Error-Resilient Coding Tools In MPEG-4

By

CHENG Shu Ling

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF THE MASTER OF PHILOSOPHY

DIVISION OF INFORMATION ENGINEERING

THE CHINESE UNIVERSITY OF HONG KONG

July 1997

Acknowledgement

I would like to thank my supervisor Prof. Victor K. W. Wei for his invaluable guidance and support throughout my M. Phil. study. He has given to me many fruitful ideas and inspiration on my research. I have been the teaching assistant of his channel coding and multimedia coding class for three terms, during which I have gained a lot of knowledge and experience. Working with Prof. Wei is a precious experience for me; I learnt many ideas on work and research from him.

I would also like to thank my fellow labmates, C.K. Lai, C.W. Lam, S.W. Ng and C.W. Yuen. They have assisted me in many ways on my duties and given a lot of insightful discussions on my studies. Special thanks to Lai and Lam for their technical support on my research.

Last but not least, I want to say thank you to my dear friends: Bird, William, Johnny, Mok, Abak, Samson, Siu-man and Nelson. Thanks to all of you for the support and patience with me over the years.

Abstract

Error resiliency is becoming increasingly important with the rapid growth of the mobile systems. The channels in mobile communications are subject to fading, shadowing and interference and thus have a high error rate. Channel coding cannot correct all the errors when the channel condition is too bad. Compressed data are very sensitive to errors; even a single bit error can cause significant quality degradation of the decoded data. Error resiliency is about to code the multimedia data in a way such that the effect of errors on quality degradation is minimized.

One of the targets identified by the MPEG-4 is to provide error robustness for the storage and communications in error-prone environment. The schemes being investigated include layering, resynchronization, data recovery and error concealment. This thesis will review some of these techniques.

One of the causes of the vulnerability of the compressed data is the use of variable length entropy codes. The decoder will lose the synchronization of codewords when there are errors in the compressed stream. The two main approaches discussed in this thesis to solve this problem are (1) Fixed length codes and (2) Self-synchronizable code.

Fixed length codes inherently eliminate the problem of codeword synchronization because the boundaries of codewords are at known positions. The errors are always confined within the corrupted codeword. Tunstall code and Lempel-Ziv code are examples of fixed length codes.

The self-synchronizable code has the property that the punctuation at the position in question can always be determined by observing *s* neighbouring symbols. This synchronization procedure can start anywhere within the bitstream and this property limits the propagation of errors.

The organization of this thesis is as follows: Chapter 1 gives the basic of JPEG and MPEG. Chapter 2 addresses the issue of error resiliency and describes the techniques studied by MPEG-4. Chapter 3 reviews the Tunstall and the Lempel-Ziv algorithm. Chapter 4 gives the construction procedure of a self-synchronizable code and its synchronizer. We then describe its application to image coding over noisy channels.

簡介

隨著無線電通訊系統的迅速發展，訊息之抗錯(Error Resiliency)能力亦變得更重要。已被壓縮的影視訊息十分容易受錯誤影響，即使旦一個位元的錯誤也能使解壓後的影像質素大大下降。

傳統上信道編碼被應用來糾錯，但當信道的環境十分差的時候，這些糾錯碼並不能糾正所有的錯誤。所謂訊息之抗錯能力就是指將錯信道錯誤帶來的影響儘量減少。

MPEG-4 其中一個目標是提供一些編碼工具來提高訊息之抗錯能力。正被研究的工具包括多層編碼(Layering)，重新同步(Resynchronizarion)，數據之復原(Data Recovery)和錯誤隱蔽(Error Concealment)。這篇論文將會介紹這些技術的一些基本概念。

壓縮後的訊息容易受錯誤影響，是因爲它們採用了不同長度的字碼。一但發生錯誤，解碼器便不能辨認各字碼的正確位置。這篇論文研究兩項方法來解決這問題：

一、固定長度字碼(Fixed length Code)
　　例子包括：Tuntsall 和 Lempel-Ziv 碼
二、自步碼(Self-Synchronizable Code)
　　自步碼的特性是我們只要檢查 s 個鄰近的字元，就能夠判斷某位置是否字碼的邊界。

本論文的大綱如下：
第一章爲影視壓縮技術的標準 JPEG 和 MPEG 的基本知識。
第二章對訊息之抗錯作深一步解釋，並介紹 MPEG-4 現時所作之研究。
第三章解釋 Tuntsall 和 Lempel-Ziv 碼和其應用。
第四章介紹如何建立自步碼和判斷字碼的邊界，也解釋它對錯誤時的表現。

Table of Contents

Chapter 1 Introduction

With the successful development of the Internet, visual communications has now become an essential part of our lives. World-Wide-Web (WWW) browsing, Internet phone, real time streaming audio and video are already maturely developed and very popular [1],[2]. These image and video applications always require some methods of compression to reduce the otherwise prohibitive demand on the bandwidth and storage space. To see the need for compression, consider a typical digital colour image with size 528 x 432 pixels at 3 bytes per pixel (1 byte for each red, green, blue colour component), it requires 684,288 bytes of storage space. To transmit this image over a 33600 bps modem, it takes about 2.7 minutes. The International Organization for Standardization (ISO) Joint Photographic Experts Group (JPEG) has developed an algorithm for coding still colour images. The JPEG algorithm can offer 20:1 compression ratio with almost no visual difference. This means that it reduces the required storage space of this image to about 34,000 bytes and transmission time to 8 seconds.

This chapter will give a brief description of the two most popular standards for coding image and video: JPEG and MPEG.

1.1 Image Coding Standard: JPEG

The JPEG committee defines four different modes of operations [4]:

Sequential DCT-based: The image is partitioned into many 8 x 8 blocks. Every block is transformed by the forward discrete cosine transform (FDCT). The blocks are then scanned from left to right and top to bottom and the transform coefficients are quantized and entropy coded in one scan.

Progressive DCT-based: This mode allows the decoder to produce a "rough" picture quickly and enhance it to full details by later scans. It is similar to sequential mode except the quantized coefficients are coded in several scans.

Lossless: The encoder entropy codes the difference between the input sample and its predicted value (based on the input sample's neighbor sample). The decoder will reproduce exactly the same digital input image.

Hierarchical: Another form of progressive coding. The input image is coded as a sequence of increasing spatial resolution frames. The first frame (lowest resolution) is coded by using either the sequential mode or progressive mode.

We shall look into more details of the sequential mode operation.

JPEG sequential DCT Codec

All JPEG DCT-based encoders begin with the partition of the digital input image into non-overlapping 8 x 8 blocks. These sample values (assume 8 bit precision, range from 0 to 255) are first level shifted by 128 so that they range from -128 to +127