{1(2)}\right)\otimes E'_{3(2)}+\left(C_{3(3)}\cdot Y_{1(3)}\right)\otimes E'_{3(3)}$$
$$=\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\cdot[10]\right)\otimes\begin{bmatrix}b\\a\\c\end{bmatrix}+\left(\begin{bmatrix}1\\0\\-2\end{bmatrix}\cdot[19]\right)\otimes\begin{bmatrix}ba\\a^2\\ca\end{bmatrix}+\left(\begin{bmatrix}1\\-1\\1\end{bmatrix}\cdot[33]\right)\otimes\begin{bmatrix}ba^2\\a^3\\ca^2\end{bmatrix}=\begin{bmatrix}10\otimes b+19\otimes ba+33\otimes ba^2\\10\otimes a+0\otimes a^2-33\otimes a^3\\10\otimes c-38\otimes ca+33\otimes ca^2\end{bmatrix},$$

$$Z_2 = \sum_{k=1}^{2}\left(C_{2(k)} \cdot Y_{2(k)}\right) \otimes E'_{2(k)} = \left(C_{2(1)} \cdot Y_{2(1)}\right) \otimes E'_{2(1)} + \left(C_{2(2)} \cdot Y_{2(2)}\right) \otimes E'_{2(2)}$$
$$= \left(\begin{bmatrix}1\\1\end{bmatrix} \cdot [-11]\right) \otimes \begin{bmatrix}a^2\\a^2\end{bmatrix} + \left(\begin{bmatrix}1\\-1\end{bmatrix} \cdot [14]\right) \otimes \begin{bmatrix}ad\\ad\end{bmatrix} = \begin{bmatrix}-11 \otimes a^2 + 14 \otimes ad\\-11 \otimes a^2 - 14 \otimes ad\end{bmatrix},$$

$$Z_3 = \sum_{k=1}^{1}\left(C_{1(k)} \cdot Y_{3(k)}\right) \otimes E'_{1(k)} = \left(C_{1(1)} \cdot Y_{3(1)}\right) \otimes E'_{1(1)} = 1 \otimes a^2,$$

$$Z_4 = \sum_{k=1}^{1}\left(C_{1(k)} \cdot Y_{4(k)}\right) \otimes E'_{1(k)} = \left(C_{1(1)} \cdot Y_{4(1)}\right) \otimes E'_{1(1)} = -13 \otimes d.$$

The intermediate outcomes of Z and scaling factors are illustrated in Figure 4-6. Note that each coefficient position (highlighted with grey) consists of the sum of those sub-coefficients, e.g., $10 \otimes b + 19 \otimes ba + 33 \otimes ba^2$.

Step 3: These scaling factors illustrated in Figure 4-6 will subsequently be integrated into the forward quantization process using Equation (4.18). The resulting forward quantized coefficients are illustrated in Figure 4-7 (a). Note that we have assumed that QP equals to 4 and f is set to $2^{qbits}/3$ (i.e., used for intra block) in this example.

When i = 1, M(1) = 3. The corresponding MF gained from Table 4-2 is as follows.

$$MF_{3(1)} = \begin{bmatrix}18893\\23301\\13443\end{bmatrix}, \qquad MF_{3(2)} = \begin{bmatrix}13443\\16384\\9709\end{bmatrix}, \qquad MF_{3(3)} = \begin{bmatrix}9709\\11398\\6721\end{bmatrix}.$$

	j=1	2	3	4
i=1		$10 \otimes b + 19 \otimes ba + 33 \otimes ba^2$	$10 \otimes a + 0 \otimes a^2 - 33 \otimes a^3$	$10 \otimes c - 38 \otimes ca + 33 \otimes ca^2$
2			$-11 \otimes a^2 + 14 \otimes ad$	$-11 \otimes a^2 - 14 \otimes ad$
3				$1 \otimes a^2$
4				$-13 \otimes d$

Figure 4-6 Intermediate outcomes of Z and scaling factors

The coefficients of the first row can be gained from:

$$Q_1 = \sum_{k=1}^{3}\left(\left|W_{3(k)}\right| \otimes MF_{3(k)} + F_3 f\right) \gg qbits$$

$$= \begin{bmatrix} (10\times 18918+10923)\gg 15+(19\times 13443+10923)\gg 15+(33\times 9709+10923)\gg 15 \\ (10\times 23301+10923)\gg 15+(0\times 16384+10923)\gg 15-(33\times 11398+10923)\gg 15 \\ (10\times 13443+10923)\gg 15-(38\times 9709+10923)\gg 15+(33\times 6688+10923)\gg 15 \end{bmatrix}$$

$$= \begin{bmatrix} 6+8+10 \\ 7-11 \\ 4-11+7 \end{bmatrix} = \begin{bmatrix} 24 \\ -4 \\ 0 \end{bmatrix}, \qquad R_{3(k)} = \begin{bmatrix} 6 \\ 7 \\ 4 \end{bmatrix} + \begin{bmatrix} 8 \\ 0 \\ -11 \end{bmatrix} + \begin{bmatrix} 10 \\ -11 \\ 7 \end{bmatrix}$$

Note that each quantized coefficient of the first row comprises the sum of three sub-coefficients. For example, the coefficient of the position (1, 2) is 24 which is the sum of sub-coefficients 6, 8 and 10.

When $i = 2$, M(2) = 2.

$$MF_{2(1)} = \begin{bmatrix} 16384 \\ \\ 16384 \end{bmatrix}, \qquad MF_{2(2)} = \begin{bmatrix} 7232 \\ \\ 7232 \end{bmatrix}.$$

The coefficients of the second row can be obtained from:

$j=$ / $i=$	1	2	3	4
1		24	-4	0
2			-2	8
3				0
4				-4

(a)

	610	62	-3
		-73	-247
			0
			-160

(b)

Figure 4-7 (a) Forward quantized coefficients; (b) Inverse quantized coefficients

$$Q_2 = \sum_{k=1}^{2}\left(\left|W_{2(k)}\right| \otimes MF_{2(k)} + F_2 f\right) \gg qbits$$

$$= \begin{bmatrix} -(11\times 16384 + 10923) \gg 15 + (14\times 7232 + 10923) \gg 15 \\ -(11\times 16384 + 10923) \gg 15 - (14\times 7232 + 10923) \gg 15 \end{bmatrix}$$

$$= \begin{bmatrix} -5+3 \\ -5-3 \end{bmatrix} = \begin{bmatrix} -2 \\ -8 \end{bmatrix}, \qquad R_{2(k)} = \begin{bmatrix} -5 \\ -5 \end{bmatrix} + \begin{bmatrix} 3 \\ -3 \end{bmatrix}$$

Note that each quantized coefficient of the second row comprises the sum of two sub-coefficients.

When $i = 3$, M(3) = 1 and $MF_{1(1)} = 16384$.

The coefficient of the third row can be gained from:

$$Q_3 = \sum_{k=1}^{1}\left(\left|W_{1(k)}\right| \otimes MF_{1(k)} + F_1 f\right) \gg qbits = \left[(1\times 16384 + 10923) \gg 15\right] = 0, \quad R_{1(k)} = 0$$

When $i = 4$, M(4) = 1 and $MF_{1(1)} = 10485$.

The coefficient of the fourth row can be calculated from:

$$Q_4 = \sum_{k=1}^{1}\left(\left|W_{1(k)}\right| \otimes MF_{1(k)} + F_1 f\right) \gg qbits = \left[-(13\times 10485 + 10923) \gg 15\right] = -4, \quad R_{1(k)} = -4$$

Note that the quantized coefficients for the third and fourth rows contain only one sub-coefficient respectively.

Step 4: The corresponding de-quantization (Equation (4.20)) is carried out as follows.

The associated *RFs* are as follows:

$i = 1$

$$V_{3(1)} = \begin{bmatrix} 37 \\ 45 \\ 26 \end{bmatrix} \qquad V_{3(2)} = \begin{bmatrix} 26 \\ 32 \\ 18 \end{bmatrix} \qquad V_{3(3)} = \begin{bmatrix} 18 \\ 23 \\ 13 \end{bmatrix}$$

$i = 2$

$$V_{2(1)} = \begin{bmatrix} 32 \\ 32 \end{bmatrix} \qquad V_{2(2)} = \begin{bmatrix} 29 \\ 29 \end{bmatrix}$$

$i = 3, \qquad V_{1(1)} = 32$

$i = 4, \qquad V_{1(1)} = 40$

Rescaling coefficients are obtained from the following calculations and results are illustrated in Figure 4-7 (b):

$$W_1^{'} = \begin{bmatrix} 6 \\ 7 \\ 4 \end{bmatrix} \otimes \begin{bmatrix} 37 \\ 45 \\ 26 \end{bmatrix} + \begin{bmatrix} 8 \\ 0 \\ -11 \end{bmatrix} \otimes \begin{bmatrix} 26 \\ 32 \\ 18 \end{bmatrix} + \begin{bmatrix} 10 \\ -11 \\ 7 \end{bmatrix} \otimes \begin{bmatrix} 18 \\ 23 \\ 13 \end{bmatrix} = \begin{bmatrix} 222+208+180 \\ 315-253 \\ 104-198+91 \end{bmatrix} = \begin{bmatrix} 610 \\ 62 \\ -3 \end{bmatrix},$$

$$W_2^{'} = \begin{bmatrix} -5 \\ -5 \end{bmatrix} \otimes \begin{bmatrix} 32 \\ 32 \end{bmatrix} + \begin{bmatrix} 3 \\ -3 \end{bmatrix} \otimes \begin{bmatrix} 29 \\ 29 \end{bmatrix} = \begin{bmatrix} -160+87 \\ -160-87 \end{bmatrix} = \begin{bmatrix} -73 \\ -247 \end{bmatrix},$$

$$W_3^{'} = 0,$$

$$W_4^{'} = [-4] \otimes [40] = -160$$

Step 5: The horizontal inverse transform is carried by Equation (4.14) and the corresponding results are illustrated in Figure 4-8 (a).

Step 6: The reconstruction of transform is performed by Equation (4.15) for vertical inverse transform. The corresponding results are illustrated in Figure 4-8 (b).

Step 7: As mentioned in Section 4.4, the reconstruction values need to perform a 6-bit right shift operation and the final output is in Figure 4-8 (c).

It was noted that in the illustrated example above, a given coefficient (e.g., the first row of the block) may be made out of a maximum of three sub-coefficients (Note that in general, in a 4 x 4 block, a coefficient may be made out of a maximum of four sub-coefficients). It has been observed that this is due to (a) the use of different vertical (or horizontal) transform sizes and (b) the fact that post-scaling is absorbed in the quantizer operation. Therefore, for guaranteed decoding, all sub-coefficients will have to be transmitted to the decoder which directly results in a coding loss. Therefore, though 2-D SA-IT provides a theoretical solution to arbitrarily shaped object coding, a practical implementation may not be sufficiently efficient. In the following section we discuss a solution to this, i.e., the use of 1-D SA-IT.

	669	616	545
		-320	174
			0
			-160

(a)

	669	296	639
		936	792
			298
			451

(b)

			9
		4	12
	10		4
		14	7

(c)

Figure 4-8 Reconstruction results (a) horizontal; (b) vertical; (c) final results

4.6 1-D SA-IT and Quantization

As shown in Section 4.5, for guaranteed decoding of an arbitrarily shaped block, sub-coefficients that result from 2-D SA-IT are required to be transmitted to the decoder. This increases cost of transmission (see experimental results in Section 5.4). A solution to this problem is to use 1-D vertical SA-IT. This approach not only resolves the sub-coefficients problem but also reduces the overall computational cost of SA-IT (only performing 1-D transform). A block diagram of the 1-D vertical SA-IT and its associated quantization stages is illustrated in Figure 4-9. The detail definitions and design considerations of the 1-D SA-IT are discussed below.

For the forward 1-D vertical SA-IT, the equation is the same as Equation (4.9), which is re-written as follows:

$$Y_j = \left(C_{N(j)} \otimes E_{N(j)}\right) X_j = \left(C_{N(j)} X_j\right) \otimes E_{N(j)} \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4 \tag{4.22}$$

Note that all terms in the above equation are defined as previously.

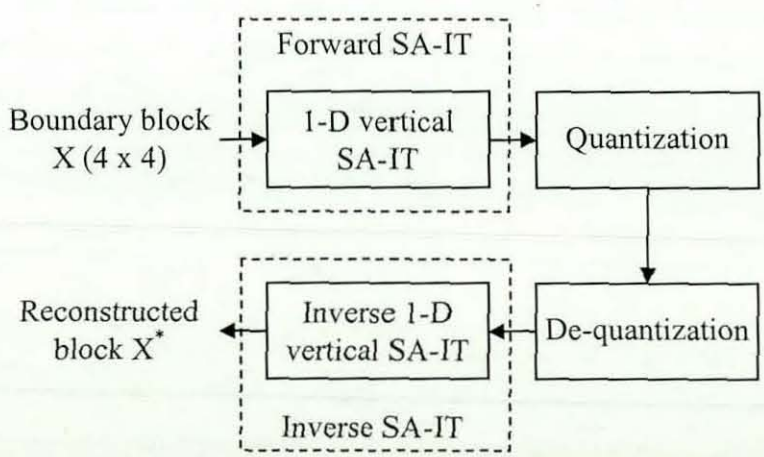


Figure 4-9 Flow diagram of 1-D vertical SA-IT and quantization

As a result of performing the vertical SA-IT, the resulting output (e.g., Figure 4-5) contains (i) an integer part, $W_j = C_{N(j)}X_j$; (ii) a non-integer part $E_{N(j)}$ which is integrated in the quantizer operation. The corresponding forward quantization formula is as follows:

$$Q_{ij} = round\left(W_{ij} \otimes E_{ij} / Qstep\right) \tag{4.23}$$

where W_{ij} and E_{ij} are the 1-D vertical transformed coefficient of the position (i, j) and the scaling factor of the coefficient respectively. Q_{ij} is the resulting quantized coefficient of the position (i, j). Q_{step} is defined as before. Note that the position (i, j) must be one of valid positions such as the ones highlighted in grey in Figure 4-5. Following the approach for quantization process design used in Section 4.4, the final quantization equation can be obtained as follows:

$$\begin{aligned} \left|Q_{ij}\right| &= \left(\left|W_{ij}\right| \cdot MF_{ij} + f\right) \gg qbits \\ sign\left(Q_{ij}\right) &= sign\left(W_{ij}\right) \end{aligned} \tag{4.24}$$

where MF_{ij} is the corresponding quantization factor of the position (i, j), which depends on the actual shape of the block. The *qbits*, f and the symbol $\gg$ are defined as before.

According to Equation (4.24) and the method for obtaining the de-quantization presented in Section 4.4 (i.e., for 2-D SA-IT), the rescaling equation for the 1-D SA-IT can be obtained as follows:

$$W_{ij}^{*} = \left(Q_{ij} \cdot RF_{ij}\right) \ll floor\left(QP / 6\right) \tag{4.25}$$

here RF_{ij} is the corresponding rescaling factor of the position (i, j), and W_{ij}^* is the corresponding de-quantized coefficient.

Finally, the equation for the inverse 1-D vertical SA-IT is similar to Equation (4.15), and can be presented as follows:

$$X_j^* = \left(C_{N(j)} W_j^*\right) \qquad 1 \le j \le 4, \quad 1 \le N(j) \le 4 \tag{4.26}$$

here, W^*_j and X^*_j are the results of de-quantization of column j and the reconstructed results of column j respectively. Note that the reconstructed results are divided by 64 as mentioned in Section 4.4.

Since the scaling factors $E_{N(j)}$ used in Equation (4.22) contain six scaling factors (i.e, 1, a, b, c, a^2, and d) which is a part of scaling factors used for 2-D SA-IT (see Table 4-1), the quantization factor MF, rescaling factor RF and the compensation value CV for 1-D SA-IT can use the same tables (Table 4-2, Table 4-3 and Table 4-4) as the 2-D SA-IT defined in Section 4.4.

For clarity, we use the same example described in Section 4.5 to illustrate the use of 1-D SA-IT and the associated quantization process that will be implemented in the proposed CODEC that will be discussed in Chapter 5.

Step 1: The forward 1-D vertical transform is performed by Equation (4.22). The resulting coefficients and scaling factors are illustrated in Figure 4-10 (same as Figure 4-5).

Step 2: The forward quantization is carried out by Equation (4.24). The resulting forward quantized coefficients are illustrated in Figure 4-11 (a). For example, the calculation of the coefficient of the position (1, 4) can be carried out as follows:

$i=$ \ $j=$	1	2	3	4
1		10	19	33
2			-11	14
3				1
4				-13

(a)

	1	a	a^2
		a	d
			a^2
			d

(b)

Figure 4-10 (a) Coefficients of vertical transform and (b) resulting scaling factors

$$MF_{14}=16384, \quad Q_{14}=\left(\left|W_{14}\right|\cdot MF_{14}+f\right)\gg qbits=\left(33\times 16384+10923\right)\gg 15=16$$

Step 3: The corresponding de-quantization is executed by Equation (4.25). The results are illustrated in Figure 4-11 (b). The rescaling result of the position (1, 4) can be calculated as follows:

$$RF_{14}=32, \qquad W_{14}^{*}=\left(Q_{14}\cdot RF_{14}\right)\ll floor\left(QP/6\right)=\left(16\times 32\right)\ll 0=512$$

$i=$ \ $j=$	1	2	3	4
1		10	13	16
2			-8	4
3				0
4				-4

(a)

	640	585	512
		-360	160
			0
			-160

(b)

Figure 4-11 (a) Forward quantized coefficients; (b) Inverse quantized coefficients

$i=$	$j=1$	2	3	4
1		640	225	592
2			945	752
3				272
4				432

(a)

10	4	9
	15	12
		4
		7

(b)

Figure 4-12 (a) Results of inverse vertical SA-IT; (b) final results

Step 4: The inverse vertical SA-IT is performed by Equation (4.26). The corresponding results are illustrated in Figure 4-12 (a). The results of the inverse transform of column $j = 3$, for example, can be obtained as follows:

$$N(3)=2,$$
$$X_3^* = \left(C_2 \cdot W_3^*\right) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} 585 \\ -360 \end{bmatrix} = \begin{bmatrix} 585-360 \\ 585+360 \end{bmatrix} = \begin{bmatrix} 225 \\ 945 \end{bmatrix}$$

Step 5: The resulting inverse transformed coefficients presented in Figure 4-12 (a) finally undergoes a 6-bit right shift operation to obtain the final results, presented in Figure 4-12 (b). The result of the position (1, 4), for instance, is obtained as follows:

$$X_{14}^{**} = \left(X_{14}^* + 32\right) \gg 6 = \left(592 + 32\right) \gg 6 = 9$$

Note that the constant 32 is used to minimize the error caused by the right-shift.

It is worthwhile to note that comparisons were made between the possible use of 1-D vertical and horizontal SA-ITs. The R-D performance results did not differ significantly. Thus, we adopted the vertical SA-IT in the proposed CODEC discussed in Chapter 5.

4.7 Preliminary Experiments & Results

Before fully incorporating the proposed SA-IT within an H.264 encoder, preliminary experiments were carried out to investigate its effectiveness in coding arbitrarily shaped objects and to compare its performance against SA-DCT. These experiments compare the average PSNR quality of the reconstructed boundary blocks (i.e. blocks which lie along the boundary of the object) of the video objects when using SA-DCT and the proposed 1-D and 2-D SA-ITs.

The video sequences "Foreman" and "Mother & Daughter" illustrated in Figure 4-13 (a) and Figure 4-13 (b) respectively were used for the experiments, assuming that the shapes of the foreground objects (i.e. of the Foreman, and Mother & Daughter) were known as alpha maps (described in Section 2.5). These are illustrated in Figure 4-13 (c) and Figure 4-13 (d). The same quantizer step size was used for all three transforms, i.e., SA-DCT, 1-D SA-IT and 2-D SA-IT.

First, each frame is segmented into two groups, namely, foreground (i.e., the objects) and background, using the associated alpha map information. Subsequently the frame is divided into 4 x 4 or 8 x 8 (for SA-DCT) blocks which will comprise of boundary (part of pixels that belong to the object), foreground (all pixels inside of the object) and background (all pixels outside of the object) blocks. Finally, the proposed SA-ITs and SA-DCT are applied to all boundary blocks separately. Note that the normal IT and DCT can be applied to all foreground object blocks while coding background object blocks are ignored as they do not belong to the foreground object. However for an effective comparison of the efficiency of SA-DCT vs. SA-ITs, experiments presented in this section were designed to compare the average PSNR values of only the boundary blocks.

Figure 4-14 and Figure 4-15 plot the variation of PSNR against QP for the "Mother & Daughter" and "Foreman" video objects respectively. It is clear from both figures that the curves of SA-ITs (either 1-D SA-IT or 2-D SA-IT) are very close to the SA-DCT. It means that the results of the new transforms are comparable with that of the

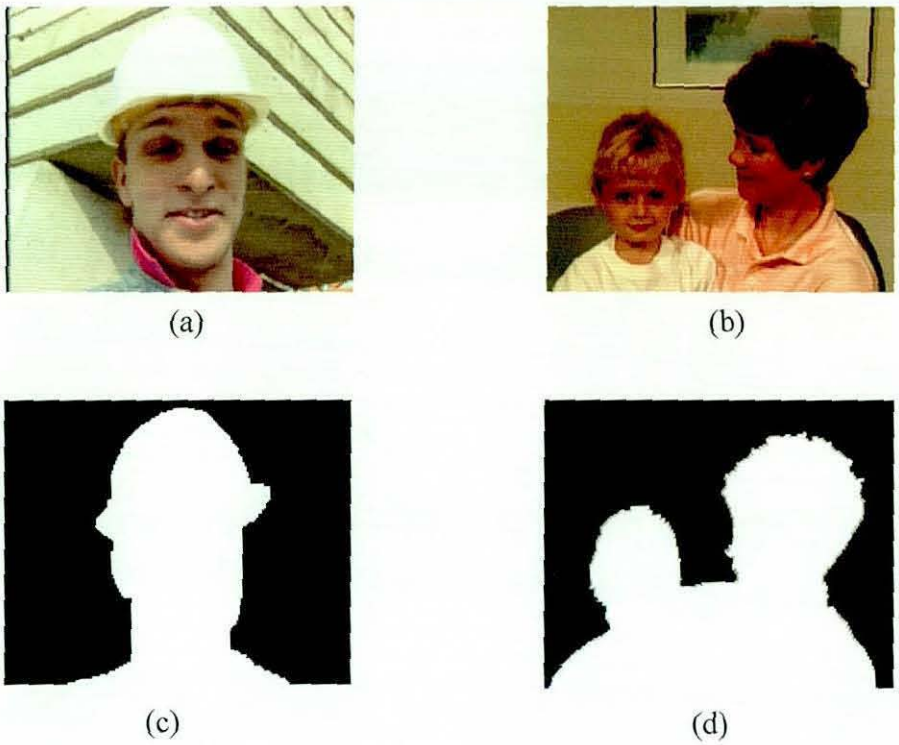


Figure 4-13 Original images: (a) Foreman; (b) Mother & Daughter. And associated alpha maps: (c) Foreman; (d) Mother & Daughter

SA-DCT traditionally used in coding arbitrarily shaped objects in the MPEG-4 standard. Figure 4-16 illustrates a resulting reconstructed frame of "Foreman" and "Mother & Daughter" video objects, when the proposed 1-D SA-IT was used in their coding. The results do not show any quality degradation, specifically any noticeable artefact in the object boundary areas.

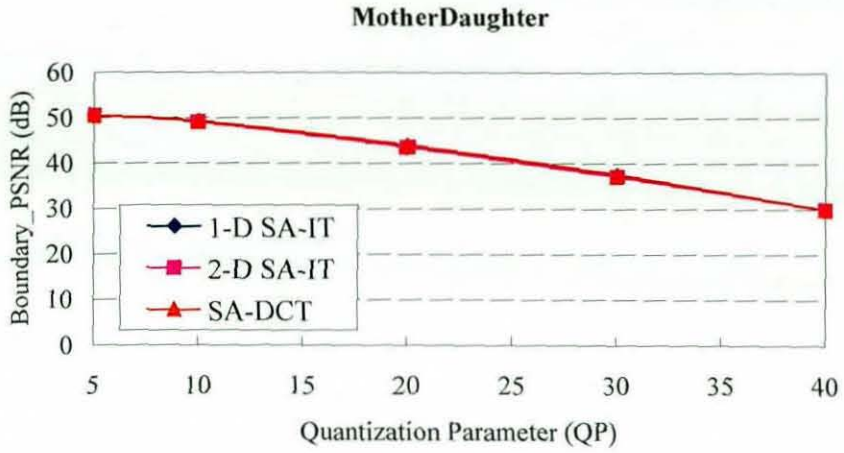


Figure 4-14 Comparison of SA-ITs and SA-DCT for Mother & Daughter.

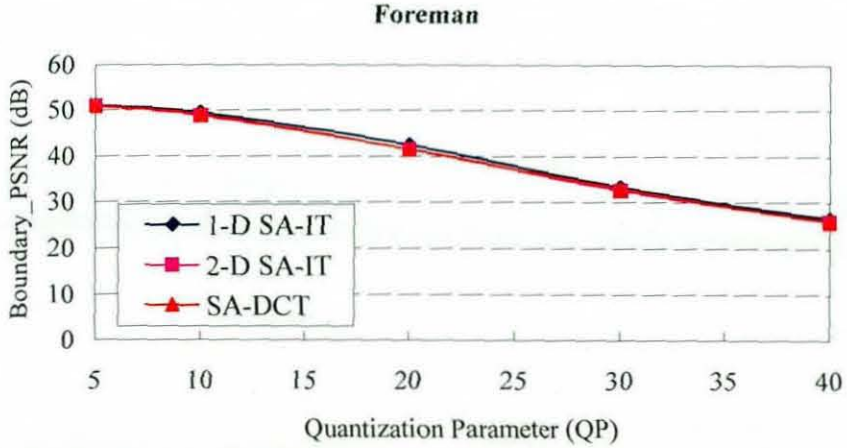


Figure 4-15 Comparison of SA-ITs and SA-DCT for Foreman

Figure 4-16 Resulting image of SA-IT for Foreman and Mother&Daughter objects

4.8 Conclusions

In this chapter, we have presented the theory of the proposed 2-D SA-IT and associated quantization procedures. We have shown that the new transform inherits the benefits from both IT used within H.264 and SA-DCT used within MPEG-4, which include: (i) allowing computation of the forward or inverse transform using simple additions and shifts, but no multiplications; (ii) minimizing computational complexity by using 16-bit arithmetic operations; (iii) avoiding divisions at quantization by the introduction of quantization table look-up strategy. (iv) supporting the coding of arbitrary shaped video object.

Moreover, we have presented the 1-D SA-IT to solve the sub-coefficients problem caused by the use of 2-D SA-IT. We have shown that the use of 1-D SA-IT not only delivers the ability to code arbitrarily shaped objects but also reduces the computational complexity as compared to the use of 2-D SA-IT for the same purpose.

We have also designed some preliminary experiments to compare the performance of the new transforms (1-D and 2-D SA-ITs) with that of the SA-DCT used in MPEG-4. The experimental results have proved that the proposed transforms have performance levels equivalent to that of SA-DCT.

In the following chapter we use the 1-D SA-IT theory developed within this chapter to introduce arbitrarily shaped object coding in the H.264 video coding standard.

Chapter 5 Object-Based H.264 CODEC

5.1 Introduction

The supporting techniques used in coding binary shape and transforms in support of the coding of the texture of arbitrarily shaped video objects have been investigated in Chapter 2 and Chapter 3. The aim of the novel transform, SA-IT, proposed in Chapter 4, is to meet the coding needs of the texture of boundary blocks of these video objects, under 16-bit integer arithmetic constraints.

This chapter presents the design, implementation and performance analysis of an object-based coding extension to the Baseline profile of H.264 standard. The basic idea is to adopt an object-based coding strategy similar to that of MPEG-4 Visual [5] discussed in Section 2.5, tailored to the specific operational and functional needs of H.264. Temporarily varying binary alpha maps (see Section 2.5.2) are used to temporarily vary the constitution of H.264 slice groups (see Section 2.3.4). These slice groups are in turn used to define video objects.

This chapter has been organized as follows. Section 5.2 gives an overview of the proposed idea of including object-based coding in H.264. Section 5.3 describes the detailed design of a novel binary shape coding strategy within both the H.264 encoder and the decoder. The texture coding of arbitrary shaped objects at macroblock layer level is introduced in Section 5.4. Simulation results for the proposed CODEC, and analysis and conclusions are presented in Section 5.5 and Section 5.6 respectively.

5.2 Overview of Object-Based H.264 CODEC

This section provides a functional overview of the proposed object-based H.264 video CODEC. A block diagram of its encoder is illustrated in Figure 5-1. The coding architecture of this encoder is detailed subsequently.

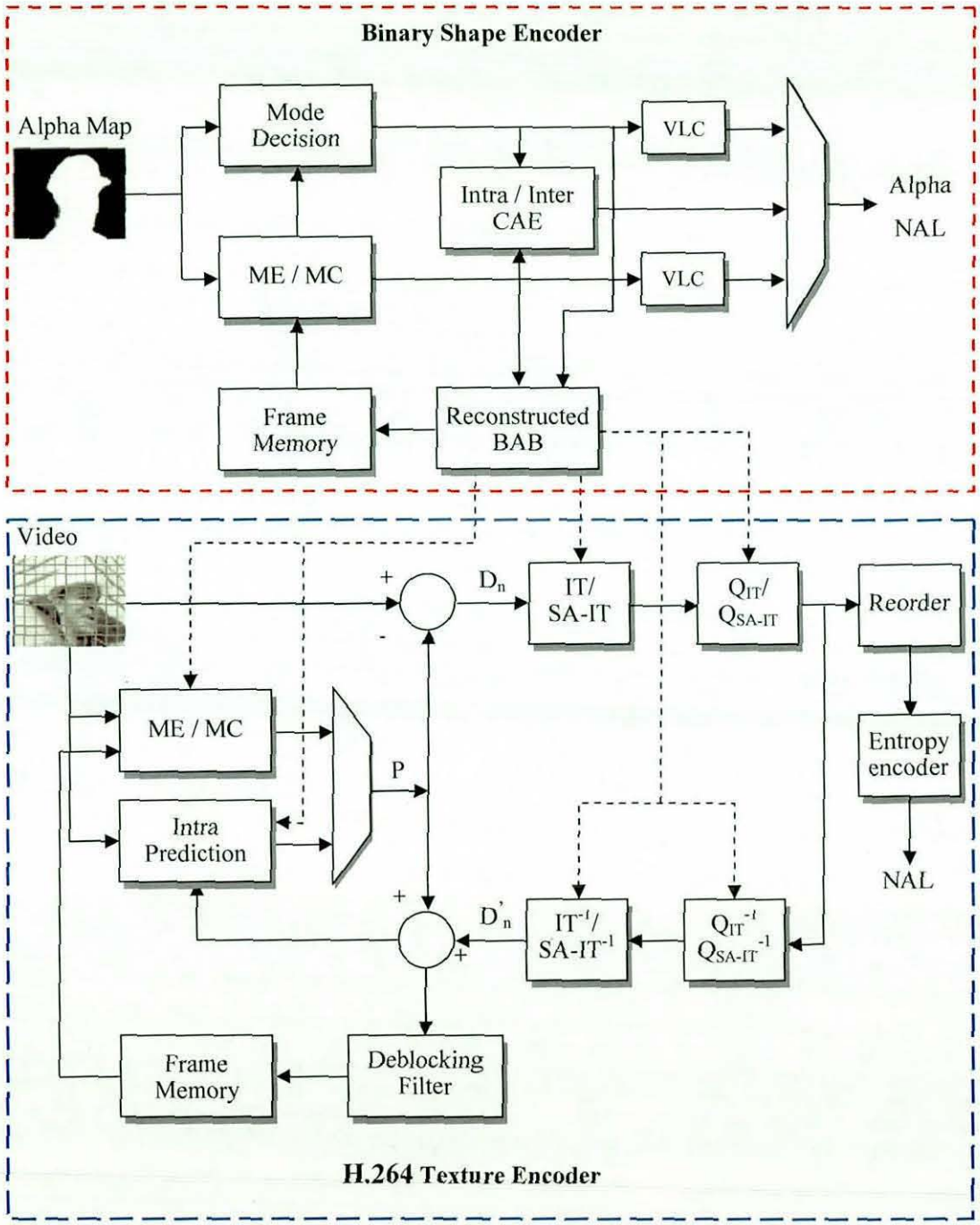


Figure 5-1 Block diagram of the proposed object-based H.264 encoder.

Object-based coding of video requires an initial stage of identifying suitable video objects or Regions-of-Interest (ROIs) within the video scenes. A number of existing computer vision based algorithms can be utilized to this effect [52] [53]. It is noted that our present research only focuses on the coding of these objects once they have been identified. Therefore the discussions are limited to this aspect.

It is noted that the shapes of arbitrarily shaped video objects are represented by so-called binary alpha maps and typically coded by a binary shape coder as illustrated by the top part of Figure 5-1 (red rectangle). The associated texture is coded by a texture coder as illustrated by the blue rectangle in Figure 5-1.

The introduction of an object-based coding architecture within the standardized frame based coding architecture of a H.264 CODEC, primarily requires the inclusion of the following:

1. **Binary Shape Coding** – exclusively used for coding binary alpha maps representing shape information of video objects.
2. **Shape Adaptive Integer Transform (SA-IT)** – a mathematical transform presented in Chapter 4, which is able to handle the border texture of an arbitrarily shaped video object.
3. **Modification to the Coding Architecture** – required for handling the temporal variations of slice groups driven by the binary alpha maps and inclusion of the extra alpha non-VCL NAL overhead to notify the presence of shape information of a coded object.

The block-based CAE (see Section 2.5.2.2) [20, 21, 54] of the MPEG-4 standard was selected as the basis for binary shape compression within the context of the proposed research. However, the binary shape compression technique used within MPEG-4 standard was exclusively designed to be used within the said standard, our detailed investigations revealed that it is not entirely suitable if used without suitable modifications within our research context. Therefore the original MPEG-4 shape coding approach was suitably modified and further improved in order to meet the

more challenging requirements of the proposed object-based H.264 CODEC (see Section 5.3).

The mathematical transforms relevant to the research context of this thesis, SA-DCT [24] and IT [17] were reviewed in Chapter 2 and Chapter 3. It was noted that SA-DCT can handle the border texture of an arbitrarily shaped video object. SA-DCT was chosen to provide the basis for the mathematical transform used within the proposed research context due to its block based nature/approach. However, SA-DCT based texture coding in MPEG-4, employs a floating point implementation. The transform used in H.264, IT, is essentially a 4 x 4 DCT transform implemented in integer arithmetic. Thus, the modification of a standard H.264 CODEC to an object-based CODEC requires the novel design of a SA-DCT that should be implemented in integer arithmetic. To achieve this, we have proposed SA-IT in Chapter 4. This transform will be used in the inclusion of object based coding functionality of the proposed object-based H.264 CODEC.

Further to the above, a number of other modifications and extensions have to be included in the proposed CODEC. It was mentioned that in H.264, a slice group can be used to independently (from the backgrounds and other slice groups) define and code a region of a video frame. Each slice group is made of a number of macroblocks and may represent an irregular (see Figure 5-2) or a rectangular region. However, in H.264, the macroblocks associated with a particular slice group are predefined by a user; the shape, size, and position of the slice group in relation to the frame does not vary between frames, i.e., temporally. In contrast, in the proposed scheme, the slice group(s) of each frame is defined by the temporally varying binary alpha maps. Thus, the slice group(s) becomes shape, size and position variant with respect to time. Furthermore, the boundary accuracy of an object defined by a slice group of H.264 is not accurate enough (minimal 4 x 4 block) for completed arbitrarily shaped object coding. However, in the proposed method, we modified the slice group definition (by binary alpha maps) to allow fully arbitrarily shaped object (accuracy in pixel by pixel) coding.

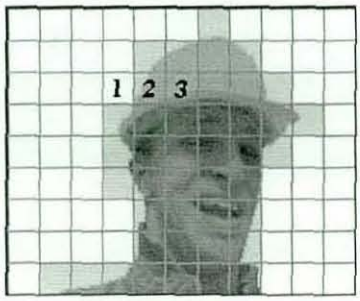


Figure 5-2 Foreman object is grouped into a slice group (grey), a small square indicates a macroblock of 16 x 16 pixels.

In addition to the above modification, due to the need of introducing binary shape coding within the proposed design, the texture coding of individual macroblocks is dependent on their corresponding shape information. Therefore the macroblock level texture coding process of MPEG-4, has to be considerably modified to be used within the present research context. The main modifications adopted within the proposed coding process are covered in Section 5.4

The shape information has to be effectively organized for efficient transmission and storage. In MPEG-4, the shape is coded and transmitted as a byte stream that is a part of the main bit stream which includes the video content, rather than as an independent stream. In contrast, non-VCL NAL units are to be utilized in H.264 for transmitting additional data, rather than being mixed with VCL information. It was mentioned in Section 2.3.1 that the NAL is designed for "network friendliness" and to enable simple and effective customization of the use of the VCL for a broad variety of systems. Therefore, in the proposed design, a new non-VCL NAL unit, named the *Alpha NAL* unit as illustrated in Figure 5-3 is defined to include and transmit the overhead shape information. In the first place, all coded binary alpha blocks in the video picture are encased into a stream with a shape header to form the binary shape data. The resulting binary shape data is then prefixed with a single byte long header data that indicates the type of *Raw Byte Stream Payload* (RBSP) data structure it contains. The RBSP is thereafter organized into a non-VCL Alpha NAL unit in byte-stream format together with a start code prefix that is a unique identifier

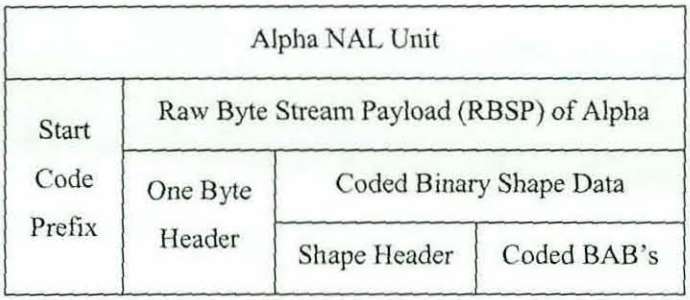


Figure 5-3 Structure of an alpha NAL unit.

of the start of a new NAL unit. The Alpha NAL unit is afterwards encapsulated to the NAL unit stream prior to the primary coded picture that consists of a set of VCL NAL units containing the main data of the video picture. Eventually, the NAL unit stream is stored or transmitted to the decoder. For more detailed information on NAL units, see [16].

All fundamental units of the proposed object-based coding scheme have been outlined above. The schematic framework illustrated in Figure 5-1 consists of a binary shape encoder and a H.264 texture encoder. It was mentioned that a moving arbitrarily shaped video object is entirely defined by its shape, motion and texture information. The object coding begins with the definition of its shape, i.e., the binary alpha map, which further guides its texture coding so that only the data within the object is encoded. The map is processed in units of 16 x 16 pixel blocks. The encoded shape information is sent to the decoder, and serves as a reference for motion estimation of the subsequent map and in the texture decoding. Subsequent to the shape encoder, the texture encoder commences its operation. Its basic work flow is the same as that of a standard H.264 encoder but its processes are guided by shape information as illustrated by the dashed arrowheads of Figure 5-1. Further a number of supplementary procedures have been added into the encoder such as padding techniques for transparent pixels in the reference block or picture. Section 5.4 provides detailed explanations of these procedures.

5.3 Binary Shape Coding Method

This section describes the design and implementation of the binary shape encoding and decoding mechanisms for coding object shape information in the proposed CODEC. A block diagram of the binary shape encoder is illustrated in Figure 5-1. Though similar in nature to shape coding used in MPEG-4 which has been introduced in Section 2.5.2, several modifications aimed at its adoptability and increased efficiency have been proposed within the context of our research when used within H.264.

5.3.1 Encoder Implementation

The first task in shape coding is the introduction of a user-defined parameter named as *ArbitraryShapedObject*, which is used as a switch to indicate the presence (or absence) of object based coding. This parameter is encoded and included within the sequence parameter set of the H.264 bitstream to form a part of the coded bitstream.

In MPEG-4, a video object within a binary alpha map is enclosed in a tightest fitting rectangular bounding box as depicted in Figure 2-7 (b) that consists of a number of Binary Alpha Blocks (BABs). These BABs are categorized into three classes: transparent, opaque and boundary. However as illustrated in Figure 5-2, the video object within the binary alpha map in H.264 is defined in a slice group whose shape need not be constrained to being rectangular, which is contrary to the object coding principles adopted within MPEG-4 (enclosed in a rectangular bounding box). Therefore in our design, only two categories of BABs are considered, the ones which are opaque and ones that lie on the boundary, as marked with '2' and '3' in Figure 5-2. We show later that the non existence of the transparent BABs in the proposed design, can significantly increase the shape coding efficiency.

Subsequently the binary shape encoder processes the BABs within the slice group on a macroblock-by-macroblock basis, in a raster scan order. During encoding, a BAB may be treated in one of six ways (as against seven ways in MPEG-4), as listed in

Table 5-1. It is the responsibility of the mode-decision block (shown in Figure 5-1) to choose an efficient encoding method for each BAB. In I-frames, only two of the above six modes are used (see Table 5-1). Opaque BABs of I-frames, are encoded using short variable-length codes (VLC). A boundary BAB of an intra-frame is coded using intra-CAE. In P-frames, shape is coded adopting a technique similar to that used by MPEG-4 in shape coding in Section 2.5.2. All six modes may be used for inter-coded BAB as listed in Table 5-1. However the partitioning of the candidate motion vectors from the corresponding texture block used in prediction is selected to be that used by H.264, rather than the one traditionally used by MPEG-4. This difference is clearly illustrated in Figure 5-4 when comparing with Figure 2-11. Note that transparent BABs in the proposed design need not be coded. The reason for that is explained in the next paragraph.

Table 5-1 BAB coding modes as represented by the *bab_type*

***bab_type* value**	**Type**	**Used in**
0	No update, without MVD	P-Frames
1	No update, with MVD	P-Frames
2	Opaque	I-, P-Frames
3	Intra CAE	I-, P-Frames
4	Inter CAE, without MVD	P-Frames
5	Inter CAE, with MVD	P-Frames

After all BABs of the slice group (i.e. opaque and boundary BABs) are encoded, a prefixed shape header needs to be appended to these coded BABs before encapsulating within the Alpha NAL unit as illustrated in Figure 5-3. It is worthwhile to note that the header of a VideoObjectPlane (VOP) in MPEG-4 (consists of four fields, 56 bits in total, see [20]) defines the position and size of the bounding box for motion compensation purposes. However, the header of a VOP in the proposed scheme has a distinct definition. To facilitate motion estimation, the proposed shape encoder estimates motion vectors for shape based directly on an absolute coordinate system, i.e., by regarding the dimensions of the entire frame as a bounding box. This

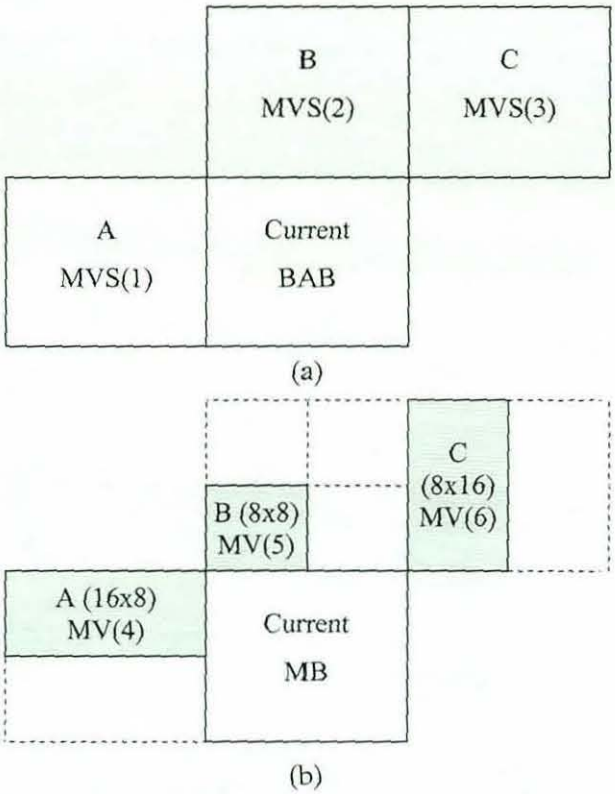


Figure 5-4 A list of candidate motion vectors are used for prediction. (a) MV for shape; (b) an example of MV for texture

is due to the fact that the accurate shape of the irregular slice group in the proposed design is not possible to be described by few fields as processed in MPEG-4. Therefore in the proposed design instead of defining the position and the size of a rectangular box bounding the object, a second binary map, called a BAB map as illustrated in Figure 5-5 is created, encoded and transmitted. The BAB map is made of all BABs of the binary alpha map and each pixel with magnitude '1' represents either an opaque or boundary BAB and a pixel with magnitude '0' represents a transparent BAB. The BAB map is then coded by a block-based intra CAE algorithm in a horizontal or vertical raster scan order. Finally, the resulting bit string is inserted as the header to the coded BABs to form the coded binary shape data as shown in Figure 5-3. Note that the BAB map is surrounded by a 2-pixel wide border prior to the intra CAE coding, and the pixel values of the border are assumed to be zero (transparent).

0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	0	0	0

Figure 5-5 BAB map of the Figure 5-2, each small square comprises 16 x 16 pixels

The number of bits required for coding a BAB map depends on the extent of local correlation of the video object in the BAB map and the size of frame resolution. Figure 5-6 and Figure 5-7 show the average bits per frame spent on the header and coded binary shape data (shape header and coded BABs) of the proposed shape encoder and MPEG-4 shape encoder individually for the video sequences of "News" and "Coastguard" having different resolutions, i.e., CIF and QCIF respectively. It shows clearly from these figures that, although the header size in CIF resolution coded by the proposed method is slightly higher than that of the MPEG-4 method, the proposed method successfully reduces the total number of bits required for coding binary shape data by around 25%, 22%, 38% and 26% respectively for video sequences, "News-CIF", "News-QICF", "Coastguard-CIF" and "Coastguard-QCIF". The reason for this improved performance is that no transparent BABs are required to be coded in the proposed method.

Eventually, after all shape information including coded BABs and the shape header are encoded, they are encapsulated as an Alpha NAL unit (see Figure 5-3) into the bitstream, which is subsequently stored or transmitted to the decoder.

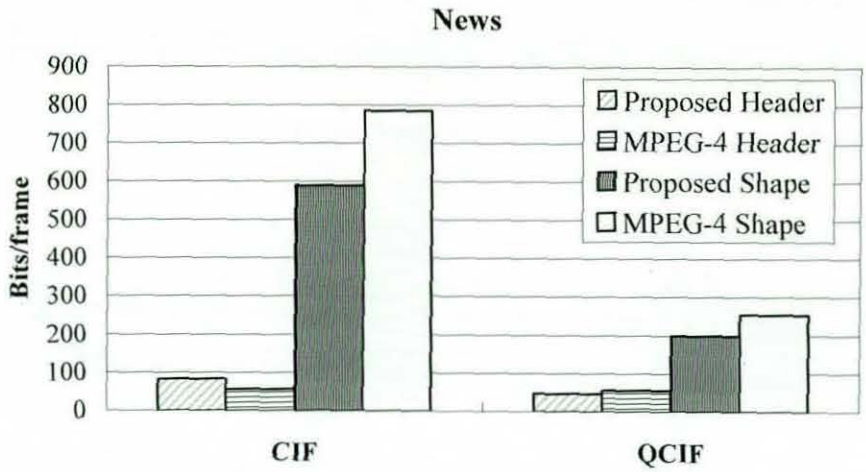


Figure 5-6 Shape coding results of News with resolutions CIF and QCIF

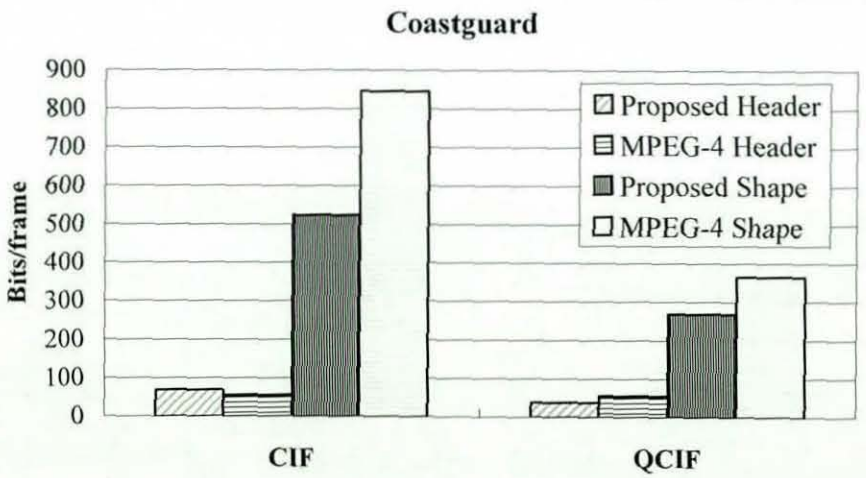


Figure 5-7 Shape coding results of Coastguard with resolutions CIF and QCIF

5.3.2 Decoder Implementation

Having discussed the binary shape encoder in detail, attention is now turned to the specific details of the shape decoding process of the proposed CODEC. Figure 5-8 illustrates the basic flow diagram of the decoder.

At the decoder of the proposed CODEC, subsequent to the decoding of the sequence parameter set (SPS) and the picture parameter set (PPS) NAL units, the Alpha NAL unit that consists of the coded video object shape information, is decoded.

As depicted in Figure 5-3, the first part in the coded binary shape data is the shape header which is decoded by using a CAE decoding algorithm. The result of decoding the header produces the BAB map which will indicate the absolute positions of the BABs yet to be decoded. Subsequently individual BABs are decoded from top-to-bottom and left-to-right in raster scan order. It has to be noted that the decoding procedure and algorithm for a given BAB used by the proposed scheme is similar, but yet not identical to that used in MPEG-4. It is explained briefly below with particular emphasis given to describing the differences when compared with that used by MPEG-4.

The first field of a BAB that is decoded is the "*bab_type*" that tells the decoder what coding mode has been decided by the encoder to be the most efficient for predicting a given BAB. As explained before in Table 5-1, we have considered the use of six coding modes within our encoder design. In Section 5.3.1, it was described that, for intra-coded BABs, only two modes (opaque and intra CAE) are considered whilst all six modes may be used for inter-coded BAB. The decoder employs VLC tables [22] to look up and decode the BAB type. Once the BAB type is decoded for a given intra-coded boundary BAB, its shape is decoded, by context-based arithmetic decoding. Inter-coded BABs are reconstructed using the received and decoded prediction errors and/or motion vectors and/or context-based arithmetic decoding (Intra CAE or Inter CAE). The candidate texture motion vectors that are used for predicting motion vector for shape information depend on partition sizes (see H.264

block partitions in Section 2.3.6) are available in the reference macroblock as shown in Figure 5-4. This is different to that adopted by MPEG-4 standard, which only uses 8 x 8 partitions. It is noted that for opaque BABs shape is known after decoding its BAB type. Transparent BABs need not be decoded in the proposed decoder.

After reconstructing the BAB shape entirely, it is ready for being used in texture decoding, i.e. for recovering the shapes of the coded texture blocks, and is discussed in the following section.

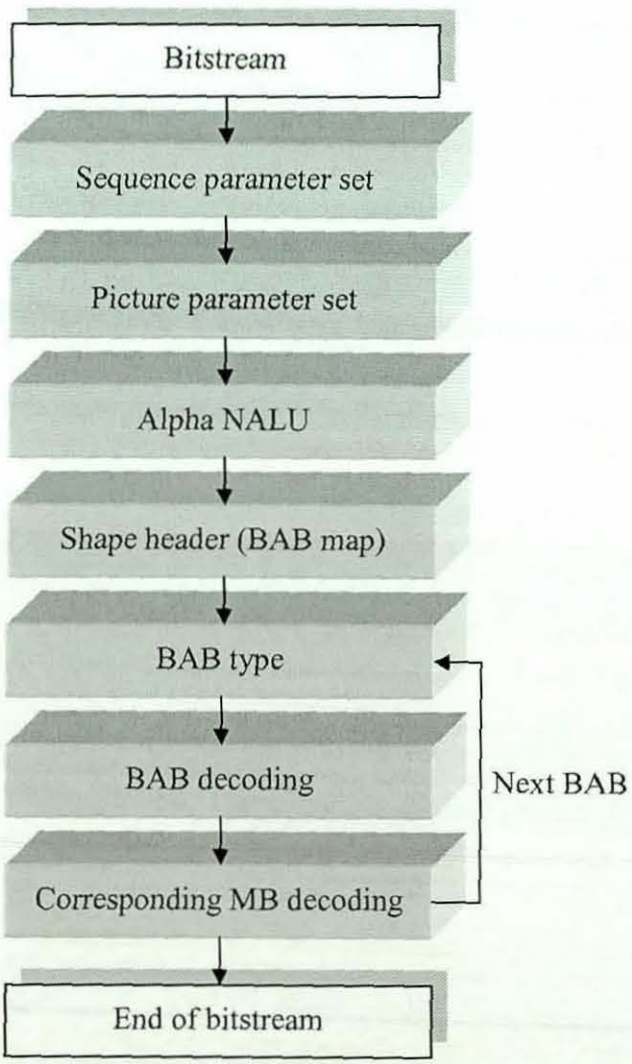


Figure 5-8 Basic processes flow of the proposed decoder

5.4 Macroblock Layer Texture CODEC

The methods adopted in macro-block level texture coding in H.264 are reasonably well known [14, 16]. However, the macroblock level texture coding we adopt in the proposed object-based H.264 CODEC has significant differences to the above standard technique, which relies to a great extent on the reconstructed shape information. This section highlights the main difference of the above algorithms. Where similarities can be mapped to the techniques adopted in either H.264 or MPEG-4 standards, appropriate references have been given without providing detailed explanations.

Fundamentally there are four differences between the macroblock level texture coding techniques adopted by the proposed technique and the technique adopted by H.264.

Firstly, in the proposed approach, in order to obtain a maximized coding efficiency, transparent macroblocks are not coded, which is similar to the approach used by MPEG-4. In the case where a macroblock contains one or more transparent partitions, all transparent partitions are not coded. Thus, the *Coded Block Pattern* (CBP-indicates which of the 4 x 4 or 8 x 8 blocks of a macroblock have non-zero transform coefficients) of the macroblock in the proposed approach is not only dependent on the transform coefficient levels but also on the number of transparent partitions within the macroblock. It is further noted here that for I-frame and P-frame macroblocks, the partitioning techniques are same as H.264.

Secondly, it was mentioned that in predicting the macroblock texture, there is significant use made of information associated with previously decoded blocks or macroblocks of the current frame or reference frames. In the case where reference blocks or macroblocks are transparent, then no information can be used for predictive purposes. Therefore we adopt an extended version of the technique used by MPEG-4, i.e., *repetitive block padding* [5] introduced in Section 3.2.1.2, to aid in the predictive coding of such blocks. Figure 5-9 shows the padding modes for boundary blocks. We

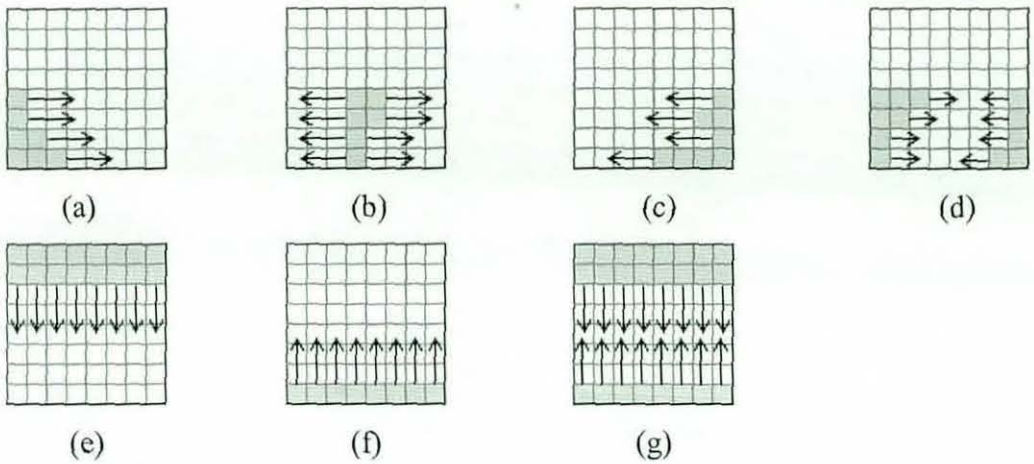


Figure 5-9 Padding modes for boundary blocks (8 x 8). (a) – (d) are horizontal padding and (e) – (g) are vertical padding after horizontal padding.

have introduced one additional padding mode as depicted in Figure 5-9 (b) to the original list of padding modes used in MPEG-4 in order to satisfy an arbitrarily shaped object's pixels in a block. In this mode, transparent pixels are filled with the value of the closest opaque pixel. It is note that in H.264, macroblock predictions are performed for both intra and inter coded frames, the block padding is introduced to intra and inter frame prediction. For intra prediction, the padding technique is applied to the current block's or macroblock's neighbouring blocks or macroblocks on the boundary. For example, Figure 5-10 shows a 4 x 4 luma block X that is required to be predicted. The samples (labeled A-M) are located in the nearby blocks B1, B2, B3 and B4, some samples are opaque pixel marked with grey such as C, G, H, K and L, and some are transparent marked with white. These transparent samples would be filled by using the above padding method before executing the prediction except the sample M which locates at a transparent block B2 that is not used for prediction. For inter prediction, the above padding approach is employed on boundary macroblocks at first followed by the extended padding [5] for transparent macroblocks. Figure 5-11 illustrates a reference frame after padding.

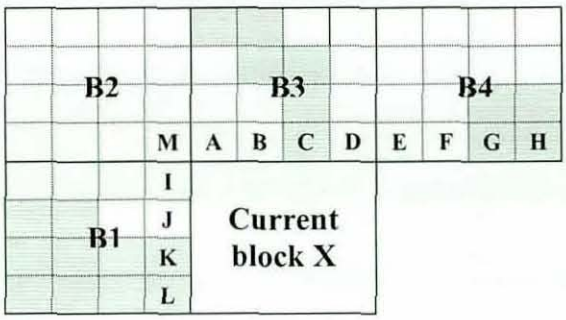


Figure 5-10 Example of neighbouring blocks of the current block X, opaque pixel given by grey.

Figure 5-11 Padding of a reference frame.

Thirdly, motion vector prediction is considered a part of texture coding. In H.264, the motion vector for a given partition is predicted from the neighbouring motion vectors of previously coded partitions as shown in Figure 5-4 (b). The presence of transparent partitions introduces the challenge of calculating the motion vector when a nearby partition is transparent. As a solution, we use *vector padding* [5] used in MPEG-4 to generate the vectors for the transparent partitions within a non-transparent macroblock in P-frame. It works in a manner similar to repetitive padding,

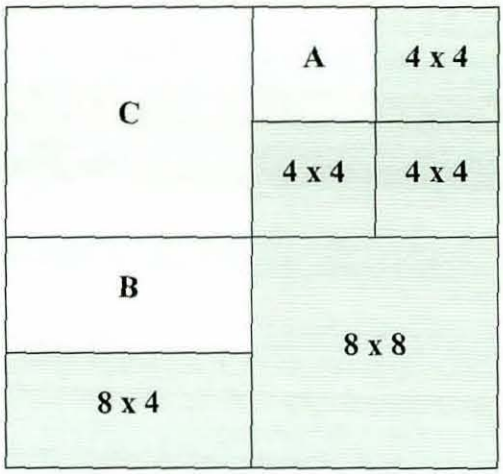


Figure 5-12 A macroblock with different partition size

i.e., the horizontal padding followed by the vertical padding. Since a macroblock in H.264 may have more than one partition, the padding follows the priority rule in horizontal, vertical and diagonal directions, and starts from the smallest transparent partition within the macroblock and ends with the larger partition. In Figure 5-12, the transparent partitions A, B and C are marked with white whereas non-transparent partitions are marked in grey. Partition A is first padded by the right partition (4 x 4) of A; Partition B is then padded by the bottom partition (8 x 4) of B; Finally partition C is filled with the value of A. The vectors after padding are used in the P-frame vector decoding and binary shape decoding as the texture candidate motion vectors mentioned in Section 5.3.2. If the reference partition is outside the picture or slice group or is part of a transparent macroblock, it is set to be unavailable.

Finally, as discussed in Chapter 4, SA-IT is used to encode texture of all intra or inter predicted 4 x 4 boundary blocks. However a complication arises from the fact that each transformed coefficient of two-dimensional SA-IT introduces so-called transformed sub-coefficients discussed at the end of Section 4.4. In addition, these sub-coefficients need to be transmitted after quantization for decoding purposes. The amount of the sub-coefficients depends on the actual shape within a block. Figure 5-13, for example, shows a part of an object (grey) within a 4 x 4 block. There are 35 transformed sub-coefficients in total generated by 2-D SA-IT in comparison with

only 11 transformed coefficients produced by SA-DCT. In this case, the transformed sub-coefficients could cause the texture bit rate to rise (depending actual shape) and an increased memory requirement such that the texture encoder becomes overly stressed. Thus, to solve this problem, we propose the use of 1-D vertical SA-IT (discussed in Section 4.6) and only suffer a very small penalty in the texture bit rate compared to 2-D SA-IT without sub-coefficients. Figure 5-14 shows the bitrate spent on 1-D SA-IT and 2-D SA-IT (with sub-coefficients and without sub-coefficients) on video sequences "Foreman", "News" and "Coastguard" respectively. Note that the 2-D SA-IT without sub-coefficients has been used only for comparison purpose and could not actually be decoded. It is obvious from the figure that 1-D SA-IT produces very close bitrates to 2-D SA-IT without sub-coefficients but a much lower bitrate than 2-D SA-IT with sub-coefficients. Note that, no transformations are required for the coding of transparent blocks, and opaque blocks using IT.

The statements above have highlighted the differences of coding texture in the presence of transparent blocks or macroblocks, when dealing with arbitrarily shaped objects. Further to the above changes required at the encoder side, similar changes are required at the decoder side, to handle the decoding of the objects. It is noted that decoder only contains inverse quantization and inverse SA-IT stages.

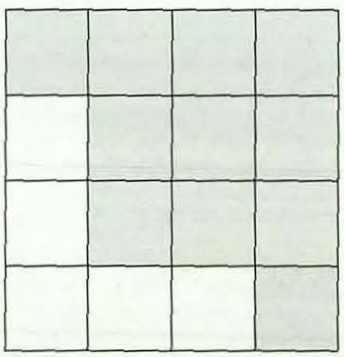


Figure 5-13 A 4 x 4 block of an object (grey)

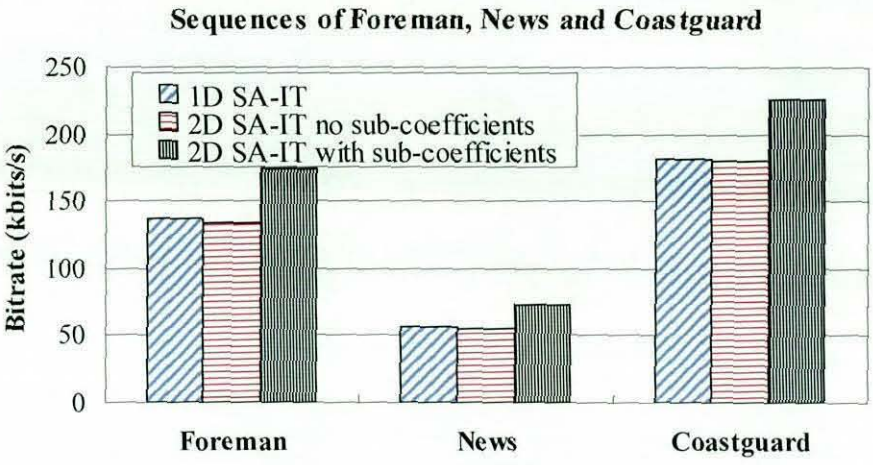


Figure 5-14 Bitrate comparison of 1-D SA-IT and 2-D SA-IT

5.5 Experimental Results

This section provides results of the experiments designed to evaluate the detailed performance of the proposed CODEC. We have used H.264 reference model software JM 10 [51], as a benchmark for comparison as well as to provide the initial implementation and operational platform for the proposed extensions. Rate-distortion graphs have been used for performance comparison, in which the rate has been calculated as the average number of bits required to encode a single frame, and the distortion has been measured using the combined-channel peak signal-to-noise ratio (PSNR) [20], which can be expressed as follows:

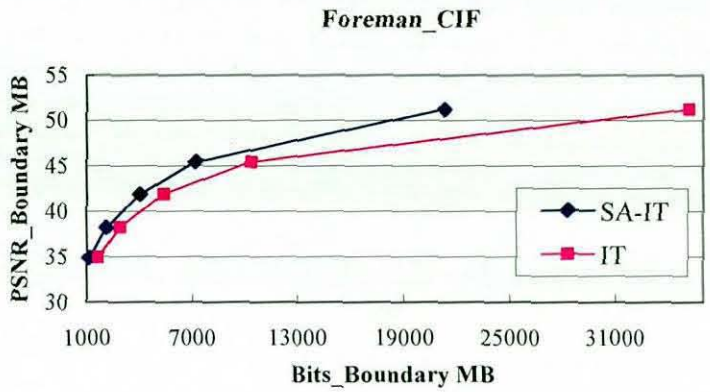
$$PSNR = 10\log\left(1.5 * M * N * (255)^2 / (SE_Y + SE_U + SE_V)\right) \tag{5.1}$$

where SE_Y, SE_U and SE_V are the squared errors for the three color components (YUV) respectively. Three video sequences (298 frames each); the "Foreman", "News" and

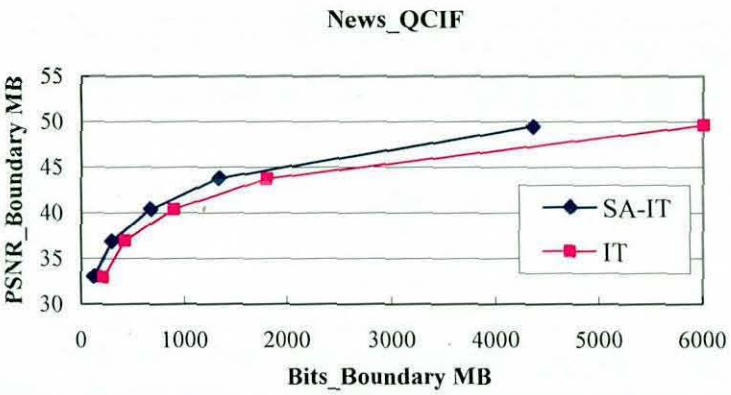
"Coastguard" were used for experimentation. Their selection was based on the presence different statistical variations of pixel values, due to the differences of number of objects in the scenes and their motion characteristics. Both the foreground and background of the "Foreman" sequence moves minimally while the foreground varies with time slightly in the "News" sequence. "Coastguard" possesses significant movements in the foreground and in the background areas. Further the camera used in capturing this video, shows panning motion. All test sequences are in 4:2:0 format and were coded at 25 fps. Figure 5-15 illustrates the objects of the test sequences to be coded. We assume that significant object shapes of foreground have been extracted into binary alpha maps for coding purposes, either manually, semi-automatically or automatically [52] [53]. In our experiments, binary masks previously used in the MPEG-4 standardization activities that are currently available in the public domain, were used.

(a) (b) (c)

Figure 5-15 (a) The ship object; (b) The foreman object; (c) The news man and girl object.



(a)



(b)

Figure 5-16 Rate-distortion diagram for object boundary of sequences Foreman (a) and News (b)

Two different experiments were designed to evaluate the performance of the proposed object-based H.264 CODEC and to compare it with the standard, H.264 CODEC, henceforth named as the non object-based coding algorithm. The results can be analyzed as follows.

The first experiment was designed to compare the rate distortion performances of the object-based and non object-based techniques when only considering the coding of the boundary macroblocks. Figure 5-16 illustrates the performance comparison of the proposed and standard H.264 techniques for the sequences of "News-QCIF" and

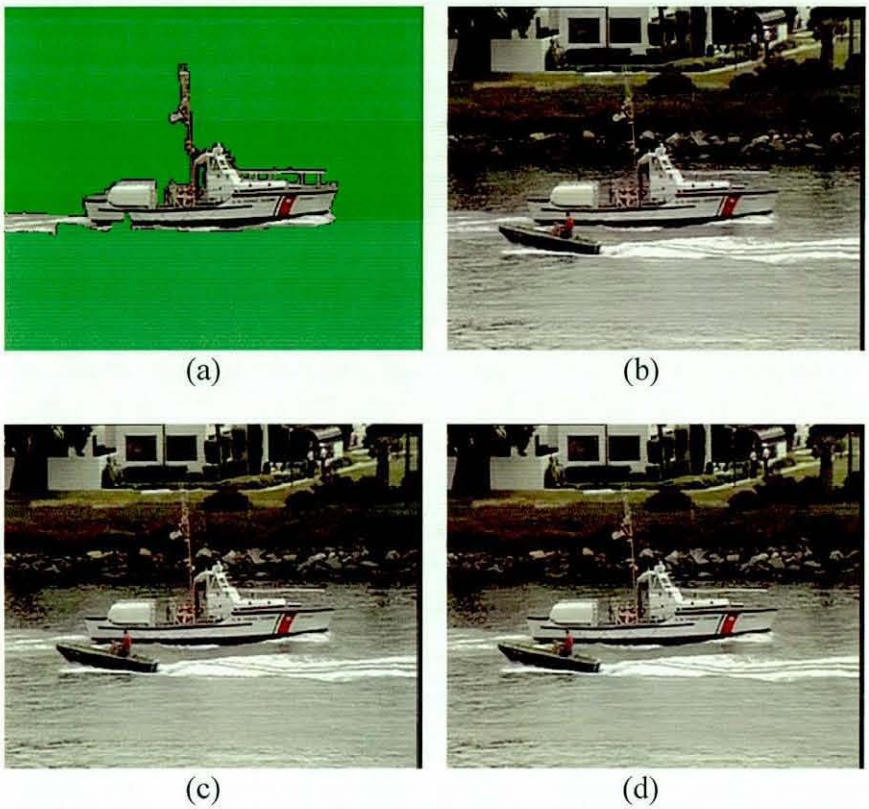
"Foreman-CIF" coded with five different QPs equaling 10, 20, 25, 30 and 35 respectively. Note that the R-D analysis is limited only to the boundary macroblocks of the object-based algorithm, and the corresponding macroblocks of the non object-based algorithm. In calculating the total bits spent by the boundary macroblocks of the object-based technique, the overhead that will be required for the shape coding of the blocks have been considered. Furthermore both sequences are coded without periodic intra refreshes, with the frames split into two slice groups, namely, foreground and background slice groups. The results illustrate that the proposed coding algorithm requires a lower amount of bits for coding the boundary macroblocks at any PSNR level, despite the need to code overhead shape information. The coding efficiency of the improved shape coding algorithm we have used and the efficiency of the SA-IT algorithm proposed are the main reasons for the above result. A further comparison of the results in Figure 5-16 (a) and (b) illustrate that the proposed algorithm performs at almost the same quality, i.e., PSNR, as the benchmark algorithm.

The second experiment was designed to evaluate the efficiency of the intended functionality of the proposed CODEC. As mentioned in Chapter 1, the main advantage/functionality of the proposed object-based coding scheme is the selective coding ability of the foreground (FG) and background (BG) areas. The algorithm provides means for coding the foreground at a better quality level as compared to the background. Table 5-2 compares the bitrate required for the QCIF "Coastguard" sequence when all but the first frame (which is coded as an I-frame), are coded as P-frames. When calculating the bit budget for the object-based coding algorithm, it is noted here that the overhead required for shape coding has been included. The coding of the benchmark algorithm has been performed using the quantization parameter QP=28 for all frames. When the proposed algorithm is used the background quantization parameter has been set at 51, 40, 35 and 30 whereas the foreground ('ship') QP has been held fixed at 28. The first experiment refers to a scenario where the background is not coded. Although the average $PSNR_{AVG}$ in the object-based coding is lower than non object-based coding because of the significantly higher quantization parameter used in coding the background, the quality of the foreground

area compares well with that of the non object-based coding case, with a significantly lower bitrate requirement. Table 5-2 shows that when QP=30 is used for quantizing the background information, the visual quality loss is not significant (see Figure 5-17 (c)), though the bitrate requirement has been reduced to 87% of the standard coding approach. It is further observed that no perceivable quality loss has been introduced in the contextually important foreground object when an acceptable loss of quality has been introduced to the background region while using QP=35 (see Figure 5-17 (b)). This choice of quantization parameters have resulted in a major saving of 50% of the bit budget. A weak result is observed in the case of the "News" sequence (QCIF) as shown in Table 5-3 and Figure 5-18. This is due to the almost static background and relative complexity of the shape.

Table 5-2 Coding results for sequence of Coastguard

Object-based Coding					
Q_{FG}	**Q_{BG}**	**$PSNR_{FG}$ (dB)**	**$PSNR_{BG}$ (dB)**	**$PSNR_{AVG}$ (dB)**	**Bitrate (Kbps)**
28	-	38.85	-	38.65	55.53
28	51	38.75	32.95	35.85	59.66
28	40	38.74	34.64	36.59	67.83
28	35	38.82	36.73	37.78	80.09
28	30	38.84	39.35	39.00	141.16
Non object-based Coding					
Q_{FG}	**Q_{BG}**	**$PSNR_{FG}$ (dB)**	**$PSNR_{BG}$ (dB)**	**$PSNR_{AV}$ (dB)**	**Bitrate (Kbps)**
28	28	38.93	41.03	39.98	161.67



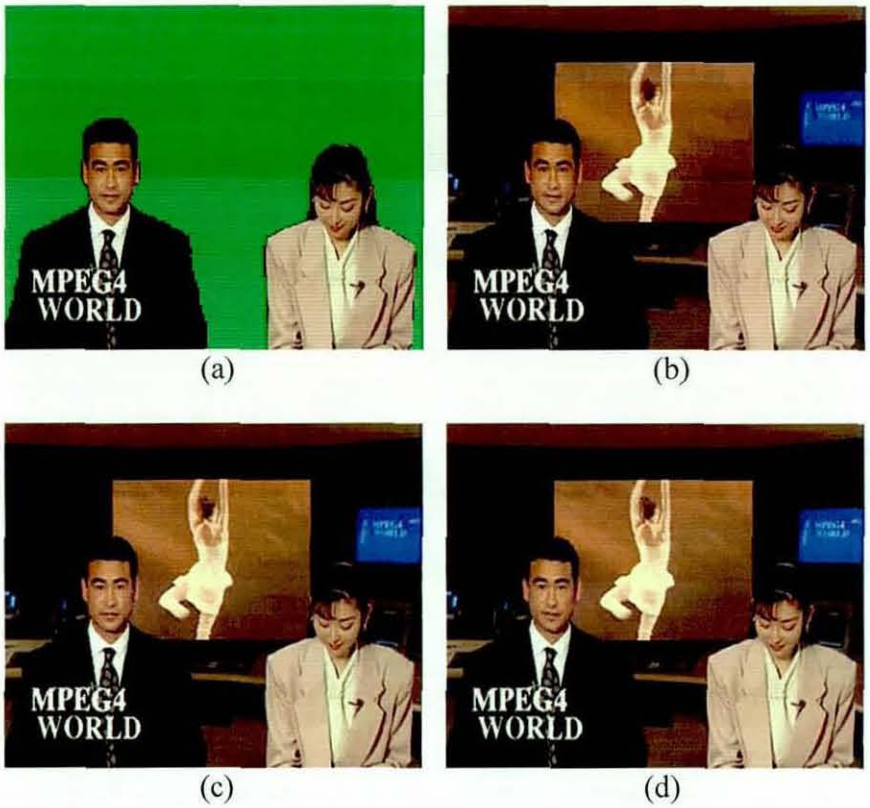
(a) (b)

(c) (d)

Figure 5-17 Coastguard (a) Only QP_{FG}=28; (b) QP_{FG}=28 and QP_{BG}=35; (c) QP_{FG}=28 and QP_{BG} =30; (d) QP_{FG}=QP_{BG}=28 coded with standard coding.

Table 5-3 Coding results for sequence of News

Object-based Coding					
Q_{FG}	**Q_{BG}**	**$PSNR_{FG}$ (dB)**	**$PSNR_{BG}$ (dB)**	**$PSNR_{AVG}$ (dB)**	**Bitrate (Kbps)**
28	-	39.02	-	39.02	22.90
28	51	39.05	29.05	34.05	27.06
28	40	39.04	32.77	35.90	34.52
28	35	39.04	35.22	37.13	42.48
28	30	39.03	38.01	38.52	57.01
Non object-based Coding					
Q_{FG}	**Q_{BG}**	**$PSNR_{FG}$ (dB)**	**$PSNR_{BG}$ (dB)**	**$PSNR_{AV}$ (dB)**	**Bitrate (Kbps)**
28	28	39.03	38.81	38.92	59.58



(a) (b)

(c) (d)

Figure 5-18 News (a) Only QP_{FG}=28; (b) QP_{FG}=28 and QP_{BG}=35; (c) QP_{FG}=28 and QP_{BG}=30; (d) QP_{FG}= QP_{BG}=28 coded with standard coding.

5.6 Conclusions

In this chapter, an object-based extension to H.264 video coding standard has been proposed. In order to facilitate object-based coding within H.264, the slice group structure has been modified, extended and used along with a novel design of a SA-IT that is capable of coding arbitrarily shaped boundary blocks. A novel shape coding algorithm based on the MPEG-4 shape coding methodology has been proposed which has been proven to be more efficient compared to that used within the MPEG-4 standard. We have shown that the object-based coding functionality provides the ability to selectively code images (video frames), enabling the ability to reconstruct important, pre-defined, foreground objects at high quality levels. Such flexibility is

of importance in applications such as security and surveillance, medical imaging, sports footage coverage etc. We have provided experimental results (both subjective and objective) and a detailed analysis to demonstrate the coding flexibilities and efficiency of the proposed algorithm as comparing with H.264.

We have further conceptually compared the functionality of the proposed object-based CODEC with that of Region of Interest (RoI) scalability of the upcoming H.264 Scalable Video Coding (SVC) extension [55, 56]. It has been revealed that H.264 SVC regions cannot specify arbitrarily shaped regions, but only regions made out of rectangular blocks.

Due to the successful design and implementation of the proposed object based H.264 CODEC, it is currently being considered as one of the proposals that can support the coding infrastructure of the DTI funded CRIMEVIS project that commences on 1st September 2007. This project is to further investigate the extension of the novel ideas to H.264 SVC standard.

Chapter 6 H.264 CODEC Analysis

6.1 Introduction

This chapter aims to determine the coding parameters of a H.264 CODEC that have a significant influence on its bit-rate, distortion, memory utilization and computational complexity. The thesis later aims to use these parameters in a multi-objective performance optimization of a H.264 CODEC (see Chapter 7).

H.264 [14, 16] (introduced in Section 2.3) provides high coding efficiency through added features and functionality. However, such features and functionality also entail additional resource consumption, i.e., computational complexity and memory usage. Specifically the cost effectiveness of a H.264 CODEC is affected directly by the computational complexity of the coding algorithms (such as motion estimation and compensation, transform and entropy coding) and their memory requirements. The efficiency of a coding algorithm is further dependent on various parameters used in defining the operational status of the CODEC at a given time. A number of investigations to this effect have already been carried out in literature [57-59]. However, these studies are only based on addressing either a single or two objectives at the encoder (either the computational complexity [58] or rate-distortion [59]) or the decoder end (memory utilization and computational complexity [57]). In this chapter, an analysis based on the H.264 baseline profile CODEC (JM 10) is carried out in order to find out the coding parameters which significantly affect the computational complexity, memory spending, rate and distortion performances. Based on this analysis, a multi-objective optimization framework for the H.264 CODEC is proposed in Chapter 7.

The rest of this chapter is organized as follows. A brief overview of the coding parameters used in our experiments is given in Section 6.2. The video test sequences used in our experiments are introduced and evaluated in Section 6.3. A

comprehensive analysis of the encoder is presented in Section 6.4, while Section 6.5 provides an analysis of the decoder. Finally Section 6.6 concludes this chapter.

6.2 Coding Parameters

For clarity, the coding parameters of the H.264 CODEC (JM 10) used in this analysis are summarised below:

- **Resolution:** Image width and height in luminance samples.
- **NumberReferenceFrames:** Sets maximum number of references stored in buffer for motion estimation and compensation.
- **UseFME:** Enable Fast motion estimation algorithms (0: disable, 1-3 enable).
- **SearchRange:** Sets allowable search range for motion estimation.
- **RDOptimization:** Enable rate distortion optimized mode decision.
- **Slicegroup:** Number of slice group to be used.
- **IntraPeroid:** Period of I-frames, i.e. frame will be coded using intra slices every IntraPeriod frames.
- **QP:** Sets quantization parameter value.
- **InterSearch4x4:** Enable 4 x 4 inter prediction & motion compensation.
- **InterSearch4x8:** Enable 4 x 8 inter prediction & motion compensation.
- **InterSearch8x4:** Enable 8 x 4 inter prediction & motion compensation.
- **InterSearch8x8:** Enable 8 x 8 inter prediction & motion compensation.
- **InterSearch8x16:** Enable 8 x 16 inter prediction & motion compensation.
- **InterSearch16x8:** Enable 16 x 8 inter prediction & motion compensation.
- **InterSearch16x16:** Enable 16 x 16 inter prediction & motion compensation.
- **Intra4x4ParDisable:** Disable intra 4 x 4 vertical & horizontal prediction modes.
- **Intra4x4DiagDisable:** Disable intra 4 x 4 diagonal down-Left and diagonal down-right prediction modes.
- **Intra4x4DirDisable:** Disable intra 4 x 4 vertical right, vertical left, horizontal down, and horizontal up prediction modes.

- **Intra16x16ParDisable:** Disable intra 16 x 16 vertical & horizontal prediction modes.
- **Intra16x16PlaneDisable:** Disable intra 16 x 16 plane prediction mode.
- **DisableThresholding:** Disable threshold of quantized coefficients that is used for discarding expensive coefficients. If after quantization there are only few small nonzero coefficients in a macroblock and the cost of coding these coefficients exceeds a fixed threshold, these coefficients are forced to zero [60].

The reasons for choosing the above parameters used in this analysis are given in the later sections (Section 6.4 and Section 6.5) of analysis.

6.3 Video Test Sequences

Since the computational complexity, bit rate and video quality performance greatly depend on the content of the source video, in this analysis 8 test sequences were chosen with distinct content and motion characteristics so that the results could reflect generality. The picture formats of the selected video clips include QCIF, CIF and SIF as introduced in Table 2-1. Each sequence is in 4:2:0 sampling format (see Section 2.2.2) and has 112 frames.

"Claire" and "Paris" are conversational video sequences with simple motion of the foreground and fixed background. Moderate movements in the foreground and slight movements in the background characterize the video sequences, "Foreman" and "Mobile". The sequences "Garden" and "Coastguard" show fast motion on both foreground and background regions. The most complicated motion characteristics are represented in "Football" and "Tennis" sequences. The above eight test sequences represent a wide range of videos with different properties and behaviours; from low to high detailed scenes, from moderate to high movement, from fixed to changing background. Therefore it is possible to systematically assess the performance of the proposed video CODEC through these test sequences. Figure 6-1 shows the 30^{th} frame of each video sequence.

(a) Claire (QCIF)

(b) Paris (CIF)

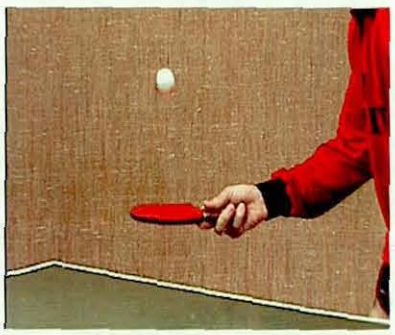
(c) Foreman (QCIF & CIF)

(d) Mobile (QCIF & CIF)

(e) Garden (SIF)

(f) Coastguard (QCIF & CIF)

(g) Football (SIF)



(h) Tennis (SIF)

Figure 6-1 Example frames of test video sequences

6.4 Encoder Estimation

6.4.1 Computational Complexity

The analysis of the encoder computational complexity is achieved by estimating the number of central processing unit (CPU) cycles required by it to perform key encoding functions. A CPU cycle (also referred as a clock tick) [61] is the smallest time unit recognized by a processor. The profiling results were generated with Intel's VTune Performance Analyzer, which enables the collection of run-time data indicating the number of cycles consumed by each function of the H.264 CODEC. This provides accurate information about processor utilization. The profiling tests were performed on a PC with a processor Intel P4-2800MHz to examine the computational complexity of the key functions of the encoder. The actual time spent on each function was measured in seconds using the following equation,

$$T_{function} = \frac{C_{function}}{H_{processor}} \tag{6.1}$$

where $C_{function}$ indicates the amount of cycles consumed by the function while the processor's frequency is given by $H_{processor}$. For example, 2800 MHz means 2800 million cycles per second. The total time spent on the function is presented by $T_{function}$.

The "Garden" video sequence was coded at 30 fps with QP = 28 and various motion estimation algorithms [62]. The percentage of processing time spent on the main functions and the total time consumed in encoding the video sequence are summarized in Table 6-1. In terms of H.264 encoding stages in Figure 2-3 (a), the main functions are grouped into the relevant encoding stages for evaluation of processor utilization in the table. The DCT, quantization, inverse DCT and inverse quantization processes are grouped together referred to as "Transform & Quantization". Other functions that do not belong to any encoding related step shown in Figure 2-3 (a) are grouped together as "Remaining functions".

From the results of Table 6-1 it is demonstrated that motion estimation and compensation consume more than half of the encoding time of the "Garden" video. Similar results were also obtained when analyzing the remaining test video sequences. The experimental results further demonstrate that motion estimation and compensation, reconstruction & store, intra prediction and transform & quantization are the most computationally expensive stages of the encoder. Moreover, the table reveals that fast motion estimation algorithms (see Section 2.3.6) UMHS, SUMHS and EPZS are much faster than FS. This is due to the fact that the fast motion estimation methods only search specified positions (instead of all search positions used in FS) and adopt early termination schemes [62]. Since the aim of this chapter is to seek coding parameters that significantly affect the total encoding time, we focus on identifying those coding parameters (listed in Table 6-2) that directly affect the above significant coding stages. From Table 6-2, we are only interested on those having over 10% influences on the processor utilization used for the CODEC optimization in Chapter 7.

Table 6-1 Profiling results of Garden video sequence

Coding stages	**Processor Utilization on various ME algorithms**			
	FS	**UMHS**	**SUMHS**	**EPZS**
Intra Prediction	2.55%	5.01%	4.82%	4.75%
Transform & Quantization	1.98%	3.43%	4.08%	3.24%
ME/MC	79.05%	58.63%	53.45%	58.41%
Reconstruction & Store	3.87%	7.34%	9.12%	7.37%
Deblocking Filter	0.55%	1.11%	1.19%	1.04%
Entropy	0.69%	1.60%	1.37%	1.22%
Remaining functions	11.31%	22.88%	25.97%	23.97%
Total coding time (Seconds)	210.527	104.941	88.246	110.034

Table 6-2 Coding parameters and coding conditions

Coding parameter	Range of value	Default
Resolution	QCIF, SIF, CIF	QCIF
NumberReferenceFrames	1 - 5	1
UseFME	0 - 3	0 (off)
SearchRange	16 - 32	16
RDOptimization	0 - 2	0 (off)
InterSearch4x4	0 - 1	1 (on)
InterSearch4x8	0 - 1	1 (on)
InterSearch8x4	0 - 1	1 (on)
InterSearch8x8	0 - 1	1 (on)
InterSearch8x16	0 - 1	1 (on)
InterSearch16x8	0 - 1	1 (on)
InterSearch16x16	0 - 1	1 (on)
Intra4x4ParDisable	0 - 1	0 (off)
Intra4x4DiagDisable	0 - 1	0 (off)
Intra4x4DirDisable	0 - 1	0 (off)
Intra16x16ParDisable	0 - 1	0 (off)
Intra16x16PlaneDisable	0 - 1	0 (off)

Table 6-3 Processor utilization at various video resolutions

	Processor utilization (seconds)	
	QCIF	CIF
Foreman	60.532	242.599
Mobile	60.271	239.612
Coastguard	60.626	241.415

In the following experiments, each video sequence was coded at a fixed frame rate of 30 fps, with QP = 28 while the coding parameters were varied within their full range (see Table 6-2). The resulting processor utilization is compared with the benchmark results, i.e., results obtained by coding each video sequence with the default coding parameters value (see Table 6-2).

A video sequence may be coded at different resolutions, e.g., QCIF and CIF. The experimental results in Table 6-3, illustrates that a video sequence coded at higher

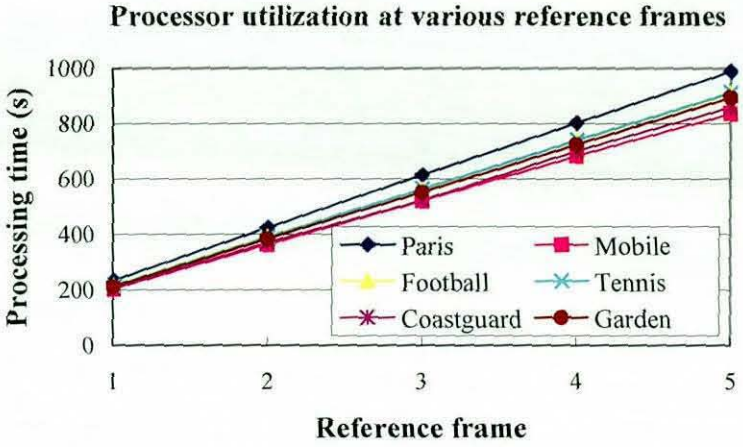


Figure 6-2 Processing time varies with different reference frames

resolution (e.g., CIF) requires much more time (around 4 times) to encode compared with the video sequence coded at lower resolution (e.g., QCIF). It is note that QCIF is a quarter of CIF resolution.

The variation in processing time when the number of reference frames (*NumberReferenceFrames*) is varied from 1 to 5, is shown in Figure 6-2. It can be observed that the extra processing time required for the addition of each additional reference frame is approximately 80% of the processing time required when one reference frame is used. This is due to the fact that the addition of each reference frame approximately doubles the number of comparisons required to complete motion estimation. This observation is true for all tested video sequences.

As mentioned earlier in this section, motion estimation and compensation is the most computationally complex process in the encoder. The motion estimation algorithms (see Section 2.3.6) are defined by setting the parameter *UseFME* (see Table 6-2) to 0, 1, 2 and 3 respectively in the H.264 encoder's profile file with '0' referring to the use of FS, i.e. disabling fast motion estimation. The results tabulated in Table 6-4 shows that FS on average requires 40% extra CPU time as compared to other fast search algorithms. On the other hand, SUMHS needs the least CPU time compared to FS,

UMHS and EPZS, while the CPU utilization performance of UMHS and EPZS algorithms are approximately similar.

Table 6-4 CPU time (in seconds) spent on ME algorithms

Video sequences	Motion estimation algorithms			
	FS	UMHS	SUMHS	EPZS
Claire	58.913	21.676	16.225	24.630
Foreman	60.703	27.872	23.038	29.371
Paris	233.487	98.660	82.321	111.260
Mobile	201.669	97.389	87.078	101.464
Football	214.694	117.402	99.894	119.987
Tennis	212.647	101.708	84.770	110.356
Coastguard	203.257	99.785	90.945	104.802
Garden	210.527	104.941	88.246	110.034

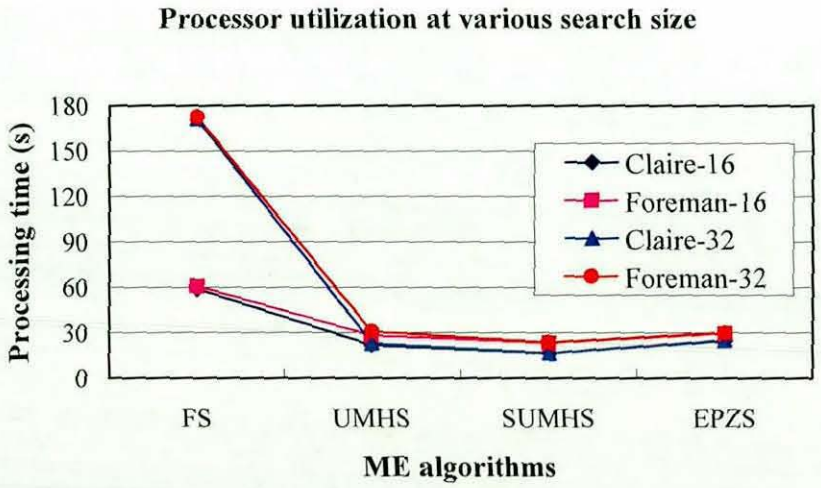


Figure 6-3 Processing time spent with various search size and ME algorithms

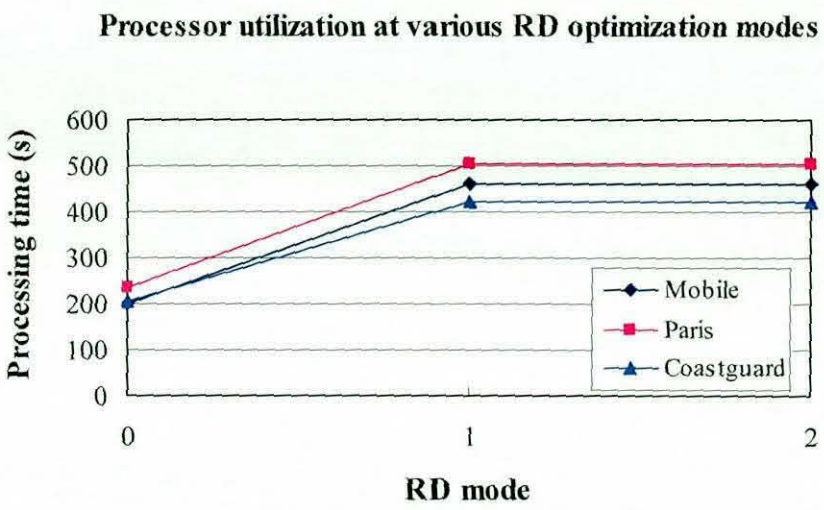


Figure 6-4 Processing time spent with various RD modes

For an inter prediction macroblock, the size of the search range (*SearchRange*) used by motion estimation algorithms has considerable impact on the processing time, as illustrated by Figure 6-3. It is specifically noted that the processing time for the FS algorithm is significantly high compared to the rest. Additionally, it is noted that for the FS algorithm, using a 32 x 32 pixels window size requires approximately three times more processing time when compared to a 16 x 16 window size. However, when utilizing the fast search algorithms, the processing times remain approximately similar. This is due to the fact that the FS loops over all search positions, whose number quadruples when the window size is doubled, both width and height-wise. For the fast motion estimation algorithms, the effect on processing power is significantly less since these algorithms only search specified positions.

The rate-distortion (R-D) optimization in H.264 is used to obtain the optimal coding mode, i.e., minimal rate and maximal quality, for a macroblock, block or partition. However, the use of R-D optimization in video coding comes with a penalty, i.e. an increase in processing cost. Figure 6-4 illustrates that a video sequence coded without R-D optimization (*RDOptimization*) mode marked as "0" requires much less (over

50% less) processor utilization as compared to video coded with RD modes marked as"1" and "2" respectively. It is noted that RD mode '0' signifies the non use of RD optimization and the other two modes refers to two different approaches to RD optimization [51]. The additional processor utilization observed in Figure 6-4 is due to the need of having to compute RD cost for each of the prediction modes when using modes '1' and '2'. However, mode '0' only requires the calculation of the cost for the best prediction mode.

In the H.264 encoder, a macroblock is coded either as a 16 x 16 pixel area or it is further sub-partitioned. There exist several parameters (see Table 6-2) within the encoder's profile file used for managing the sub-partition modes, which may have an influence on the computational complexity. The results tabulated in Table 6-5 show that the individual partition modes have a very slight effect on processing time, i.e. less than 10% computational cost reduction when switching off a single mode.

Table 6-5 Processing time (second) varies with different partition sizes

Partition mode	**Video sequence**				
Inter frame	**Paris**	**Foreman**	**Football**	**Mobile**	**Garden**
All modes are on	233.487	60.703	214.694	201.669	210.527
4 x 4 mode off	220.259	57.101	202.416	190.980	200.087
4 x 8 mode off	224.225	58.134	205.891	194.440	206.114
8 x 4 mode off	224.642	58.216	205.731	194.208	202.913
8 x 8 mode off	226.544	58.750	207.366	196.034	205.864
8 x 16 mode off	227.890	58.889	208.275	196.767	206.737
16 x 8 mode off	228.083	59.018	208.374	196.980	206.634
16 x 16 mode off	228.019	58.904	207.982	196.596	206.478
Intra frame	**Paris**	**Foreman**	**Football**	**Mobile**	**Garden**
All modes are on	233.487	60.703	214.694	201.669	210.527
4 x 4 parallel off	233.209	60.330	213.870	200.639	210.003
4 x 4 45° diagonal off	233.096	60.396	213.959	200.803	210.489
4 x 4 other diagonal off	232.440	60.216	213.381	200.951	210.426
16 x 16 parallel off	232.352	60.330	214.248	200.203	210.543
16 x 16 plane off	233.088	60.533	214.759	200.762	210.788

From the above analysis of computational complexity of the H.264 encoder, a number of parameters that have a significant impact on computational complexity (those that result in more than 10% influences) have been identified and follow for the latter use within the multi-objective optimization framework proposed in Chapter 7; they are, *Resolution*, *NumberReferenceFrames*, *RDOptimization*, *UseFME* and *SearchRange*.

6.4.2 Memory Utilization

Memory utilization of the H.264 encoder may be classified into two groups namely; (i) temporary memory, which is the memory allocated to local variables within the life span of a function call, i.e., allocates and frees memory within a function or block; (ii) global memory, which is the memory allocated to global variables that are used during the entire encoding process. The global memory required by the encoder can be further partitioned into two categories, namely; (a) dynamic memory, memory allocated to variable encoding parameters such as the number of reference frames, picture resolution and so on; (b) constant memory, memory allocated for fixed data such as quantization tables, intra-prediction probability tables, variable-length encoding tables and some other small constant tables. In this section we only focus on dynamic memory allocation since it has the potential to be optimized through the use of more appropriate encoding parameters.

The approach we used for identifying encoding parameters that significantly impact dynamic memory allocation (those that results in over 10% share) was the use of "*malloc*" and "*calloc*" functions within the 'C' program codes. Table 6-6 presents a summary of the formulas used to calculate the memory required at different buffer levels ("Other" refers to the memory that does not belong to any buffer level). It is evident that there are only 5 coding parameters that influence dynamic memory allocation. The first parameter is *Resolution* of a video sequence represented by w, the width of the picture and h, the height of the picture. Further parameters are, the number of reference frames (n, *NumberReferenceFrames*), the number of slice group (*sg*, *SliceGroup*), the size of search range (s, *SearchRange*), and the use of motion

estimation algorithms (*fme*, *UseFME*). As expected, frame resolution has a significant impact on storage requirements. Table 6-7 shows how memory requirements vary with the above coding parameters. The first row marked in grey in the table is used as a benchmark for comparison. A number of observations can be made using the results tabulated. A video with CIF resolution requires almost 4 times more memory than a video with QCIF resolution. The use of two reference frames against one, with a fixed window size of 16-pixels and QCIF resolution, requires 25% more memory. Further the use of a 32-pixels search window against a 16-pixels search window, with fixed resolution and number of reference frames, requires 35% extra memory. When the slice group number is incremented to 3, 13% more memory is needed. However it is observed that when benefiting from fast motion estimation (*fme*), 12% less memory will be required, when keeping all other parameters at default value.

Table 6-6 Memory requirement formulas for various buffer levels

Buffer level	**Formula (in Bytes)**
Frame	$(((153+89*n)*w*h)\gg 3)+(48+32*n)*(w+8)*(h+8)$
Slice	$sg*(63+3137*(w*h\gg 7))$
MB	$(255*w*h)\gg 7$
Prediction	$fme=0?(9*(w*h)\gg 4)+2600*n+(1024*n+8)*(2*s+1)^2$ $:(9*(w*h)\gg 4)+2560*n+8*(2*s+1)^2$
Entropy	$96+3*((w*h)\gg 3)$
FME mode	$fme=1:\ 49+(2*s+1)^2+9*((w*h)\gg 1)+576*n$ $2:\ 49+9*((w*h)\gg 1)$ $3:\ 12+(w*h)\gg 1+(2*s+1)^2+(168*n+42)*w$
RD mode	$406+343*((w*h)\gg 1)$
Other	$2048+73*((w*h)\gg 8)$

Note: formulae above are taken from the memory allocations in the JM software. *n*: number of reference frames, *w*: width of frame, *h*: height of frame, *sg*: slice group, *s*: search range, *fme*: fast motion estimation.

Table 6-7 Memory requirements in Mbytes

Resolution	Ref. frames	Search Range	FME	Slice group	Memory requirement
QCIF	1	16	0	1	8.76
CIF	1	16	0	1	31.39
QCIF	2	16	0	1	10.95
QCIF	1	32	0	1	11.84
QCIF	1	16	1	1	7.8
QCIF	1	16	0	3	9.94

6.4.3 Rate and Distortion Analysis

For a given video sequence, rate (i.e., bit rate, measured in Kbits/s) and distortion (i.e., quality, PSNR measured in dB) are sensitive to coding parameters. Modifying coding parameters such as using optional coding modes or choosing different R-D optimization algorithms will affect the output of the encoder, resulting in different levels of bit rate and visual quality. The coding parameters (based on baseline profile) affecting the rate and distortion are tabulated in Table 6-8. Experiments were performed in order to find out those parameters that can significantly influence the bit rate ($\geq 10\%$) and quality (≥ 0.2 dB). It is noted that in the experiments performed, all video sequences were 30 fps. The benchmark experiment is denoted by a grey highlight in each table (6 tables).

It is clear that a high resolution video sequence (e.g., CIF) requires much more bits for storage or bandwidth for transmission, than a lower resolution video (e.g., QCIF). The test sequences in Table 6-9 were coded at QCIF and CIF resolutions with identical encoding conditions. As expected, the rates of CIF sequences are much higher than QCIF sequences. It is also clear from the table that more than 0.2 dB PSNR is gained when CIF resolution is used.

Table 6-8 Candidates for parameter selection

Coding parameter	Range of value	Default
Resolution	QCIF, SIF, CIF	QCIF
NumberReferenceFrames	1 - 5	1
UseFME	0 - 3	0 (off)
SearchRange	16 - 32	16
RDOptimization	0 - 2	0 (off)
Slicegroup	1 – 5	1
IntraPeriod	0 – 15	0
QP	10 - 41	28
DisableThresholding	0 – 1	0 (off)
InterSearch4x4	0 – 1	1 (on)
InterSearch4x8	0 – 1	1 (on)
InterSearch8x4	0 – 1	1 (on)
InterSearch8x8	0 – 1	1 (on)
InterSearch8x16	0 – 1	1 (on)
InterSearch16x8	0 – 1	1 (on)
InterSearch16x16	0 – 1	1 (on)
Intra4x4ParDisable	0 – 1	0 (off)
Intra4x4DiagDisable	0 – 1	0 (off)
Intra4x4DirDisable	0 – 1	0 (off)
Intra16x16ParDisable	0 – 1	0 (off)
Intra16x16PlaneDisable	0 – 1	0 (off)

Table 6-9 Rate and distortion varies with resolution

Resolution	Video sequences					
	Foreman		Coastguard		Mobile	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
QCIF	129.20	38.49	258.68	40.36	516.84	34.32
CIF	414.60	38.80	1387.88	40.62	2115.61	35.25

The aim of multiple reference pictures supported in H.264 is to improve quality or reduce bit rate by the better prediction gained from the enhanced reference picture selection. However, the computational complexity (in Section 6.4.1) and memory (in Section 6.4.2) expense is significantly high. The bit rate and PSNR variations when the number of reference frames is varied from 1 to 5 are tabulated in Table 6-10. It is noted that for all test videos, the bit rate required marginally reduces (with quality approximately held constant) when a higher number of reference frames are used.

Further experiments were performed to analyze the effect of using different motion estimation algorithms on reconstructed video quality and bit rate. The results in Table 6-11 show that only 1-2% variation in bit rate and only 0.01 – 0.06 dB variation in quality were observed. This is expected as the motion estimation algorithms included in the H.264 reference software are the most efficient techniques that give reasonable results close to FS [62]. The actual enhancement possible via the use of one of the three fast search algorithms is that the considerable reduction of time taken for ME (see Section 6.4.1).

Table 6-10 Rate and distortion varies with reference pictures

Reference frame	Video sequences (QCIF)					
	Claire		Coastguard		Foreman	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
1	35.40	40.59	258.68	40.36	129.20	38.49
2	34.65	40.59	250.48	40.39	119.86	38.50
3	34.56	40.58	250.15	40.39	119.64	38.54
4	34.84	40.58	248.74	40.38	119.38	38.55
5	34.67	40.57	248.62	40.41	118.87	38.55
Reference frame	**Video sequences (SIF)**					
	Football		Garden		Tennis	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
1	1977.85	35.72	2445.59	34.34	795.09	37.24
2	1927.27	35.73	2399.32	34.34	762.37	37.22
3	1915.80	35.75	2384.89	34.35	749.54	37.23
4	1911.49	35.75	2365.22	34.35	739.06	37.23
5	1910.75	35.76	2329.23	34.36	732.69	37.23

Table 6-11 Rate and distortion varies with ME algorithms

ME algorithm	Video sequences (QCIF)					
	Claire		Coastguard		Foreman	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
FS	35.40	40.59	258.68	40.36	129.20	38.49
UMHS	35.37	40.50	257.66	40.34	126.04	38.44
SUMHS	35.32	40.51	258.23	40.31	126.58	38.44
EPZS	35.01	40.53	259.12	40.32	127.17	38.48
ME algorithm	**Video sequences (SIF)**					
	Football		Garden		Tennis	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
FS	1977.85	35.72	2445.59	34.34	795.09	37.24
UMHS	1958.06	35.71	2440.80	34.33	796.64	37.24
SUMHS	1955.20	35.70	2444.92	34.33	798.69	37.22
EPZS	1954.33	35.73	2443.18	34.34	795.12	37.24

Table 6-12 indicates the variations in bit rate and video quality when the video is coded with different search window sizes, number of slice groups and rate-distortion optimization modes. It is clear from the results that the variations in rate and quality are minimal when extending the search window size from 16 to 32 pixels. This can be due to the fact that a 16 pixel search window is of sufficient size to capture the motion in all test videos. Moreover, the results show that the use of a larger search window can not always guarantee a better prediction as seen in the sequence "Claire". It is noted that the advantage of being able to find a better match when the search window size is enlarged is offset by the need of transmitting potentially larger motion vectors, that can affect the R-D performance negatively. It is further seen that the bit rate increases (less than 5%) with the increase of the number of slice groups since the encoder requires a small amount of bits to encode the header of each slice group. However, image quality does not improve when more slice groups are used. The use of rate-distortion optimization modes produces 5%-8% decrease in rate but with slight penalty (0.06dB-0.17dB) in image degradation.

Table 6-13 tabulates the R-D performance data for test sequences "Claire", "Coastguard" and "Foreman", when four different intra refresh rates, i.e., 1:0 (not any periodic intra refreshes except the first frame), 1:10 (one in every ten frames is encoded in intra mode), 1:5 and 1:1 (every frame is encoded as I-frame) were used. As the intra refresh rate increases, it can be seen that the bit rate increases (over 10%) and distortion decreases (over 0.2 dB) when compared to the results of the benchmark experiment. QP is the encoding parameter that has the most significant effect on the RD performance of a video CODEC. The results in Table 6-13 clearly support the above statement. In "Mobile" sequence, for example, the rate is decreased approximately 100% and the PSNR quality is decreased approximately by 3.73dB when the QP is increased from 28 to 32. When the threshold of coefficients (*DisableThresholding*, see Section 6.2) is disabled (set to "1"), the bit rates of videos; "Paris", "Garden" and "Foreman" increase by 13%, 5.6% and 28% respectively and the PSNR improves by 0.48dB, 0.5dB and 0.57dB respectively.

Table 6-12 R-D varies with search range, slice group and RD mode respectively

Search range	**Video sequences**					
	Claire (QCIF)		**Coastguard (CIF)**		**Foreman (QCIF)**	
	Rate	**PSNR**	**Rate**	**PSNR**	**Rate**	**PSNR**
16	35.40	40.59	1387.88	40.62	129.20	38.49
32	35.41	40.58	1386.97	40.60	128.89	38.51
48	35.43	40.57	1387.11	40.59	128.79	38.50
Slice group	**Video sequences (SIF)**					
	Mobile		**Garden**		**Paris**	
	Rate	**PSNR**	**Rate**	**PSNR**	**Rate**	**PSNR**
1	2115.61	35.72	2445.59	34.34	608.55	37.73
2	2159.54	35.73	2468.70	34.33	626.68	37.73
3	2161.55	35.72	2472.80	34.34	628.06	37.73
4	2164.72	35.72	2477.50	34.34	632.93	37.73
5	2166.91	35.72	2479.77	34.34	635.69	37.73
RD mode	**Video sequences (QCIF)**					
	Claire		**Coastguard**		**Foreman**	
	Rate	**PSNR**	**Rate**	**PSNR**	**Rate**	**PSNR**
0 (off)	35.40	40.59	258.68	40.36	129.20	38.49
1 (on)	32.67	40.44	246.33	40.29	119.24	38.40
2 (on)	32.68	40.42	246.24	40.30	119.80	38.40

Table 6-13 R-D varies with intra period, QP and threshold mode respectively

Intra period	Video sequences					
	Claire (QCIF)		Coastguard (CIF)		Foreman (QCIF)	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
0	35.40	40.59	1387.88	40.62	129.20	38.49
1	390.80	41.26	3547.60	41.33	771.95	39.32
5	101.95	41.09	1767.57	40.99	242.26	38.97
10	67.86	41.02	1578.65	40.91	183.95	38.80
QP	Video sequences (SIF)					
	Mobile		Garden		Paris	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
23	3988.35	38.98	4246.58	38.39	1113.03	41.09
28	2111.61	35.25	2445.59	34.34	608.55	37.73
32	1049.94	32.89	1402.39	31.81	343.94	35.56
Disable threshold	Video sequences					
	Paris (CIF)		Garden (SIF)		Foreman (QCIF)	
	Rate	PSNR	Rate	PSNR	Rate	PSNR
0 (off)	608.55	37.73	2445.59	34.34	129.20	38.49
1 (on)	688.06	38.21	2583.76	34.84	165.68	39.06

Table 6-14 Rate and distortion varies with prediction modes

Prediction mode	Video sequence					
	Paris		Mobile		Garden	
Inter frame	Rate	PSNR	Rate	PSNR	Rate	PSNR
All modes are on	608.55	37.73	2115.61	35.25	2445.59	34.34
4 x 4 mode off	603.86	37.75	2094.90	35.25	2423.39	34.33
4 x 8 mode off	617.75	37.72	2131.21	35.24	2460.17	34.33
8 x 4 mode off	614.36	37.70	2122.81	35.24	2477.54	34.33
8 x 8 mode off	621.87	37.72	2147.09	35.25	2474.01	34.34
8 x 16 mode off	611.36	37.73	2123.13	35.25	2447.25	34.34
16 x 8 mode off	610.87	37.73	2124.42	35.25	2449.12	34.34
16 x 16 mode off	683.39	37.79	2164.50	35.27	2470.93	34.34
Prediction mode	Video sequence					
	Paris		Mobile		Garden	
Intra frame	Rate	PSNR	Rate	PSNR	Rate	PSNR
All modes are on	608.55	37.73	2115.61	35.25	2445.59	34.34
4 x 4 parallel off	613.21	37.73	2117.67	35.25	2445.47	34.33
4 x 4 45° diagonal off	609.54	37.74	2118.31	35.26	2445.21	34.34
4 x 4 other diagonal off	609.66	37.75	2116.05	35.25	2446.35	34.34
16 x 16 parallel off	608.07	37.75	2114.85	35.25	2444.12	34.34
16 x 16 plane off	609.44	37.74	2116.17	35.26	2442.25	34.34

The intra or inter prediction modes do not appear to have a significant impact on the R-D performance. It is clear from Table 6-14 that, for inter prediction, the variation in rate is around 1-2%, except "Paris" with 16 x 16 mode off which has 12.3% increase in rate as compared to the benchmark. The latter observation is due to the fact that most macroblocks within the background (unchanged) of the video "Paris", is coded using the 16 x 16 mode (so when the 16 x 16 mode is off, the smaller size prediction modes have to use, which result in an increase in motion vectors). Furthermore, the bit rate remains almost the same as the benchmark for any of the intra predictions. The quality of image is almost constant whatever prediction mode (intra or inter prediction) is used.

Based on the above analysis, the most appropriate coding parameters for consideration towards R-D performance of a video CODEC are; *Resolution*, *IntraPeriod*, *QP* and *DisableThresholding*.

6.5 Decoder Estimation

In the previous section, the encoding parameters that have a significant impact on rate, distortion, CPU utilization and memory, were identified through careful design and implementation of a number of experiments. In this section we perform a similar analysis to determine the significant decoder parameters. It is noted that we assume an ideal network where the network does not introduce any data loss or delay. Based on this observation, we assume that the quality and bit rate of the video received by the decoder are the same as that at the encoder output. In other words, we deduce that the coding parameters affecting the bit rate and reconstructed video quality on both encoder and decoder are identical. Therefore in this section the decoder related study is focused only on computational complexity and memory utilization.

6.5.1 Computational Complexity

The analysis method used for estimating the computational complexity of the decoder is the same as that used for the encoder. The computational complexity is first evaluated by calculating the number of CPU cycles (i.e., the actual time) spent

on the main functions (see Equation (6.1)). The major emphasis of the second stage is on exploring the effect of coding parameters on these functions. The profiling tests for the decoder (JM 10) were performed on the PC (Intel P4-2800MHz).

The "Claire", "Coastguard", "Foreman" and "Garden" video sequences were coded at 30 fps with QP = 28. The percentage of processing time spent on the main functions and the total times consumed in decoding each video sequence are summarized in Table 6-15. In terms of H.264 decoding stages in Figure 2-3 (b), the main functions are grouped into the relevant decoding stages for evaluation of processor utilization in the table. The inverse quantization and inverse DCT processes are grouped together and are indicated as "Inverse Quantization & Transform". Other functions that do not belong to any identified decoding stage shown in Figure 2-3 (b) are grouped together as "Remaining functions".

The distribution of computational complexity amongst the decoder's major functions is clearly shown in the Table 6-15. It is observed that the "Deblocking filter" is the most complex component of the decoder (Note that the "Deblocking filter" can not be disabled within the JM 10. However, the "Deblocking filter" may be switched off in other H.264 CODECs and this may be a useful decoder parameter to investigate in further work), accounting for 37.46% (averaged over 4 sequences) of the decoding time. "Motion compensation" follows requiring around 28.31% of the decoding time. Further "Entropy decoding" accounts for 14.23% and "Inverse quantization & transform" accounts for 12.06% of the decoding time. The results are very similar to that obtained in [57]. The next task is to explore the source code of the decoder to identify the coding parameters that can have above 5% effect on the computational complexity. The list of significant parameters selected as a result of the above experiments is tabulated in Table 6-16.

For detailed computational complexity analysis three video sequences "Foreman", "Mobile" and "Coastguard" at two different resolutions QCIF and CIF were used. As previously stated the videos were fixed at 30 fps and QP = 28. The default settings for each coding parameter tabulated in Table 6-16 were used as the benchmark result

Table 6-15 Profiling results of four sequences decoded by the decoder

Coding stage	Video sequence			
	Claire	Coastguard	Foreman	Garden
Entropy decoding	15.17%	14.36%	14.25%	13.15%
Inverse Q&T	12.87%	11.83%	11.36%	12.16%
MC	25.15%	26.28%	26.35%	27.65%
Intra prediction	3.01%	2.56%	2.37%	1.91%
Deblocking filter	36.13%	37.43%	37.94%	38.33%
Reconstruction & store	3.71%	3.37%	3.68%	3.57%
Remaining functions	3.96%	4.17%	4.05%	3.23%
Total coding time (Seconds)	7.509	8.638	8.256	17.968

Table 6-16 Candidates used for parameter selection in complexity at decoder

Coding parameter	Range of value	Default
Resolution	QCIF, SIF, CIF	QCIF
NumberReferenceFrames	1 - 5	1
IntraPeriod	0 – 15	0
InterSearch4x4	0 - 1	1 (on)
InterSearch4x8	0 - 1	1 (on)
InterSearch8x4	0 - 1	1 (on)
InterSearch8x8	0 - 1	1 (on)
InterSearch8x16	0 - 1	1 (on)
InterSearch16x8	0 - 1	1 (on)
InterSearch16x16	0 - 1	1 (on)
DisableIntra4x4modes	1 - 0	0 (off)
DisableIntra16x16modes	1 - 0	0 (off)

for each experiment highlighted in grey in each result table.

The results in Table 6-17 show that a higher resolution (e.g. CIF) video sequence requires approximate double the time to decode as compared to that of a video sequence decoded at lower resolution (e.g., QCIF). It is also clear from the table that the processor time utilized is almost similar when up to 5 reference frames are used. It is noted that at the decoder, the number of reference frames is only used to inform the decoder how many decoded frames should be stored in the buffer for motion compensation. The time taken to fetch a block from one of the reference pictures is similar, which explains the reason for the above observation. In addition to resolution and the number of reference frames, this table also describes the variation in processor utilization at different intra refresh rates. The processor consumption increases gradually with increasing intra frame refresh rate. It is because intra decoding has more calculations than inter decoding. Over 5% gain in decoding time is observed when using 1:1 refresh rate.

The results tabulated in Table 6-18 clearly show that the processing time only negligibly varies regardless of which prediction mode is off. The biggest variation in the processing time is found in the "Paris" sequence (approximate 2.3%) when 16 x 16 inter prediction mode was off. The reason is that the fixed background in "Paris" was observed to be mostly coded using the 16 x 16 prediction mode. It is noted that when the 16 x 16 prediction mode is used, only one motion vector is required in coding a block, as against the need of more than one motion vector when other modes are used.

Table 6-17 Processor utilization (in seconds) at various video resolutions, reference frames and intra period

Resolution	Video sequence		
	Foreman	**Mobile**	**Coastguard**
QCIF	8.256	9.241	8.638
CIF	16.482	20.840	19.263
Reference frame	**Claire (QCIF)**	**Garden (SIF)**	**Football (SIF)**
1	7.509	17.968	17.717
2	7.521	17.950	17.675
3	7.464	17.958	17.602
4	7.466	17.900	17.560
5	7.483	17.856	17.604
Intra period	**Tennis (SIF)**	**Paris (CIF)**	**Coastguard (CIF)**
0	14.691	15.255	19.263
1	15.844	17.930	22.345
5	14.850	15.648	19.835
15	14.724	15.399	19.589

Table 6-18 Processing time (second) varies with different prediction modes

Prediction mode	**Video sequence**				
Inter frame	**Paris**	**Foreman**	**Football**	**Mobile**	**Garden**
All modes are on	15.255	16.482	17.717	20.840	17.968
4 x 4 mode off	15.233	16.483	17.710	20.890	18.084
4 x 8 mode off	15.289	16.533	17.762	20.873	18.045
8 x 4 mode off	15.277	16.533	17.780	20.908	18.101
8 x 8 mode off	15.338	16.470	17.748	20.852	18.054
8 x 16 mode off	15.248	16.554	17.869	20.841	18.064
16 x 8 mode off	15.291	16.478	17.723	20.875	18.086
16 x 16 mode off	15.602	16.753	17.886	20.829	18.092
Intra frame	**Paris**	**Foreman**	**Football**	**Mobile**	**Garden**
All modes are on	15.255	16.482	17.717	20.840	17.968
4 x 4 modes off	15.261	16.574	17.869	20.894	18.043
16 x 16 modes off	15.269	16.524	17.741	20.834	17.923

6.5.2 Memory Utilization

The approach used for analyzing the storage (memory) requirement for the H.264 decoder is the same as the one utilized for the encoder. Table 6-19 presents the summary of the memory requirements of the decoder based on various buffer levels. In the table, *w*, *h*, *n* and *sg* have the same definitions as before (see Section 6.4.2). It is clear from the table that there are only three coding parameters (*Resolution*, *NumberReferences* and *Slicegroup*) dominating the storage requirements of the decoder. It is observed that screen resolution is the most significant parameter affecting the memory requirement. For example, a high-resolution video (e.g., CIF) requires approximate 4 times memory than a lower resolution video (e.g., QCIF).

Table 6-19 represents the memory requirement formulae of the decoder at different buffer levels. It is noted that the constant memory used for variable-length decoding tables, intra-prediction probability tables, quantization tables and a few other small constant tables, are not included in this table. Although error concealment has not been a focus of this thesis, extra memory is required including the calculation since it is allocated automatically when the decoder starts.

Table 6-19 Memory requirement formulas for decoder (JM) at various buffer levels

Buffer level	**Formula (in Bytes)**	**QCIF** sg = n = 1	**CIF** sg = n = 1
Frame	$3*(n+1)*w*h$	152064	608256
Slice	$sg>1?\ 12*(n+1)+5*((w*h)\gg 8)$: $12*(n+1)+((w*h)\gg 6)+4$	424	1612
MB	$400 + 1589*((w*h)>>8)$	157711	629644
Intra prediction modes	$(w*h)>>2$	6336	25344
Motion vectors	$(w*h)\gg 1$	12672	50688
Entropy (CAVLC)	$3*((w*h)>>3)$	9504	38016
Error concealment	$68 + 188*((w*h)\ >>\ 8)$	18680	74516
Total		357391	1428076

Note: *n*: number of reference frames, *w*: width of frame, *h*: height of frame, *sg*: slice group

6.6 Conclusions

This chapter has presented a detailed analysis of the H.264 baseline profile CODEC (JM 10) based on computational complexity, memory utilization, bit rate and distortion. The experimental analysis has lead to the identification of coding parameters (both encoder and decoder parameters, see Table 6-20) that can significantly influence the CODEC's complexity, memory, bit rate and distortion performance. These selected parameters will be used for the CODEC's performance optimization in the next chapter.

Note: The cell marked with √ indicates that a particular parameter significantly affects the relevant constraint. Blank entries indicate non-significant or no effect. For example, the variation in QP does not increase or reduce the computational complexity or memory requirement. It is further observed that a slight variation in visual quality and compression ratio can be gained by adopting different motion estimation algorithms (*UseFME*).

Table 6-20 Significant coding parameters on H.264 CODEC (JM 10)

Encoder					
Parameter	**Symbol**	**Complexity**	**Memory**	**Bitrate**	**Quality**
NumberReferences	x_1	√	√		
SearchRange	x_2	√	√		
UseFME	x_3	√	√		
RDOptimization	x_4	√			
SliceGroup	x_5		√		
QP	x_6			√	√
IntraPeriod	x_7			√	√
DisableThreshold	x_8			√	√
Resolution	x_9	√	√	√	√
Decoder					
Parameter	**Symbol**	**Complexity**	**Memory**	**Bitrate**	**Quality**
NumberReferences	x_1		√		
IntraPeriod	x_2	√		√	√
SliceGroup	x_3		√		
QP	x_4			√	√
DisableThreshold	x_5			√	√
Resolution	x_6	√	√	√	√

Chapter 7 Multi-Objective Performance Optimization of a H.264 CODEC

7.1 Introduction

An investigation on coding parameters that significantly affect the H.264 CODEC's (JM 10) bit rate, reconstructed video quality, memory and CPU utilization was carried out in Chapter 6. A summary of the results is listed in Table 6-20. This chapter presents a joint complexity-memory-rate-distortion (C-M-R-D) multi-objective optimization framework for the H.264 CODEC based on the results of Chapter 6.

The proposed framework considers a video streaming system, which uses a multi-objective optimization scheme to produce a set of optimal configurations of coding parameters according to the CODEC's computing resources constraints to maximize the video presentation quality. The theoretical model of the proposed framework is summarized by the block diagram of Figure 7-1. The client (receiver/decoder) initiates the process by sending a request of video content alongside a decoder capability description that includes bandwidth, memory and CPU utilization constraints. The server (sender/encoder) then selects a set of the best coding parameters for the CODEC from a group of optimal or near optimal tradeoff parameter sets which is produced by the proposed scheme under the joint consideration of client and server resource constraints. Note that since the encoder needs to code the requested video under client's resource limitation, the optimal parameter set of the decoder should be generated at first. Subsequently, the corresponding optimal parameter set for the encoder is selected which is used in encoding. Finally, the coded video is delivered to the receiver. For simplicity, the proposed framework assumes a lossless, wired network environment, thus not requiring the modeling of network parameters within an end-to-end encoder-channel-

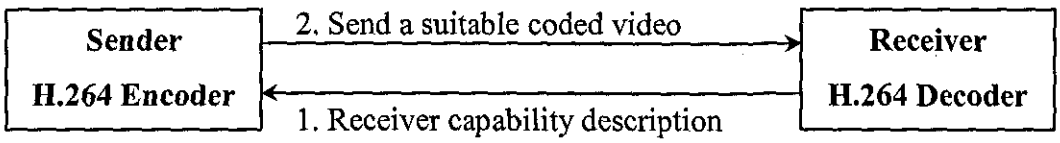


Figure 7-1 A theoretical framework of optimization mechanism

decoder optimization framework. It is noted that a common lossy channel requires the consideration and modeling of network parameters such as transmission delay and packet loss etc., which is considered beyond the scope of the work presented in this thesis. In Chapter 8, we consider this as a possible future extension. It is noted that the proposed framework provides theoretical guidelines for system design and performance optimization of a H.264 video CODEC that can be easily applied to any other image or video CODEC.

The proposed framework is accomplished by following two main steps. Firstly, the objective function for each objective/constraint requires to be developed. Secondly, these objective functions are used within a multi-objective optimization strategy that adopts a genetic algorithm (GA) to produce optimal solutions. Two reasons have contributed in choosing a GA based approach as against a Lagrange Multiplier based approach for multi-objective optimization. The first reason is that a GA has the ability to find multiple optimal solutions in one single simulation run. This is in contrast to the Lagrange Multiplier that converts a multi-objective optimization problem to a single-objective optimization problem by emphasizing one optimal solution at a time [63]. The second reason is the presence of a well established, popular public software tool, Non-dominated Sorting Genetic Algorithm (NSGA-II) [63] that can effectively be utilized in the proposed work (see Section 7.2.3).

For clarity of presentation, this chapter is organized as follows. The fundamental concepts of multi-objective optimization and NSGA-II software are introduced in Section 7.2. Section 7.3 formulates the joint C-M-R-D optimization problem for the framework. The approach of obtaining objective functions of the optimization problem for encoder and decoder is presented in Section 7.4. The simulated results are presented in Section 7.5. Finally, Section 7.6 concludes this chapter.

7.2 Introduction to Multi-objective Optimization

7.2.1 Definition of a Multi-objective Optimization Problem

As the name suggest, a multi-objective optimization problem (MOOP) has a number of objective functions which are to be minimized or maximized. Moreover, the problem usually has a number of constraints which any feasible solution must satisfy. The general form of the MOOP may be stated as follows [64]:

$$\begin{aligned} \text{Minimize/Maximize } & f_m(X), && \text{m=1,2,...,}M; \\ \text{subject to } & g_j(X) \geq 0, && j=1,2,...,J; \\ & h_k(X) = 0, && k=1,2,...,K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i=1,2,...,n. \end{aligned} \tag{7.1}$$

There are M objective functions $f(X) = (f_1(X), f_2(X), ..., f_M(X))^T$ considered in Equation (7.1). A solution X is a vector of n decision variables: $X = (x_1, x_2, ..., x_n)^T$. The terms $g_j(X)$ and $h_k(X)$ are called inequality and equality constraint functions respectively. The last set of constraints are called variable bounds, the value of each decision variable x_i is restricted within a range of lower $x_i^{(L)}$ and an upper $x_i^{(U)}$ bound. These bounds form a *decision variable space*, or simply the decision space. Each feasible solution is subjected to J inequality and K equality constraints. If any solution X does not meet all of $(J+K)$ constraints and all of the variable bounds, the solution is called an *infeasible solution*. On the other hand, if a solution X satisfies all constraints and variable bounds, it is known as a *feasible solution*. It is noted that the entire decision variable space need not be feasible. The set of all feasible solutions is called the *feasible region*.

7.2.2 Pareto-Optimal Solutions

It is important to note that not all feasible solutions in the feasible region are optimal. In other words, the feasible region not only contains optimal solutions, but also

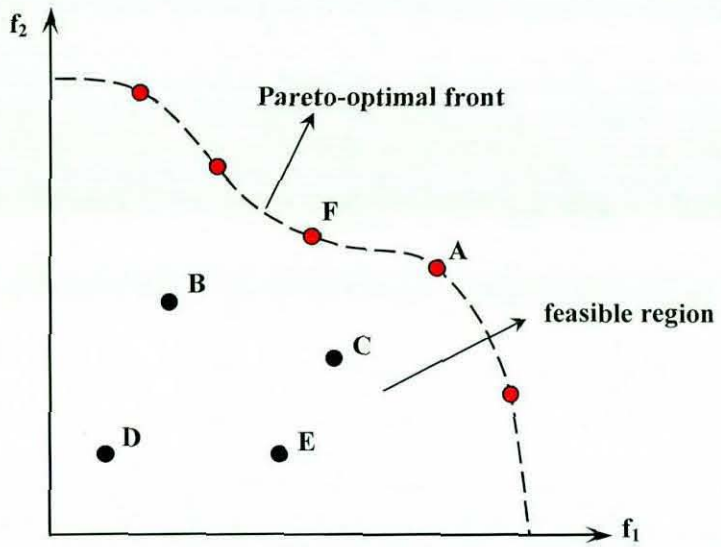


Figure 7-2 Feasible region and Pareto-optimal front in two-objective optimization problem

solutions that are not optimal. Figure 7-2 [67] illustrates the feasible region and a number of feasible solutions of a two-objective optimization problem with two conflicting objectives f_1 and f_2. Assume that the target of this optimization problem is to maximize both objectives. It is obvious that solution A is better than any one of B, C, D and E in terms of both objectives. Therefore, B, C, D and E are feasible solutions but not optimal. On the other hand, solution F has a smaller f_1, but has a larger f_2 than A. When both objectives are equally important, none of these two solutions can be said to be better than the other with respect to both objectives. When this relationship exists between two solutions, they are called *non-dominated* solutions. There exist many such solutions (highlighted with red) in the feasible region. For clarity, these solutions are joined with a dashed curve in the figure. All solutions lying on this curve are called *Pareto-optimal* solutions or *Pareto-optimal* set. The curve formed by joining these solutions is known as a *Pareto-optimal front (curve)*. In fact, the task in multi-objective optimization (MOO) is to find a set of *Pareto-optimal* solutions in the feasible region [64].

7.2.3 NSGA-II Software

Non-dominated sorting genetic algorithm II (NSGA) was proposed in [63], which is at present one of the popular evolutionary algorithms (EAs) used in multi-objective optimization research. Since NSGA II works with a population of solutions, it can be extended to maintain a multiple set of solutions. With an emphasis for moving toward the Pareto-optimal region, the NSGA-II can find multiple Pareto-optimal solutions in one single simulation run [63]. A well tested, public domain software solution for NSGA-II exists [65] that can be used directly within the research context presented in this thesis. An introduction to this tool is presented in the following sub-sections.

7.2.3.1 Input Parameters

The input parameters that need to be defined prior to running NSGA-II, are listed in Table 7-1. Note that the digits in the bracket denote the value or value range of the corresponding parameter. In addition to the input parameters, a MOOP and its objective (fitness) functions also need to be defined. The definition of MOOP and objective functions used for the optimization framework in this thesis are discussed *in Section 7.3 and Section 7.4 respectively.*

Table 7-1 Input parameters of NSGA-II

Input parameter	Description
popsize	Population size
ngen	Number of generations
nobj	Number of objectives
ncon	Number of constraints
nreal	Number of real variables
min_realvar[i]	Minimum value of i^{th} real variable
max_realvar[i]	Maximum value of i^{th} real variable
pcross_real	Probability of crossover of real variable (0.6-1.0)
pmut_real	Probability of mutation of real variable (1/nreal)
eta_c	Distribution index for simulated binary crossover (5-20)
eta_m	Distribution index for real variable polynomial mutation (5-50)

7.2.3.2 Main Procedure

Initially a random parent population P_0 is created. The population is then sorted into different non-domination levels, where *level 1* is used to refer to the best level. Each solution is assigned a fitness that equals to its non-domination level. Subsequently a binary tournament selection with a crowded tournament operator, recombination, and mutation operators are used to create an offspring population Q_0 of size N (the population size). After the initial generation, the procedure (as in [63]) is outlined as follows (the mth generation is given as an example):

Step 1: Parent and offspring population is combined to form a new population $R_m = P_m \cup Q_m$.

Step 2: The combined population R_m (of size $2N$) is sorted into different levels of non-dominated sets F_1, F_2 ....

Step 3: New population P_{m+1} is formed from the non-dominated sets in the order of F_1, F_2 ... until the size equals to N. If only a portion of the last non-dominated set, F_k, is selected to fill the new population, the best solutions in F_k are chosen by the crowded-comparison. The procedure is also shown in Figure 7-3.

Step 4: The new population P_{m+1} is used for selection, crossover, and mutation to create a new offspring population Q_{m+1}.

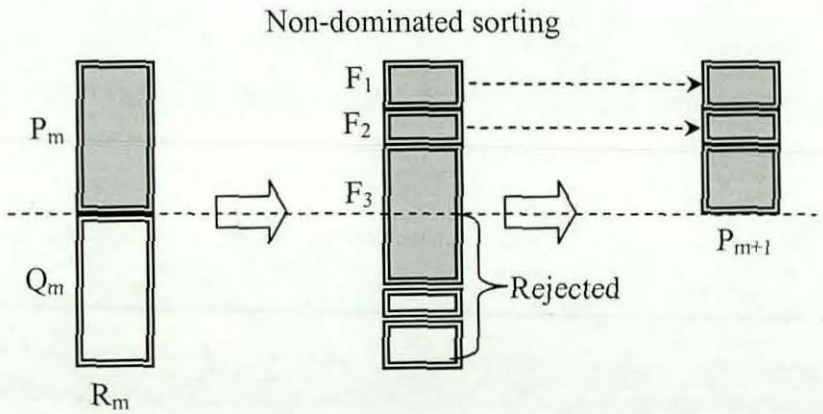


Figure 7-3 NSGA-II procedure

7.2.3.3 Output of NSGA-II

The steps 1 to 4 are continued until the required number of generation is reached. Finally, NSGA-II outputs a file that contains the best solutions obtained at the end of the simulation run.

7.3 Problem Formulation

Consider in Figure 7-1 that the encoder received the client's video demand and capability information such as available memory, CPU constraints/capabilities and available bandwidth. Based on such knowledge, the goal is to find a set of optimal or near optimal solutions that minimize computational complexity, memory utilization and bandwidth utilization while maximizing video quality. Each solution contains a set of optimal coding parameters (listed in Table 6-20) that are used for coding the demanded video sequence. Therefore, the goal can be considered as a four-objective constrained optimization problem that all objectives are to be minimized and presented by $F_{complexity}$, F_{memory}, F_{rate} and $F_{distortion}$ respectively. The budgets of memory and bandwidth can be regarded as constraints (note that CPU can be a constraint as well) and presented by G_{memory} and G_{rate} separately in this thesis. The selected coding parameters are described as decision variables by a vector variable $X = (x_1, x_2, \ldots, x_n)$. According to such given knowledge, the MOOP regarding C-M-R-D optimization model can be formulated in its general form as follows:

$$\min F(X) = \left(F_{complexity}(X), F_{memory}(X), F_{rate}(X), F_{distortion}(X)\right)^T$$

$$\begin{aligned} &subject\ to\ G_{rate}(X) \le R, \\ &\qquad G_{memory}(X) \le M, \\ &\qquad x_i^L \le x_i \le x_i^U, \qquad i = 1,2,\ldots,n. \end{aligned} \tag{7.2}$$

where R and M represent the constrained rate and memory respectively. Decision variables $n = 9$ for the encoder and $n = 6$ for the decoder in the proposal. Each decision variable x_i is restricted within a range $[x_i^L, x_i^U]$ inclusive. The subsequent

task is to obtain the four objective functions $F_{complexity}$, F_{memory}, F_{rate} and $F_{distortion}$, and two constraint functions G_{memory} and G_{rate}. This is discussed in the following section. Note that $G_{memory}(X) = F_{memory}(X)$ and $G_{rate}(X) = F_{rate}(X)$.

7.4 Obtaining Objective Functions

The constraint optimization problem was formulated in Equation (7.2) of the previous section. In this section, the task is to analyze the mathematical relationship between the decision variables x_i and each objective. The above relationships named as objective functions, can be obtained through the informed use of Matlab's [66] polynomial regression tool.

The procedure of obtaining objective functions can be described as follows: (i) for each objective a large number of experiments are carried out both at the encoder and decoder based on all possible combinations of settings of the aforementioned coding parameters; (ii) the values obtained for the objective and the corresponding parameter settings are used to form a data set for polynomial regression; (iii) the polynomial terms are defined to include all possible combinations of the decision variables (i.e. the coding parameters); (iv) finally the polynomial regression function of Matlab is used to fit the data set of (ii) and to determine the coefficients of each significant polynomial term of the objective function. Since computational complexity, rate and distortion (however not memory) obtained even under identical parameter settings are dependent on the source video, there are two options for estimating the objective functions. The first option is the use of average value of the objective calculated experimentally from different videos (for example the bit-rate) to be used in polynomial regression. The advantage of this approach is that the resulting objective function will be able to reasonably well model the relationship between the objective and the coding parameters for any given video. However, the results will be sub-optimal. The second option is that for each video sequence to have its own objective function, which can yield optimal results. However, these results will not provide optimal results for other videos. Since the framework is to be considered for use

within a video streaming system, the second option was chosen. The details of this optimization strategy are given in the following sub-sections.

7.4.1 Objective Functions of Encoder

7.4.1.1 Computational Complexity

As mentioned in Section 6.4.1, the computational complexity of the encoder was measured using the Intel VTune Performance Analyser tool. The experiments for obtaining computational complexity data were performed on a Pentium-4 2.8GHz computer with the coding parameter setting described in Table 7-2. The number of reference frames (x_1) varies from 1 to 5 in increments of two. The search window size (x_2) can take either of the two values 16 or 32. The control variable (x_3) can take values from 0 to 3 corresponding to various motion estimation modes. The control variable (x_4) for rate-distortion optimization (RDO) mode can take values from 0 to 2. The resolution of video sequence (x_9) varies from 1 to 2 corresponding to QCIF and CIF respectively. Therefore there are a total of 144 combinations of the five control variables stated above. Four video sequences (112 frames for each) "Foreman", "Mother & Daughter", "Mobile" and "Coastguard" were used for analyzing the computational complexity. Therefore, total of 4 x 144 = 576 experiments were run. A subset of the experimental results is shown in Table 7-3. The computational complexity (measured in seconds) per frame obtained by averaging the total number of frames was used for evaluating the objective function.

Table 7-2 Coding parameter settings for estimating complexity

Sequence	Data set	x_1	x_2	x_3	x_4	x_9
Foreman	144	1 - 5 (+2)	16 – 32	0 - 3	0 - 2	1 - 2
Coastguard	144	1 - 5 (+2)	16 – 32	0 - 3	0 - 2	1 - 2
Mo. & Da.	144	1 - 5 (+2)	16 – 32	0 - 3	0 - 2	1 - 2
Mobile	144	1 - 5 (+2)	16 – 32	0 - 3	0 - 2	1 - 2

Table 7-3 The average computational complexity (seconds) per frame for each sequence.

x_1	x_2	x_3	x_4	x_9	Foreman	Coastguard	Mo. & Da.	Mobile
1	16	0	0	1	0.540464286	0.541303571	0.535616071	0.538133929
3	16	0	0	1	1.429142857	1.439008929	1.419946429	1.406589286
5	16	0	0	1	2.309946429	2.3308125	2.291232143	2.259321429
5	16	1	0	1	0.850214	1.053848	0.752402	0.953196
5	16	2	0	1	0.685786	0.803732	0.551902	0.766857
5	16	3	0	1	0.926821	0.993732	0.840964	0.918804

It is noted that all possible combinations of control variables in different degrees were initially considered in an attempt to exhaustively determine the terms of each fitness function. Since normally higher-order terms produce a better fitness (i.e., higher *R-square* [66] which is one minus the ratio of the error sum of squares to the total sum of squares, a value closer to 1 indicates a better fit.) to a data set, a polynomial of degree six with a total of 420 terms was used for obtaining the fitness function for computational complexity. However finally, only the significant terms are retained. The polynomial regression results of the four test video sequences are shown in Table 7-5. It was observed that the *R-square* and *RMSE* (root mean square error) values are maintained at desirable levels. The significant terms and relevant coefficients of the fitness function for "Mother & Daughter" video sequence are given in Table 7-4. A complete set of terms and coefficients obtained for all four test video sequences are listed in Appendix A. In the table, each row indicates a monomial of the polynomial consisting of a coefficient multiplied by one or more control variables. The integer number describes the power of the relevant control variable. For example, the 1st row is a constant term because of $-0.2041517956412x_1^0x_2^0x_3^0x_4^0x_9^0$ and the 10th row is $1.3794327907866x_1^1x_2^1x_3^2x_4^0x_9^2$. Thus, the general form of the objective function for computational complexity can be written as follows:

$$F_{complexity\_E}(X_c) = \sum_{i=1}^{n} c_i x_1^a x_2^b x_3^c x_4^d x_9^e, \qquad a,b,c,d,e = 0,...,5; \quad n = 31,26,19,20. \tag{7.3}$$

where $X_c = (x_1, x_2, x_3, x_4, x_9)^T$, a, b, c, d and e are the corresponding powers of the relevant variables. The number n indicates the number of sufficiently significant polynomial terms of the objective functions for sequences "Foreman", Coastguard", "Mother & Daughter" and "Mobile".

Table 7-4 Terms and coefficients of the fitness function of Mother & Daughter

Coefficients	x_1	x_2	x_3	x_4	x_9
-0.2041517956412	0	0	0	0	0
0.3165646733353	0	0	1	0	0
-0.3385939801707	0	0	1	0	1
0.2750319893053	0	0	0	1	1
-2.3004076262064	1	1	1	0	0
-2.5655364292056	1	0	0	0	2
3.7864693852438	1	1	0	0	2
2.9953569554126	1	0	1	0	2
-1.0795540124119	1	0	2	0	2
1.3794327907866	1	1	2	0	2
0.3057298569872	1	1	3	0	0
-0.0293689868774	1	1	0	0	3
-0.4582364962363	1	1	3	0	1
-1.7154566609996	1	1	1	0	3
0.1250301095724	1	0	3	0	2
0.0338324704338	0	0	0	4	0
0.0038274781955	0	0	2	0	4
0.0316089895731	0	0	0	2	4
-0.0124814415485	0	0	0	5	1

Table 7-5 Fitness results of computational complexity

Sequence	R-square	RMSE
Foreman	0.9999	0.055
Coastguard	0.9999	0.110
Mother & Daughter	0.9995	0.020
Mobile	0.9999	0.047

7.4.1.2 Memory Utilization

The memory requirement for the encoder was analyzed in Section 6.4.2. It was revealed that the encoder memory requirement only depends on the coding parameters but not on the video content. Therefore, the objective function for memory utilization should theoretically be applicable to any video sequence. The parameter settings for memory utilization related experiments are shown in Table 7-6. The number of slice groups (x_5) varies from 1 to 7. The variations of values of the other parameters are the same as that of computational complexity. The same 420 terms used in obtaining the objective function for computational complexity was used for obtaining the objective function for memory utilization. A highest degree of six was used. The statistical results of *R-square* and *RMSE* reflecting the goodness of fit are tabulated in Table 7-6 and are promising. The final objective function of memory usage contains 35 terms which is listed in Appendix A and the equation can be described in general terms as follows:

$$F_{memory\_E}(X_m) = \sum_{i=1}^{n} c_i x_1^a x_2^b x_3^c x_5^d x_9^e, \qquad a,b,c,d,e = 0,...,5; \quad n = 35. \tag{7.4}$$

where $X_m = (x_1, x_2, x_3, x_5, x_9)^T$, a, b, c, d and e are defined similar to that in Equation (7.3).

Table 7-6 Coding parameter settings and the fitness results of memory

Data set	x_1	x_2	x_3	x_5	x_9	R-square	RMSE
336	1 - 5 (+2)	16 – 32	0 - 3	1 - 7	1 - 2	0.99999	0.003

7.4.1.3 Rate & Distortion

Since rate and distortion have the same control parameters, they can use identical data sets as listed in Table 7-7. The quantization parameter (x_6) varies from 17 to 49 with assumed increments of four for experimental purposes. The control variable (x_7), *IntraPeriod*, can takes values: 0 (means that the first frame is coded as an I-frame and subsequent frames are coded as P-frames), 1 (all frames coded as I-frames), 4, 7, 10, 13 and 16. The control variables (x_8) and (x_9) represents *DisableThreshold* (see Section 6.2) and resolution of video respectively, and can take only one of two possible values. The same test video sequences and an identical specification PC as used in the experiments presented in Section 7.4.1.1 were used. A total of 252 experiments for each sequence were performed. The averaged distortion (measured in terms of PSNR, dB units) and rate (measured in kbits) per frame were employed as the input data set. A total of 329 polynomial terms (the highest-order terms are in degree seven) were used in the definition of the initial fitness functions for rate and distortion respectively. The results of goodness of fit are presented in Table 7-8. The fitness (see *R-square* and *RMSE* in the table) for the average rate reveals that a polynomial does not fit the average rate well. Therefore, in order to improve the goodness of fit, the average rate was log-transformed (base 10) prior to fitting to the data set. From the results tabulated in Table 7-8, it is clear that, the log-transformed data can be fitted better than the non-log-transformed data using a polynomial function. Accordingly the terms and the corresponding coefficients of the polynomial obtained for fitting the data are listed in Appendix A. The generalized objective functions for rate and distortion are shown in Equation (7.5) and Equation (7.6) respectively.

$$F_{rate\_E}(X_r) = 10^{\sum_{i=1}^{n} c_i x_6^a x_7^b x_8^c x_9^d}, \qquad a,b,c,d = 0,...,7; \quad n = 27,42,27,34. \tag{7.5}$$

$$F_{distortion\_E}(X_d) = \sum_{i=1}^{n} c_i x_6^a x_7^b x_8^c x_9^d, \qquad a,b,c,d = 0,...,5; \quad n = 23,16,18,19. \tag{7.6}$$

where $X_r = X_d = (x_6, x_7, x_8, x_9)^T$, a, b, c and d vary from 0 to 7 for rate and 0 to 5 for distortion. It is noted that "n" in Equation (7.5) and (7.6) define the same meaning as that of Equation (7.3).

Table 7-7 Coding parameter settings for estimating rate and distortion

Sequence	Data Set	x_6	x_7	x_8	x_9
Foreman	252	17 - 49 (+4)	0 – 16 (+3)	0 - 1	1 - 2
Coastguard	252	17 - 49 (+4)	0 – 16 (+3)	0 - 1	1 - 2
Mo & Du.	252	17 - 49 (+4)	0 – 16 (+3)	0 - 1	1 - 2
Mobile	252	17 - 49 (+4)	0 – 13 (+3)	0 - 1	1 - 2

Table 7-8 Fitness results of rate and distortion

Sequence	Averaged PSNR		Averaged Rate		Log10(Averaged Rate)	
	R-Square	RMSE	R-Square	RMSE	R-Square	RMSE
Foreman	0.9998	0.09	0.9978	1.24	0.9997	0.011
Coastguard	0.9997	0.11	0.9987	0.93	0.9998	0.011
Mo & Du.	0.9997	0.10	0.9991	0.74	0.9996	0.017
Mobile	0.9997	0.13	0.9962	1.64	0.9998	0.008

7.4.1.4 Formulation of C-M-R-D of Encoder

After obtaining objective functions of computational complexity, memory utilization, rate and distortion, the equation used for optimizing the encoder on C-M-R-D is given by:

$$
\begin{aligned}
&Min\ F_{complexity\_E}(X_c) = \sum_{i=1}^{n} c_i x_1^a x_2^b x_3^c x_4^d x_9^e, \quad a,b,c,d,e = 0,...,5; \quad n = 31,26,19,20;\\
&Min\ F_{memory\_E}(X_m) = \sum_{i=1}^{n} c_i x_1^a x_2^b x_3^c x_5^d x_9^e, \quad a,b,c,d,e = 0,...,5; \quad n = 35;\\
&Min\ F_{rate\_E}(X_r) = 10^{\sum_{i=1}^{n} c_i x_6^a x_7^b x_8^c x_9^d}, \quad a,b,c,d = 0,...,7; \quad n = 27,42,27,34;\\
&Min\ F_{distortion\_E}(X_d) = \sum_{i=1}^{n} c_i x_6^a x_7^b x_8^c x_9^d, \quad a,b,c,d = 0,...,5; \quad n = 23,16,18,19;\\
&subject\ to \quad G_{rate\_E}(X_r) = F_{rate\_E}(X_r) \le R;\\
&\qquad\qquad G_{memory\_E}(X_m) = F_{memory\_E}(X_m) \le M;\\
&\qquad\qquad x_i^L \le x_i \le x_i^U, \qquad i = 1,2,...,9.
\end{aligned}
\tag{7.7}
$$

where X_c, X_m, X_r, X_d, R and M are defined as before.

7.4.2 Objective Functions of Decoder

There are a total of six coding parameters (marked x_1 to x_6 and listed in Table 6-20) that significant affect the decoder's computational complexity, memory utilization, received bit rate and distortion. As mentioned in Section 7.1, the proposed multi-objective optimization framework assumes a lossless network, which means that the decoder can receive all the data transmitted by the encoder at the specific rate. Under such a situation, the reconstructed video quality at the decoder depends entirely on the coding parameters used by the encoder. Thus, the objective functions for video quality (distortion) and rate at the decoder can use the functions derived at the encoder for the said objectives. Thus only the deduction of the objective functions for

computational complexity and memory utilization at the decoder are presented in the following sub-sections.

7.4.2.1 Computational Complexity

From the analysis of the computational complexity at the decoder presented in Section 6.5.1, it was seen that there are only two significant coding parameters that have a significant impact (i.e. over 5%) on the processor's execution time. They are, *IntraPeriod* (x_2) and *Resolution* (x_6). The approach used for obtaining the objective function for computational complexity at the decoder is the same as that used at the encoder. The same PC and test video sequences were used. Table 7-9 presents the parameter settings that were used in the experiments. The values of variable x_2 vary from 0 to 16 with two different frame sizes (i.e. the parameter, x_6) 1 and 2. Therefore a total of 34 experiments were performed for each sequence. A polynomial of degree nine was employed to obtain the fitness function. The fitness statistics are tabulated in Table 7-10. In terms of polynomial terms and coefficients provided in Appendix B, the fitness function can be written as follows:

$$F_{complexity\_D}(X_c) = \sum_{i=1}^{n} c_i x_2^a x_6^b, \qquad a,b = 0,...,5; \quad n = 4,5,7,5. \tag{7.8}$$

where $X_d = (x_2, x_6)^T$, a, b and n are defined as before but are different in value.

Table 7-9 Coding parameter settings for estimating computational complexity

Sequence	Data set	x_2	x_6
Foreman	34	0 – 16	1 - 2
Coastguard	34	0 – 16	1 - 2
Mother & Daughter	34	0 – 16	1 - 2
Mobile	34	0 – 16	1 - 2

Table 7-10 Fitness results of decoder's computational complexity

Sequence	R-square	RMSE
Foreman	0.9984	0.0015
Coastguard	0.9999	0.0005
Mother & Daughter	0.9998	0.0004
Mobile	0.9999	0.0005

7.4.2.2 Memory Utilization

At the decoder (JM 10), three coding parameters, the number of reference frames (x_1), the number of slice groups (x_3) and frame solution (x_6), dominate the memory required for decoding a video sequence (see Table 6-19). The value range for each variable is as follows. Up to 5 reference frames may be used for motion compensation. The maximum number of slice groups is 7 and video resolution can take values of either 1 or 2. A polynomial of degree six with the 70 data sets was used in the polynomial regression for decoder memory utilization. The results are shown in Table 7-11. The objective function can therefore be written as follows, with the polynomial terms and coefficients given in Appendix B:

$$F_{memory\_D}(X_m) = \sum_{i=1}^{n} c_i x_1^a x_3^b x_6^c, \qquad a,b,c = 0,...,6; \quad n = 6. \tag{7.9}$$

where $X_m = (x_1, x_3, x_6)^T$, a, b, c and n are defined as before but are different in value.

Table 7-11 Fitness result of decoder's memory utilization

Data set	R-square	RMSE
70	0.99999	0.0001

7.4.2.3 Formulation of C-M-R-D of Decoder

After obtaining the objective functions for computational complexity and memory utilization, the optimization problem of the decoder becomes:

$$
\begin{aligned}
&Min\ F_{complexity\_D}(X_c) = \sum_{i=1}^{n} c_i x_2^a x_6^b, && a,b = 0,...,5; \quad n = 4,5,7,5;\\
&Min\ F_{memory\_D}(X_m) = \sum_{i=1}^{n} c_i x_1^a x_3^b x_6^c, && a,b,c = 0,...,6; \quad n = 6;\\
&Min\ F_{rate\_D}(X_r) = 10^{\sum_{i=1}^{n} c_i x_2^a x_4^b x_5^c x_6^d}, && a,b,c,d = 0,...,7; \quad n = 27,42,27,34;\\
&Min\ F_{distortion\_D}(X_d) = \sum_{i=1}^{n} c_i x_2^a x_4^b x_5^c x_6^d, && a,b,c,d = 0,...,5; \quad n = 23,16,18,19;\\
&subject\ to \quad G_{rate\_D}(X_r) = F_{rate\_D}(X_r) \le R;\\
&\qquad\qquad G_{memory\_D}(X_m) = F_{memory\_D}(X_m) \le M;\\
&\qquad\qquad x_i^L \le x_i \le x_i^U, && i = 1,2,...,6.
\end{aligned}
\tag{7.10}
$$

where $X_r = X_d = (x_2, x_4, x_5, x_6)^T$, other terms have definitions similar to as before.

7.5 Experimental Results

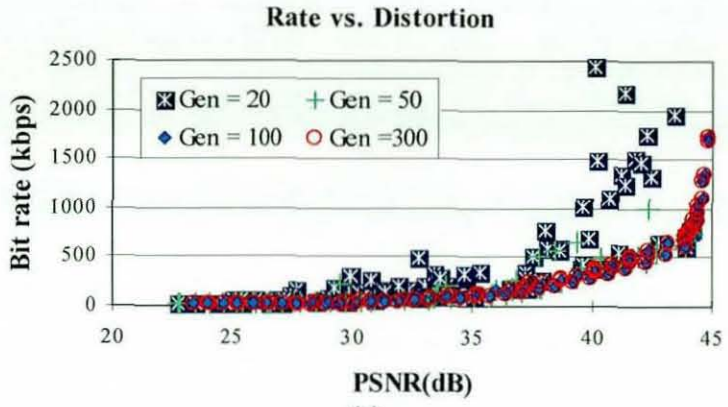
The objective functions for computational complexity, memory, rate and distortion at both the encoder and decoder have been derived in the above sections. Therefore the framework is now ready to be used to solve the C-M-R-D optimization problems at both ends of the CODEC. In this section, a number of simulations are performed to demonstrate the effectiveness of the optimization framework. The parameters used for NSGA-II are first described. A reasonable set of parameter settings was chosen from those proposed by the authors of NSGA-II [63] and therefore it is noted that we have not made any effort in finding the best parameter setting. In all simulations, the population size was chosen as 100, crossover probability was 0.9, and mutation probability (calculated from $1/n$, where n is the number of decision variables) was 0.11 for the encoder and 0.17 for the decoder. The distribution index for both

crossover and mutation were set to 20. All four video sequences were used for the simulations and the four objectives, computational complexity, memory utilization, rate (based on 25 frames per second) and distortion were measured in milliseconds per frame, megabytes, kbps and dB respectively.

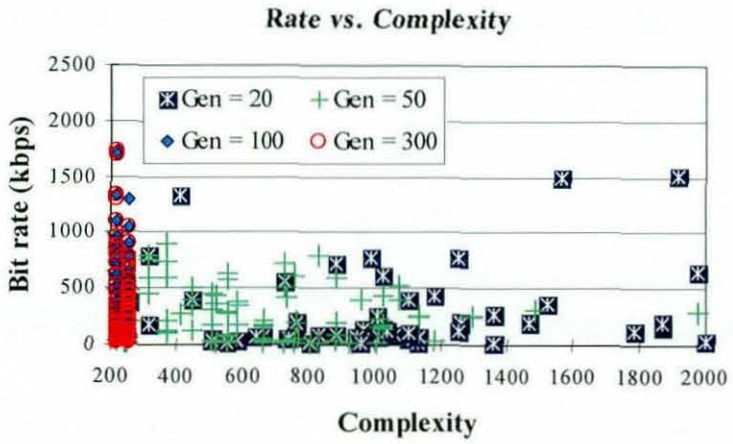
The first simulation was run for various generations. There are three goals in this simulation: (i) verifying if NSGA-II can help in finding a set of solutions which are close to the Pareto-optimal set, (ii) can a set of diverse solutions be found within the Pareto-optimal set, and (iii) obtaining Pareto-optimal solutions for each video sequence. In this simulation, the C-M-R-D optimization was considered as an unconstrained optimization problem so that the feasible region (see Section 7.2) could be as large as possible. 20, 50, 100 and 300 generations were used in the simulation. With four objectives, it is difficult to discuss the effect of the results on a 4-D graph. Therefore, all six pairwise plots of solutions obtained at the end of generations for "Foreman" at the encoder side are illustrated in Figure 7-4 and Figure 7-5.

In achieving the first goal of the simulation, it is observed from the graphs that the solutions converge towards the Pareto-optimal set, as the number of generations increased. However, the convergence hardly improve after 100 generations, which means the solutions obtained after 100 generations are the (local) Pareto-optimal solutions that are very close to the true Pareto-optimal set.

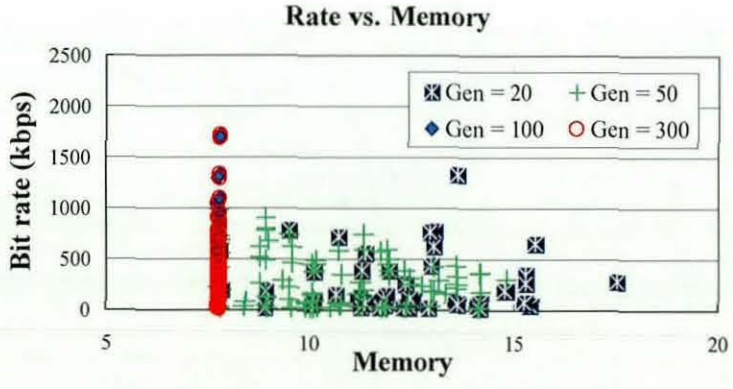
In achieving the second aim of the experiments stated above, it is noted that the spread of the Pareto-optimal solutions obtained at the end of both 100 and 300 generations (marked with blue rhombus and red circle in the figures individually) is also desirable. This is clearly proved in rate-distortion curve of Figure 7-4 (a). Note that, in Figure 7-4 (b)-(c) and Figure 7-5 (a)-(c), a Pareto-optimal solution can be found, which can obtain the maximum or minimum value on both objectives, without tradeoff. This is due to the fact that such pairwise objectives do not conflict with each other.



(a)

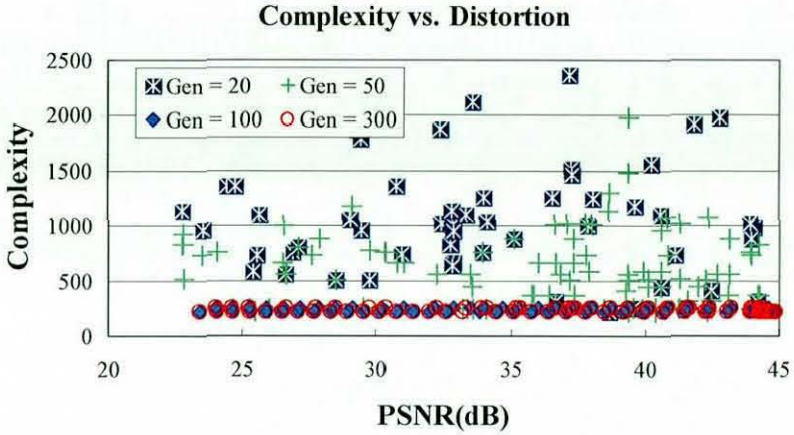


(b)

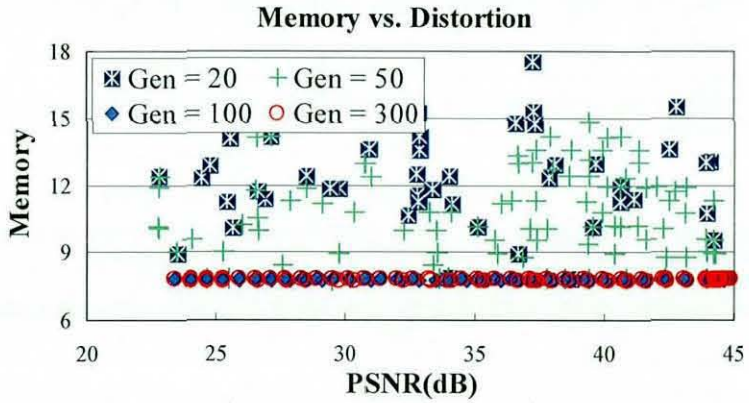


(c)

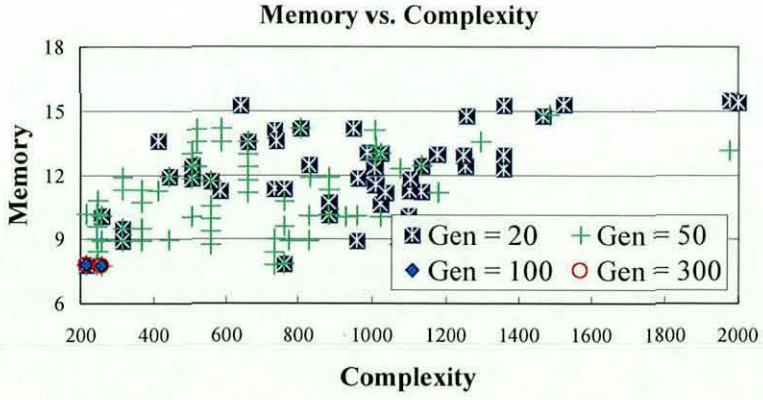
Figure 7-4 Pairwise plots of solutions for rate vs. distortion, complexity and memory respectively



(a)



(b)



(c)

Figure 7-5 Pairwise plots of solutions: (a) complexity vs. distortion, (b) memory vs. distortion, and (c) memory vs. complexity

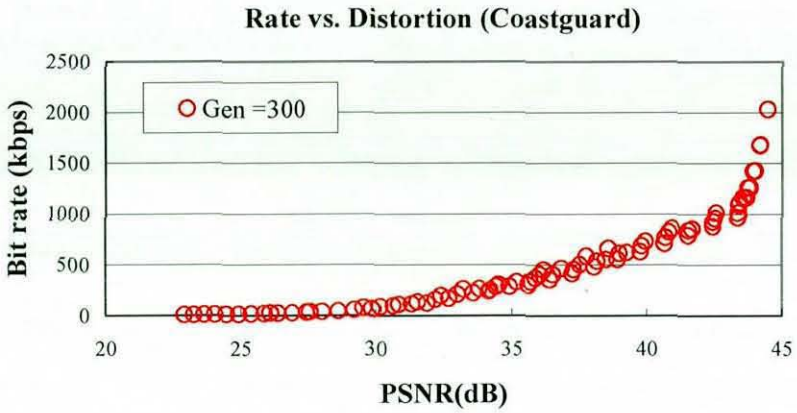
In achieving the third aim of the experiments, it was observed that similar results were obtained when other test video sequences were tested. Figure 7-6 (a)-(c) depict the rate-distortion curves obtained at the end of 300 generations for "Coastguard", "Mother & Daughter" and "Mobile" respectively. Since a population of size 100 was used, it is not expected that the set of solutions obtained at the final generation are the global Pareto-optimal solutions, but rather a local Pareto-optimal set sufficiently close to the global Pareto-optimal solution.

In order to evaluate the performance in the case of constrained Pareto-optimal set, a second simulation was performed where all parameters of NSGA-II tool were selected to be the same as before and 300 generations were run. Two different constraints: i) rate less than 1024 kbps, ii) rate between 56 kbps and 128 kbps, were used for this experiment. The resulting rate-distortion curves for the four sequences are given in Figure 7-7 (a)-(d). It is clear from these figures that the constrained Pareto-optimal set is a subset of the unconstrained Pareto-optimal set as compared with the results obtained in the first simulation. Similar results (see Figure 7-8 (a)-(b) for "Coastguard" and "Mobile" respectively) were obtained for 56 kbps < rate < 128 kbps but it was observed that the region of the Pareto-optimal set becomes smaller.

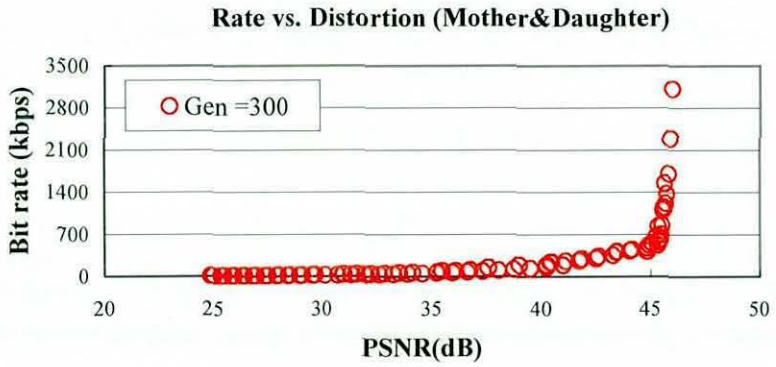
At the decoder, similar simulations were preformed for the same test sequences. It was observed that the behaviour of the resulting curves was similar to that of the encoder.

A third simulation was performed in order to demonstrate the overall procedure one has to adopt in order to make use of the proposed multi-objective optimization framework. Assume a scenario where the decoder has the following limitations:

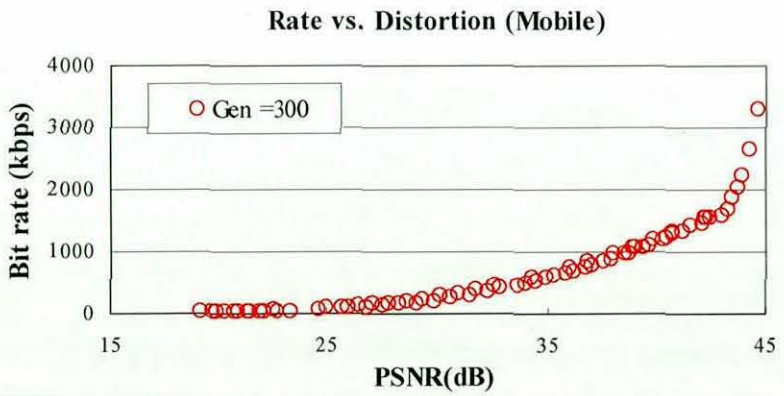
1. The bandwidth in the bound of [128 kbps, 256 kbps] inclusive;
2. Memory is limited on 5 Mbytes.



(a)



(b)



(c)

Figure 7-6 Rate-distortion curves for Coastguard, Mother&Daughter and Mobile

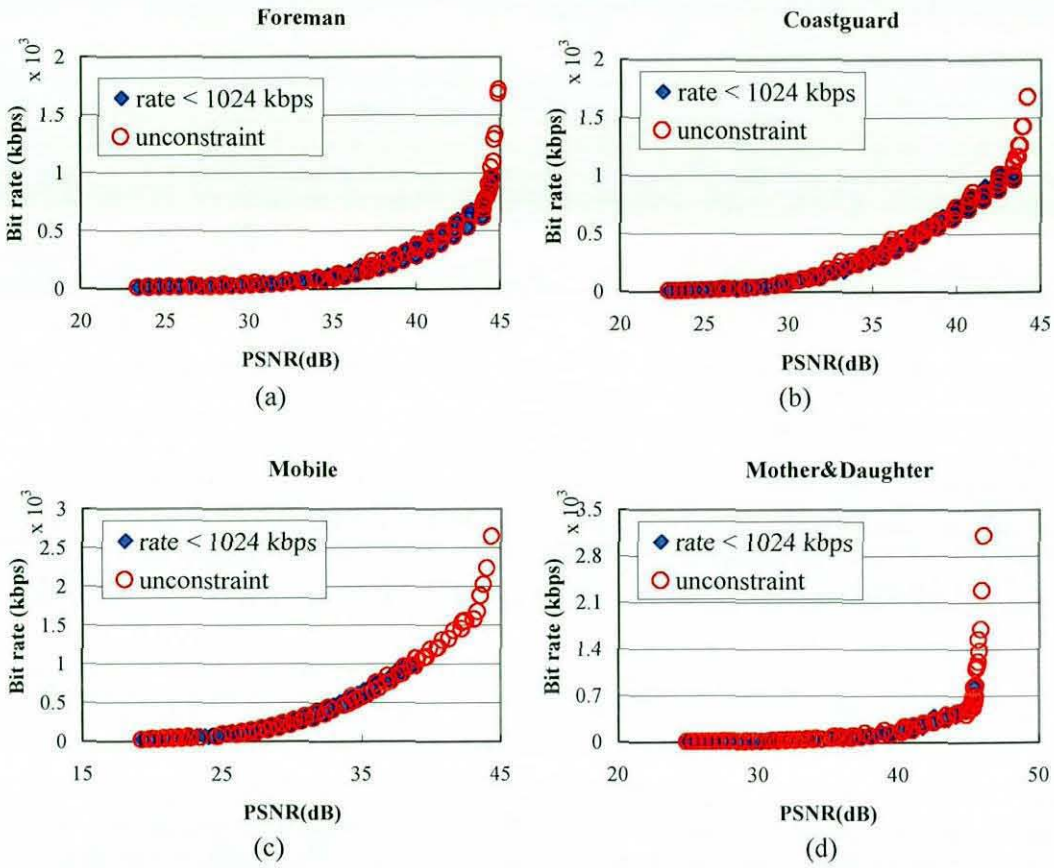


Figure 7-7 Resulting rate-distortion curves (rate < 1024 kbps) for four sequences

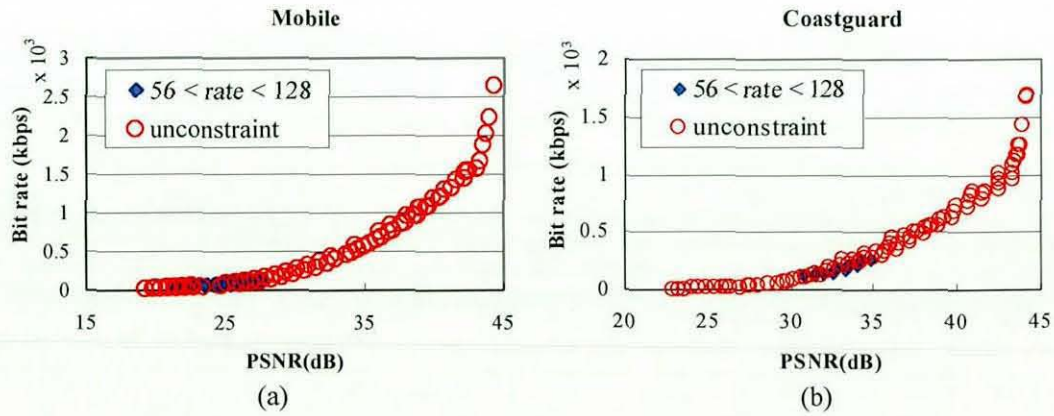


Figure 7-8 Resulting rate-distortion curves (56 < rate < 128) for two sequences

The decoder sends a request for the "Foreman" sequence alongside a decoder capability description, to the server (the encoder). Subsequently the proposed framework then optimizes the coding parameters for the decoder, based on its capability data. A part of the optimization results are given in Table 7-12. The first four columns depict the objectives results while the remaining columns represent the corresponding optimal coding parameters (decision variables) set for the decoder. The range of the coding parameters will be used as a new bound of the decision variables at the encoder. Then the framework optimizes the encoder in terms of the constraints containing the new bound of the decision variables and the limitation of bandwidth. Note that the limitations of computational complexity and memory utilization at the decoder are not used as constraints at the encoder's optimization since these two objectives at the encoder and the decoder are independent. A part of the optimization results for the encoder are shown in Table 7-13. The R-D curves for the encoder and decoder are almost the same as illustrated in Figure 7-9. Finally, the optimal coding parameter set is selected by following two steps. i) Choose the maximum value of PSNR in Table 7-12 (decoder parameters), i.e., the first row highlighted in grey. ii) Select the row from Table 7-13, which has the highest PSNR, and the values of the corresponding parameters must equal or close to the decoder parameters. In this case, the second row (highlighted in grey) of Table 7-13 was selected. Therefore, the optimal coding parameter set for coding "Foreman" sequence under the above constrained memory and bandwidth requirement is listed Table 7-14.

Table 7-12 Optimization results of decoder for Foreman sequence

Rate	PSNR	Complexity	Mem.	Ref. frames	Intra period	Slice group	QP	Disable Threshold	Frame size
254.27	38.68	73.72	0.35	1	12	1	24	1	1
218.36	37.98	73.55	0.35	1	13	1	25	1	1
205.22	37.86	72.47	0.35	1	16	1	25	1	1
238.54	38.56	72.47	0.35	1	16	1	24	1	1
221.08	37.92	72.95	0.35	1	15	1	26	2	1
131.54	35.84	72.47	0.35	1	16	1	28	1	1
193.96	37.28	73.3	0.35	1	14	1	27	2	1
216.33	37.92	72.95	0.35	1	15	1	25	1	1
188.97	37.28	73.3	0.35	1	14	1	26	1	1
171.26	36.65	73.84	0.35	1	11	1	27	1	1

Table 7-13 Optimization results of encoder for Foreman sequence

Rate	PSNR	C	Mem.	Ref.	SR	FME	RDO	SG	QP	IP	DT	Reso.
225.76	37.99	256.22	7.74	1	1	4	1	1	25	11	1	1
254.27	38.68	215.52	7.8	1	1	3	1	1	24	12	1	1
162.06	36.57	215.52	7.8	1	1	3	1	1	27	15	1	1
162.06	36.57	215.52	7.8	1	1	3	1	1	27	15	1	1
205.22	37.86	215.52	7.8	1	1	3	1	1	25	16	1	1
238.54	38.56	215.52	7.8	1	1	3	1	1	24	16	1	1
225.76	37.99	256.22	7.74	1	1	4	1	1	25	11	1	1
238.54	38.56	215.52	7.8	1	1	3	1	1	24	16	1	1
130.67	35.35	256.22	7.74	1	1	4	1	1	29	11	1	1
162.06	36.57	215.52	7.8	1	1	3	1	1	27	15	1	1

Note: C: complexity, Mem: memory, Ref: reference frames, SR: search range, FME: fast motion estimation, RDO: rate-distortion optimization, SG: slice group, IP: intra period, DT: disable threshold, Reso: resolution.

Table 7-14 Optimal coding parameter set for Foreman

Ref.	SR	FME	RDO	SG	QP	IP	Dt	Reso.
1	1	3	1	1	24	12	1	1

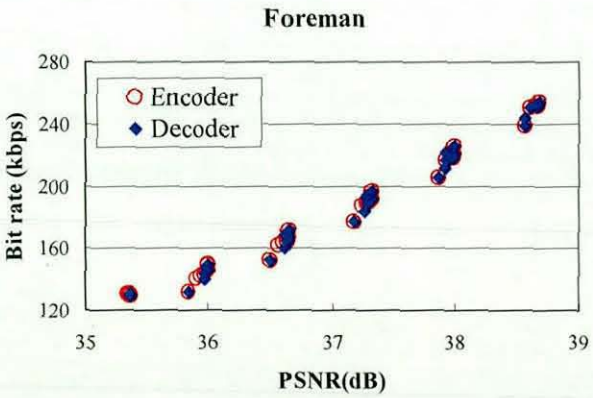


Figure 7-9 Rate-distortion curves of encoder and decoder for Foreman

7.6 Conclusions

In this chapter, a joint C-M-R-D optimization framework for a H.264 video CODEC (JM 10) has been proposed based on a detailed comprehensive performance analysis (see Chapter 6) of the H.264 CODEC. The framework adopts an evolutionary algorithm, NSGA-II, as the multi-objective optimization algorithm, and is designed for solving the joint C-M-R-D optimization problem at both the encoder and decoder based on the decoder's capabilities/limitations such as available memory and bandwidth. It produces a set of optimal coding parameters that can be used for encoding and decoding video sequences. These optimal coding parameters can minimize the computational complexity, memory utilization and rate at CODEC while achieving the maximum visual quality. According to the simulation results, the framework can yield Pareto-optimal or near Pareto-optimal solutions. In other words, it can produce an optimal coding parameter set for a video sequence. Although the framework has not considered network factors such as transmission delay and packet loss, it provides practical guidelines for the design and performance optimization of video communication systems under computing resources constraints.

Chapter 8 Conclusions and Future Work

8.1 Summary

This chapter summarizes the original contributions made by the thesis to the research area of video coding. It also highlights possible future directions of research related to the work presented.

This thesis has presented the design, implementation and performance analysis of a novel object-based extension to a standard H.264 video CODEC. It has also proposed a multi-objective optimization framework that can be used in the parameter-based performance optimization of a standard compliant H.264 CODEC. The practical relevance of these extensions/improvements has also been discussed.

In Chapter 4, a novel Shape Adaptive Integer Transform (SA-IT) and associated quantization procedures were proposed with the aim of being used in the inclusion of object-based coding in H.264. It was shown that the SA-IT calls for novel design and implementation considerations as compared to the design and implementation considerations of SA-DCT used within MPEG-4 and IT used within H.264. The main features of the SA-IT algorithm is that (1) it provides the ability of coding arbitrary shaped video objects, (2) the forward and inverse transforms can be implemented by only using simple additions and shifts without the need for multiplications, (3) it minimizes computational complexity by using 16-bit arithmetic operations and (4) the introduction of a quantization table look-up strategy that can avoid divisions at quantization. However, 2-D SA-IT causes the issue of having to code transformed sub-coefficients individually, a matter discussed in detail in Section 5.4. In the proposed research this issue has been solved by only applying 1-D (i.e. vertical) SA-IT, which only gives rise to a marginal penalty in the bit rate.

Chapter 5 presented the design, implementation and performance analysis of an object-based coding extension to the Baseline profile of H.264 standard, based on the SA-IT proposed in Chapter 4. It was shown that the slice group structure of the standard H.264 CODEC needs to be modified and extended for the effective implementation of the proposed object-based coding framework. Furthermore a novel shape coding algorithm based on the MPEG-4 shape coding methodology was also proposed for the purpose of object shape coding. Experimental results were provided to prove that the proposed shape coding algorithm is more efficient compared to that used within the MPEG-4 standard.

The inclusion of object-based coding in H.264 enables its enhanced original features such as the superior rate distortion performance to be effectively utilized within new application domains such as security and surveillance systems and highly bandwidth constrained communications applications that will benefit at least from the transmission of given ROIs at higher quality. The additional functionality above was provided at no extra cost to the CODEC's rate-distortion performance. Experimental results were provided to justify this claim (see Chapter 4 and Chapter 5). The proposed CODEC has further being conceptually compared with the upcoming H.264 Scalable Video Coding (SVC) extension. It has been revealed that H.264 SVC can not handle the independent coding of arbitrarily shaped regions, but only regions made out of rectangular blocks. However, the proposed CODEC can only work on the assumption that the binary alpha maps of a video sequence are known.

A detailed performance analysis of the H.264 baseline profile CODEC (JM 10) was presented in Chapter 6. The aim of the analysis was to identify those coding parameters which have a significant effect on computational complexity, memory utilization, rate and distortion. The above analysis was performed on both the encoder and the decoder. A total of 9 parameters (see Table 6-20) have been identified as the main contributors to computational complexity, memory utilization, rate and distortion at the encoder, while 5 parameters were identified as the main contributors at the decoder. These analysis results (i.e. significant coding parameters) were subsequently used in the joint multi-objective optimization of a H.264 CODEC.

In Chapter 7, a multi-objective optimization framework for the H.264 baseline profile CODEC (JM 10) was proposed. The framework was used to achieve a joint optimization of computational complexity, memory usage, rate and distortion based on the performance analysis results of Chapter 6. A genetic algorithm was used in the process. The most important advantage of the proposed optimization framework is that it produces an optimal or near optimal coding parameter set for encoding and decoding video sequences. It was found that the optimum parameter selection was generally dependent on the source video content. However further analysis revealed that two videos with similar characteristics/content has similar optimal coding parameter sets.

In practice the proposed multi-objective optimization strategy can be used to support a number of different application scenarios. If a video is to be made available for streaming, knowledge about the decoder constraints (e.g. memory, bandwidth and CPU constraints) can be relayed to the encoder via a feedback path. The encoder knowing its own limitations in computational power and memory can therefore use a pre-calculated look up table from which an optimal coding parameter set can be obtained. This parameter set can be subsequently be used in optimum coding. Further different application domains may have different constraints. For example if a mobile handset is to be used as the video decoder, the decoder computational power and memory capacity will be constrained In such an application the rate-distortion can be minimized under constraints of computational power and memory utilization. In another example such as digital TV broadcast, the constraints will be the rate and distortion as the bandwidth will be allocated a specific value which cannot be exceeded and the distortion will be specified by subjective quality requirements governing digital television transmission. In this application decoder computational cost and memory utilization will not be a constraint, but will have to be minimized. The proposed framework can therefore be used in the multi-objective optimization of any of the above application scenarios. It is noted that the application example used in Chapter 7 to demonstrate the framework's use in multi objective optimization only specifically considers a scenario where the rate and memory is constrained and

distortion and computational cost require to be minimized. This flexibility of the proposed framework enables its widespread use in different application domains.

It is noted that the proposed optimization framework did not consider involving network factors such as transmission delay and packet loss etc, which are important in practical applications. Therefore the application of the proposed framework in an end-to-end optimization of a video encoding, transmission and decoding system is limited. Rather the proposed framework provides a set of theoretical guidelines for the multi-objective performance optimization methodology of a video CODEC, under resource constraints.

8.2 Future Work

The research presented in thesis has resulted in a number of original contributions towards the functionality extension and performance optimization of a standard H.264 CODEC. A number of possibilities exist for the future extension and enhancement of the proposed ideas. They can be listed as follows:

1. The object-based coding of video requires the identification of objects of interest. The experiments used in this thesis have either user manual object identification or have used given object shape templates that have been provided with the test video sequences. It should be possible to integrate an automatic object extraction algorithm into the CODEC as a pre-processing stage that detects video objects (i.e. binary alpha maps) automatically prior to encoding. It is noted that depending on the application the accuracy required in the object shape identification can vary.

2. The extension of the object-based coding ideas to the upcoming H.264 Scalable Video Coding (SVC) standard is seen as a possible further direction of research. The addition of object-based scalability to H.264 SVC can further improve its practical relevance. This work is to be carried

out in the near future under a DTI, UK funded project that is looking at the selective compression of CCTV video in surveillance applications.

3. The reduction of computational cost of object-based coding is a further direction of investigation. The additional computational cost of object identification at individual frame levels, shape coding, texture coding or arbitrarily shaped video objects etc., leads to computationally additional cost that will make real time software only implementation of the algorithms close to impossible. Therefore hardware implementation options using FPGA should be considered as an alternative. This work has a direct practical relevance in industry. This work is to be carried out with industrial collaboration in the near future.

4. The optimization framework can consider two further aspects of improvement. First of all, it should be possible to group video sequences of similar statistical nature (motion, texture etc.), enabling them to share the same objective function, without significant data fitness mismatches. This will largely simplify the practical use of the system as optimal coding parameter set data can be shared by similar videos, rather than having to be individualised.

5. The overall performance of a video CODEC is not only dependent on the encoder and decoder constraints/limitations. The transmission channel properties other than the bandwidth can severely impact the overall performance. Thus the proposed optimization framework should be further extended to include channel constraints such as delay and packet loss so that it can ideally be used in real network environment.

The thesis has provided two significant contributions to video coding, in particular to the functionality enhancement and performance optimization of the latest video coding standard H.264/AVC. Experimental results and detailed analysis have been provided to support the novel concepts/ideas. The original contributions of this thesis

in the area of object-based video coding have been published at a number of conferences (see Appendix C). Further two research articles on the proposed multi-objective optimization framework, have been submitted for publication.

References

[1] ITU-T and MPEG, "Advanced video coding for generic audiovisual services," Tech. Rep. ITU-T Recommendation H.264 and ISO/IEC 14496-10, May 2003.

[2] Z. Yafan, "Complexity Management for Video Encoders," March. 2004.

[3] ISO/IEC, "Information technology—Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s-part 2: Video," Tech. Rep. ISO/IEC 11172-2 (MPEG-1 Video), 1993.

[4] ITU-T, "Generic coding of moving pictures and associated audio: Part 2 video," Tech. Rep. ITU-T Recommendation H.262 ISO/IEC 13818-2 MPEG-2 video, 1994.

[5] ISO/IEC, "Coding of audio-visual objects– part 2: Visual," Tech. Rep. ISO/IEC 14496-2 (MPEG-4 Part 2: Visual), 2001.

[6] N. Day and J. M. Martinez, "Introduction to MPEG-7," *ISO/IEC JTC1/SC29/WG11,* vol. 3751, 2000.

[7] ISO/IEC, "MPEG-21 overview," Tech. Rep. ISO/IEC JTC1/SC29/WG11/N5231, Oct. 2002.

[8] ITU-T, "Video CODEC for audiovisual services at px64 kbits," Tech. Rep. ITU-T Recommendation H.261, March 1993.

[9] ITU-T, "ITU-T H.262 (MPEG-2 video) information technology—Generic coding of moving pictures and associated audio information: Video," ISO/IEC, Tech. Rep. 13818-2, 1995.

[10] ITU-T, "Video coding for low bit rate communication," Tech. Rep. ITU-T Recommendation H.263, 1998.

[11] I. E. G. Richardson, *H. 264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia.* John Wiley & Sons, 2003.

[12] I. Wikimedia Foundation, "http://en.wikipedia.org/wiki/Source_Input_Format," vol. Accessed in 2007, December. 2006.

[13] K. Language, "Wikipedia, the free encyclopedia," *Retrieved June,* vol. 19, pp. 2006.

[14] T. Wiegand, G. Sullivan, G. Bjntegaard and A. Luthra, "Overview of the H. 264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 13, pp. 560-576, 2003.

[15] I. T. U. T. S. Sector, "ITU-T Recommendation H.264. Advanced video coding for generic audiovisual services," *ITU-T Rec.H,* pp. 14496-14410, 2005.

[16] T. Wiegand and G. Sullivan, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H. 264| ISO/IEC 14496-10 AVC)," *JVT-G050r1, Geneva, Switzerland, may,* 2003.

[17] H. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky, "Low-complexity transform and quantization in H. 264/AVC," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 13, pp. 598-603, 2003.

[18] A. Hallapuro and M. Karczewicz, "Low Complexity Transform and Quantization–Part I: Basic Implementation," *JVT Document JVT-B038, February,* 2001.

[19] H. Musmann, M. Hoetter and J. Ostermann, "Object-oriented analysis-synthesis coding of moving images." *Signal Process Image Commun,* vol. 1, pp. 117-138, 1989.

[20] N. Brady, "MPEG-4 standardized methods for the compression of arbitrarily shaped video objects," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 9, pp. 1170-1189, 1999.

[21] N. Brady and F. Bossen, "Shape compression of moving objects using context-based arithmetic encoding," *Signal Process Image Commun,* vol. 15, pp. 601-617, 2000.

[22] ISO/IEC, "Information technology—Coding of audio-visual objects part 2: Visual," Tech. Rep. 14492-2 (MPEG-2 Video), July 2000.

[23] I. H. Witten, R. M. Neal and J. G. Cleary, "Arithmetic coding for data compression," *Commun ACM,* vol. 30, pp. 520-540, 1987.

[24] T. Sikora, "Low complexity shape-adaptive DCT for coding of arbitrarily shaped image segments," *Signal Process Image Commun,* vol. 7, pp. 6, 1995.

[25] A. Kaup and S. Panis, "On the Performance of the Shape Adaptive DCT in Object-Based Coding of Motion Compensated Difference Images," *Proc.of 1997 Picture Coding Symposium,* pp. 653-657, 1997.

[26] A. Kaup, "Object-based texture coding of moving video in MPEG-4," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 9, pp. 5-15, 1999.

[27] T. Ebrahimi, "MPEG-4 video verification model version 11.0," *ISO/IEC JTC1/SC29/WG11, MPEG98,* vol. N2172, 1998.

[28] J. W. Yi, S. J. Cho, W. J. Kim, S. D. Kim and S. J. Lee, "A new coding algorithm for arbitrarily shaped image segments," *Signal Process Image Commun,* vol. 12, pp. 231-242, 1998.

[29] G. Z. Shen and B. M. L. Liou, "Arbitrarily shaped transform coding based on a new padding technique," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 11, pp. 67-79, 2001.

[30] S. Li and W. Li, "Shape-adaptive discrete wavelet transforms for arbitrarily shaped visual object coding," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 10, pp. 725-743, 2000.

[31] I. Donescu, O. Avaro and C. Roux, "A comparison of efficient methods for the coding of arbitrarily shaped image segments," *Proc.Picture Coding Symp,* pp. 181–186, 1996.

[32] J. ShinHaeng, P. JungWook and K. ShinDug, "Optimization of Memory Management for H.264/AVC Decoder," vol. 1, pp. 65-68, 2006.

[33] T. Wiegand, M. Lightstone, D. Mukherjee, T. Campbell and S. Mitra, "Rate-distortion optimized mode selection for very low bit ratevideo coding and the emerging H. 263 standard," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 6, pp. 182-190, 1996.

[34] K. H. Yang, A. Jacquin and N. S. Jayant, "A normalized rate-distortion model for H. 263-compatible codecs and its application to quantizer selection," *Proc.Int.Conf.Image Processing,* vol. 2, pp. 41–44, 1997.

[35] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Process. Mag.,* vol. 15, pp. 74-90, 1998.

[36] Z. He and S. Mitra, "A unified rate-distortion analysis framework for transform coding," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 11, pp. 1221-1236, 2001.

[37] T. Wiegand and B. Girod, "Lagrange multiplier selection in hybrid video coder control," *Image Processing, 2001.Proceedings.2001 International Conference on,* vol. 3, 2001.

[38] T. Stockhammer, D. Kontopodis and T. Wiegand, "Rate-Distortion Optimization for JVT/H. 26L Video Coding in Packet Loss Environment," *12th International Packet Video Workshop (PV 2002), may,* 2002.

[39] K. Takagi, Y. Takishima and Y. Nakajima, "A study on rate distortion optimization scheme for JVT coder," *Proceedings of SPIE,* vol. 5150, pp. 914-923, 2003.

[40] S. Ma, W. Gao and Y. Lu, "Rate-distortion analysis for H. 264/AVC video coding and its application to rate control," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 15, pp. 1533-1544, 2005.

[41] D. Kwon, P. Agathoklis and P. Driessen, "Performance and computational complexity optimization in a configurable video coding system," *Wireless Communications and Networking, 2003.WCNC 2003.2003 IEEE,* vol. 3, 2003.

[42] J. Zhang, Y. He, S. Yang and Y. Zhong, "Performance and complexity joint optimization for H. 264 video coding," *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on,* vol. 2, 2003.

[43] J. Stottrup-Andersen, S. Forchhammer and S. Aghito, "Rate-distortion-complexity optimization of fast motion estimation in H. 264/MPEG-4 AVC," *Image Processing, 2004.ICIP'04.2004 International Conference on,* vol. 1, 2004.

[44] Y. Hu, Q. Li, S. Ma and C. C. J. Kuo, "JOINT RATE-DISTORTION-COMPLEXITY OPTIMIZATION FOR H. 264 MOTION SEARCH," *Internation Conference on Multimedia & Expo.(ICME 2006), Toronto,* 2006.

[45] C. Kannangara, I. Richardson, M. Bystrom, J. Solera, Y. Zhao, A. MacLennan and R. Cooney, "Complexity reduction of H. 264 using Lagrange optimization methods," 2005.

[46] W. Pu, Y. Lu and F. Wu, "Joint Power-Distortion Optimization on Devices with MPEG-4 AVC/H. 264 Codec," *Communications, 2006.ICC'06.IEEE International Conference on,* vol. 1, 2006.

[47] Z. Chen and I. Ahmad, "Power-rate-distortion analysis for wireless video communication under energy constraints," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 15, pp. 645-658, 2005.

[48] I. Ismaeil, A. Docef, F. Kossentini and R. Ward, "A computation-distortion optimized framework for efficient DCT-based video coding," *Multimedia, IEEE Transactions on,* vol. 3, pp. 298-310, 2001.

[49] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems,* pp. 50-61, 2001.

[50] X. Li, E. Edirisinghe and H. Bez. Shape adaptive integer transform for coding arbitrarily shaped objects in H. 264/AVC. Presented at VCIP 2006.

[51] Karsten Suhring. 2006, H.264 reference software: JM 10.

[52] C. Toklu, A. M. Tekalp and A. T. Erdem, "Semi-Automatic Video Object Segmentation in the Presence of Occlusion," *IEEE TRANSACTIONS ON*

CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, vol. 10, pp. 625, 2000.

[53] T. Meier and K. Ngan, "Automatic segmentation of moving objects for video object planegeneration," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 8, pp. 525-538, 1998.

[54] G. Langdon Jr and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Commun.,* 1981.

[55] H. Schwarz, D. Marpe and T. Wiegand, "Scalable Extension of H. 264/AVC," *M10569, Munich, March,* 2004.

[56] H. Schwarz, D. Marpe and T. Wiegand, "Overview of the Scalable H. 264/MPEG4-AVC Extension," *IEEE Int'l.Conf.Image Processing, Atlanta, GA, Oct,* 2006.

[57] M. Horowitz, A. Joch, F. Kossentini and A. Hallapuro, "H. 264/AVC baseline profile decoder complexity analysis," *Circuits and Systems for Video Technology, IEEE Transactions on,* vol. 13, pp. 704-716, 2003.

[58] K. Hari and F. Borko, "Complexity Estimation of the H.264 Coded Video Bitstreams," *The Computer Journal,* vol. 48, pp. 504-513, 2005.

[59] Z. Jun, Y. Xiaoquan, L. Nam and S. Weijia, "Bit rate distribution analysis for motion estimation in H.264," *ICCE 2006,* pp. 483-484, 7-11 Jan. 2006.

[60] H. CHENG, M. ISNARDI and A. KOPANSKY, "Macro-block based mixed resolution video compression system," *MACRO-BLOCK BASED MIXED RESOLUTION VIDEO COMPRESSION SYSTEM,* 2006.

[61] I. Wikimedia Foundation , "http://en.wikipedia.org/wiki/Instruction_cycle," vol. Accessed in 2007, 22 June. 2007.

[62] X. Xiaozhong and H. Yun, "Comments on motion estimation algorithms in current JM software," 21 October, Tech. Rep. JVT-Q089, 2005.

[63] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan, "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II," *Proceedings of the Parallel Problem Solving from Nature VI Conference,* pp. 849-858, 2000.

[64] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons, 2001,

[65] Kanpur Genetic Algorithms Laboratory. (Http://www.iitk.ac.in/kangal/codes.shtml. *Accessed in 2007(2 September)*

[66] I. The MathWorks. (2007, Http://www.mathworks.com, *Accessed in 2007(22 August)*

[67] Zitzler, E., "Evolutionary algorithms for multiobjective optimization: Methods and applications", Master's thesis, Swiss federal Institute of technology (ETH), Zurich, Switzerland (1999).

[68] Zhibo Chen, Peng Zhou, Yun He, "Fast Integer Pel and Fractional Pel Motion estimation in for JVT", JVT-F017r1.doc, December, 2002.

[69] Zhibo Chen, Peng Zhou, Yun He, "Fast Motion Estimation for JVT", JVT-G016, March, 2003.

[70] Jianfeng Xu, Ping Yang, Yun He,"Modification of Fast Motion Estimation", JVT-J027, December, 2003.

[71] Xiaoquan Yi, Jun Zhang, Nam Ling, and Weijia Shang, "Improved and simplified fast motion estimation for JM", JVT-P021, July, 2005.

Appendix A: Objective Functions of Video Sequences for Encoder

1. Objective Functions of Computational Complexity

Terms and coefficients of Foreman

Coefficients	x_1	x_2	x_3	x_4	x_9
2.1639500000000	0	0	0	0	0
-1.7264615762201	0	0	0	0	1
2.2984189258933	1	1	0	0	0
-1.0048139336747	0	0	0	1	1
-2.4477332625048	1	1	1	0	0
0.4609369423541	0	0	0	2	0
-2.6999195302205	1	0	0	0	2
-0.2545023644350	0	1	0	0	2
-0.1219815153486	0	0	1	0	2
0.0869053436663	1	1	2	0	0
2.9420320911703	1	0	1	0	2
0.0681195798671	0	1	1	0	2
-1.0564765050174	1	0	2	0	2
-0.0200436738811	2	1	2	0	1
0.0511330321821	2	1	1	0	2
1.3708908736189	1	1	2	0	2
-0.0176222885209	2	2	0	0	2
0.1624456898017	0	0	1	0	3
0.3846404082604	0	0	0	1	3
0.2801060896166	1	1	3	0	0
1.7876106709402	1	1	0	0	3
-0.4434475235357	1	1	3	0	1
-1.7835619625525	1	1	1	0	3
0.0014769510112	2	0	3	0	0
-0.0256489256699	0	0	2	0	3
0.0003964872774	3	2	1	0	0
0.0024595393335	2	1	3	0	0
-0.0190315488655	2	1	0	0	3
-0.0018384118293	2	0	3	0	1
0.1255801613352	1	0	3	0	2
-0.0079777033771	0	0	0	5	1

Terms and coefficients of Coastguard

Coefficients	x_1	x_2	x_3	x_4	x_9
-0.1353090000000	0	0	0	0	0
0.6223717633215	0	0	1	0	0
-0.7639253069740	0	0	1	0	1
0.2103721141288	0	0	0	1	1
-2.2857397699410	1	1	1	0	0
-0.0188753001130	2	0	0	0	0
0.0338375348325	0	0	2	0	0
-2.4962044576467	1	0	0	0	2
3.8219157760919	1	1	0	0	2
2.9604016016292	1	0	1	0	2
0.1638829083457	1	1	2	0	1
-1.0557275018342	1	0	2	0	2
1.1088677292850	1	1	2	0	2
0.0018957821931	3	0	1	0	0
0.2430339193185	1	1	3	0	0
-0.2170583257694	1	1	0	0	3
-0.3824434031035	1	1	3	0	1
-1.4752429075846	1	1	1	0	3
-0.0000439117923	3	0	0	1	2
0.1209490596199	1	0	3	0	2
0.0500235607222	0	0	0	4	0
-0.0000544259430	4	0	2	0	0
-0.0010097186680	0	0	4	0	2
0.0135237743140	0	0	2	0	4
0.0466825571904	0	0	0	2	4
-0.0174957239122	0	0	0	5	1

Terms and coefficients of Mother & Daughter

Coefficients	x_1	x_2	x_3	x_4	x_9
-0.2041517956412	0	0	0	0	0
0.3165646733353	0	0	1	0	0
-0.3385939801707	0	0	1	0	1
0.2750319893053	0	0	0	1	1
-2.3004076262064	1	1	1	0	0
-2.5655364292056	1	0	0	0	2
3.7864693852438	1	1	0	0	2
2.9953569554126	1	0	1	0	2
-1.0795540124119	1	0	2	0	2
1.3794327907866	1	1	2	0	2
0.3057298569872	1	1	3	0	0
-0.0293689868774	1	1	0	0	3
-0.4582364962363	1	1	3	0	1
-1.7154566609996	1	1	1	0	3
0.1250301095724	1	0	3	0	2
0.0338324704338	0	0	0	4	0
0.0038274781955	0	0	2	0	4
0.0316089895731	0	0	0	2	4
-0.0124814415485	0	0	0	5	1

Terms and coefficients of Mobile

Coefficients	x_1	x_2	x_3	x_4	x_9
-0.4294162517851	0	0	0	0	0
0.7549975429279	0	0	1	0	0
-0.6718366608609	0	0	1	0	1
0.3288496101115	0	0	0	1	1
-2.1847325000571	1	1	1	0	0
-0.0222887445793	0	0	2	0	0
-2.4677417281324	1	0	0	0	2
3.6756331710827	1	1	0	0	2
2.8447847458364	1	0	1	0	2
-1.0003533778134	1	0	2	0	2
1.2833452606370	1	1	2	0	2
0.2873744854059	1	1	3	0	0
-0.0774383491485	1	1	0	0	3
-0.4280857055542	1	1	3	0	1
-1.5956803318535	1	1	1	0	3
0.1133548386039	1	0	3	0	2
0.0578099580610	0	0	0	4	0
0.0082448774314	0	0	2	0	4
0.0535867026846	0	0	0	2	4
-0.0206017085128	0	0	0	5	1

2. Objective Function of Memory Utilization

Terms and coefficients of objective function of memory

Coefficients	x_1	x_2	x_3	x_5	x_9
-11.2803946546805	0	0	0	0	0
0.0241483793543	0	0	1	0	0
17.0167003699319	0	0	0	0	1
2.6706783005441	1	0	0	0	1
-0.0161330025392	0	1	0	0	1
-0.2828079894569	1	1	1	0	0
0.0270330548904	0	1	1	0	1
0.0079895803368	0	2	0	0	0
0.1003377607644	0	0	2	0	0
0.0002126063886	0	2	1	0	0
-0.1611470391607	0	0	2	0	1
0.2665103085206	0	0	1	0	2
0.5924722824311	0	0	0	1	2
-4.3299779585142	1	2	1	0	0
0.0993202065711	1	1	2	0	0
-0.0118537056107	0	1	2	0	1
0.0184875420721	1	0	1	0	2
-0.0001332936693	0	1	0	1	2
-0.0000124933203	0	2	1	1	1
1.4984380370870	1	2	2	0	0
3.7491009439705	1	3	0	0	0
-0.0073721560041	1	0	3	0	0
-0.0111768461643	1	1	3	0	0
0.0048165979422	1	0	3	0	1
0.0015531545831	0	1	3	0	1
0.0000109293241	0	3	0	1	2
-0.1664527590542	1	2	3	0	0
-0.0042000571827	1	0	2	0	3
-1.1929322197609	1	4	0	0	0
-0.0000181213598	0	4	0	0	1
0.0007662652350	1	0	4	0	0
0.1258508224310	1	5	0	0	0
0.0000004765936	0	0	5	1	0
0.0000570994767	0	0	5	0	1
0.0162173407242	1	0	0	0	5

3. Objective Functions of Rate

Terms and coefficients of Foreman

Coefficients	x_6	x_7	x_8	x_9
2.2435143016075	0	0	0	0
-0.0083388914222	1	1	0	0
-0.0012013400334	2	0	0	0
0.0012664901953	0	1	1	2
-0.0000011607879	2	2	0	1
0.0000141554382	3	0	0	0
0.0002496691641	1	3	0	0
0.0006989071614	0	1	0	3
0.0000407452993	1	1	3	0
-0.0008498970116	1	0	2	3
0.0000000212768	3	2	0	2
0.0003966735593	1	0	3	3
0.0000000356334	4	1	0	0
-0.0000356208983	1	4	0	0
-0.0000044050569	0	4	1	1
-0.0000000029999	4	2	0	0
-0.0000000008078	4	2	0	1
0.0000000001379	4	3	0	0
0.0000000034216	4	0	0	3
0.0000020245849	1	5	0	0
-0.0000000000242	5	1	1	0
0.0000002176532	0	5	1	1
-0.0000646541968	1	0	1	5
-0.0000000002487	2	5	0	0
-0.0000000417589	1	6	0	0
-0.0000000000004	7	0	0	0
0.0043919437782	0	0	0	7

Terms and coefficients of Coastguard

Coefficients	x_6	x_7	x_8	x_9
2.0333961674744	0	0	0	0
-0.0841058555285	0	1	0	0
0.1070654275799	0	1	0	1
-0.0008107864627	2	0	0	0
0.1524712016774	0	0	0	2
-0.0003412278436	2	1	0	0
0.0007856339313	1	2	0	0
-0.0396675698067	0	2	0	1
-0.0000218303527	1	1	1	2
0.0000535583419	2	2	0	0
-0.0002014530839	1	2	2	0
0.0000146247602	2	2	2	0
0.0000111713244	3	0	0	1
0.0012428518098	0	3	1	0
-0.0000679048609	1	3	1	0
-0.0000050648441	2	3	0	0
0.0079743127476	0	3	0	2
-0.0000008093696	3	2	1	0
0.0000000055518	3	2	0	1
0.0000000175241	3	3	1	0
-0.0000000644805	4	0	1	0
-0.0007468784879	0	4	0	1
0.0000000007324	4	1	1	1
-0.0000001538453	4	0	0	2
0.0000003573058	2	4	0	0
-0.0000186586542	0	4	2	0
0.0000000029953	4	2	1	0
-0.0000000528699	2	4	1	0
0.0000012781595	1	4	2	0
0.0000007652952	2	1	4	0
0.0000000083870	4	0	3	0
-0.0000000003712	3	4	0	0
-0.0010865634277	0	3	0	4
0.0000002286618	0	5	0	0
0.0000000025776	5	1	0	0
0.0000352455425	0	5	0	1
-0.0000000000867	5	1	0	1
0.0000073802999	1	1	0	5
0.0000000016059	5	0	0	2
-0.0000000068107	2	5	0	0
-0.0000000000262	6	1	0	0
-0.0000006360545	0	6	0	1

Terms and coefficients of Mother & Daughter

Coefficients	x_6	x_7	x_8	x_9
2.5476918816222	0	0	0	0
-0.0410481765732	1	0	0	0
-0.0493028266920	0	2	0	0
-0.0032472726700	1	2	0	0
0.0000398149865	2	1	1	0
-0.0000144011157	1	2	1	1
0.0000012759856	2	1	2	1
0.0180260395138	0	3	0	0
0.0008014902210	1	3	0	0
0.0000000122287	2	3	1	1
-0.0000000973232	3	1	3	0
0.0002422268119	0	1	3	3
-0.0027153649640	0	4	0	0
0.0000000135165	4	1	0	0
-0.0000869156354	1	4	0	0
-0.0000252714966	1	1	4	1
-0.0000000017686	4	2	0	0
0.0000000000665	4	3	0	0
0.0002083877407	0	5	0	0
0.0000044493679	1	5	0	0
-0.0000000002865	2	5	0	0
-0.0000000000542	6	0	0	0
-0.0000080760452	0	6	0	0
-0.0000000868578	1	6	0	0
0.0000000000009	7	0	0	0
0.0000001256033	0	7	0	0
0.0024184451756	0	0	0	7

Terms and coefficients of Mobile

Coefficients	x_6	x_7	x_8	x_9
1.6921719044047	0	0	0	0
0.0140719987585	1	0	0	0
0.5153865375857	0	0	0	1
-0.0294840209947	1	1	0	0
0.0292600025591	1	1	1	0
-0.0007461124035	2	0	0	0
-0.0016400324217	0	2	0	1
-0.0007389478888	2	1	2	0
0.0002614447708	1	3	0	0
-0.0209470940089	0	1	3	0
0.0022021870172	0	1	0	3
0.0000009242727	3	1	0	1
-0.0000001660109	3	2	0	0
0.0000005074097	2	3	0	0
0.0000167029577	0	3	0	2
-0.0000000489926	3	2	1	0
0.0000531961312	3	1	2	0
0.0000000046372	3	3	1	0
-0.0000114312949	3	1	3	0
-0.0000385066310	1	4	0	0
-0.0000013261416	4	1	1	0
-0.0000000391800	4	1	0	1
-0.0000000029138	4	0	1	1
-0.0000000138219	2	4	1	0
0.0000000136669	5	1	0	0
0.0000021733192	1	5	0	0
0.0000000083547	5	1	1	0
0.0000000004200	5	1	0	1
0.0000000218650	1	5	1	0
-0.0000000001379	6	1	0	0
0.0000000000044	6	0	0	1
-0.0000000456600	1	6	0	0
-0.0000000113412	0	6	1	0
0.0000000012937	0	7	0	0

4. Objective Functions of Distortion

Terms and coefficients of Foreman

Coefficients	x_6	x_7	x_8	x_9
65.1588589406607	0	0	0	0
-1.4922287197157	1	0	0	0
-0.3750129163973	0	1	0	0
-0.3928929231829	0	0	1	0
-0.5796243983919	0	0	0	1
0.0230362242037	1	0	1	0
0.0060342028260	1	1	1	0
0.0349453392479	2	0	0	1
0.0312966257288	0	2	1	0
-0.0100564976455	2	0	0	2
-0.0127834701557	0	2	2	0
-0.0002629879769	3	0	0	0
-0.0000009414790	3	1	2	0
-0.0000018978762	1	4	0	0
-0.0000002771823	4	0	2	0
0.0000033407981	0	4	2	0
0.0000000078901	4	1	2	0
0.0000001703057	4	0	0	3
-0.0000086880378	3	0	0	4
0.0000000141076	5	0	1	0
-0.0000000000738	5	1	0	1
-0.0000000000070	5	2	0	0
0.0000000014449	2	5	0	0

Terms and coefficients of Coastguard

Coefficients	x_6	x_7	x_8	x_9
61.3600230412221	0	0	0	0
-1.0070409839406	1	0	0	0
-0.4386427303395	0	1	0	0
-0.1345095348741	0	0	1	0
0.1448303500236	0	0	0	1
0.0048249631790	1	1	1	0
0.0000590185186	2	0	0	1
0.0365005136265	0	2	1	0
-0.0142741998318	0	2	2	0
0.0001086189398	3	0	0	0
-0.0000003792590	3	1	2	0
-0.0000016570306	1	4	0	0
0.0000000971618	4	0	2	0
0.0000025932247	0	4	2	0
-0.0000000062894	5	0	1	0
0.0000000009285	2	5	0	0

Terms and coefficients of Mother & Daughter

Coefficients	x_6	x_7	x_8	x_9
66.7450371247196	0	0	0	0
-0.8710008865799	1	0	0	0
-0.1061563830113	0	1	0	0
-6.9544788960645	0	0	0	1
0.0361409838943	0	0	2	0
0.0000572121417	1	2	0	1
0.0028856776789	0	2	2	0
0.0001305502472	3	0	0	1
-0.0000000201228	3	1	2	1
-0.0000000191904	2	3	0	2
-0.0000009841051	4	0	0	0
-0.0010576406229	2	0	0	4
-0.0000000173704	2	4	1	0
0.0000003845855	1	4	2	0
0.0000000000607	4	3	0	0
-0.0000466999534	0	3	4	0
0.0215628377216	1	0	0	5
-0.0000000003126	5	0	1	1

Terms and coefficients of Mobile

Coefficients	x_6	x_7	x_8	x_9
55.0465698869160	0	0	0	0
-0.5743043025305	0	1	0	0
0.0049766743997	1	1	0	0
0.0033261834174	1	1	1	0
-0.0509530017537	2	0	0	0
0.0257377965568	0	2	0	0
-0.0494029683787	1	0	1	2
0.0026799892264	2	0	1	2
0.0011046138834	3	0	0	0
0.0868179165181	0	0	0	3
0.1010571718182	0	0	1	3
-0.0000513452131	3	0	1	2
-0.0000009030638	2	3	1	0
0.0000000336459	3	1	1	2
-0.0000074632336	4	0	0	0
0.0000003023367	4	0	1	2
0.0000000480302	2	4	1	0
-0.0000000561712	1	5	0	0
-0.0000004451743	0	5	1	0

Appendix B: Objective Functions of Video Sequences for Decoder

1. Objective Functions of Computational Complexity

Terms and coefficients of video sequences

Foreman			Coastguard		
Coefficients	x_2	x_6	**Coefficients**	x_2	x_6
0.0691277526841,	0	0	-0.0099169593695	0	0
0.0048561465279,	0	4	0.0847809204899	0	1
-0.0000000026428,	5	4	0.0003196623694	1	3
0.0000000191421	4	5	-0.0000183393317	2	3
			0.0000000010837	5	3
Mother & Daughter			**Mobile**		
Coefficients	x_2	x_6	**Coefficients**	x_2	x_6
0.0105819326471	0	0	0.0722232781428	0	0
0.0677158006827	0	1	0.0009873393801	1	2
-0.0049860046833	1	1	-0.0000900136367	2	2
0.0011226064433	2	1	0.0000027543944	3	2
-0.0001191243457	3	1	0.0060949914377	0	4
0.0000059530172	4	1			
-0.0000001129254	5	1			

2. Objective Functions of Memory Utilization

Terms and coefficients of memory utilization

Coefficients	x_1	x_3	x_6
-0.5239030496124	0	0	0
0.8034257057108	0	0	1
0.0723658211318	1	0	2
-0.0000002131269	5	0	1
0.0000179625956	1	0	5
0.0000000766780	6	0	0

Appendix C: List of Publications

The publications related to the work presented in this thesis are listed below.

Accepted conference papers:

1. X. Li, E. Edirisinghe and H. Bez. "Extensions to H.264 Baseline Profile Encoder Towards Video Object Coding", VIE 2007.

2. X. Li, E.A Edirisinghe and H.E. Bez, "Selective Compression of Video with H.264/AVC", Proceedings of the 6th IASTED on Visualization, Imaging, and Image processing, pp 579-584, 2006.

3. X. Li, E. Edirisinghe and H.Bez, 2006, "Shape adaptive integer transform for coding arbitrarily shaped objects in H.264/AVC", VCIP 2006, Vol. 6077, C1-C10.

Submitted papers:

1. X. Li, E. Edirisinghe and H. Bez, "Method for Coding Arbitrarily Shaped Video Objects in H.264/AVC", submitted to IEEE Transactions on Multimedia, on 19 April, 2007.

2. X. Li, E. Edirisinghe and C. Grecos, "Multi-objective Optimization of a H.264/AVC Video CODEC", submitted to the 2008 International Congress on Image and Signal Processing (CISP2008), on 26 Oct. 2007.

Paper under preparation:

1. X. Li, E. Edirisinghe and C. Grecos, "A Multi-objective Optimization framework for Video CODECs", will be submitted to IEEE Transactions on Communications.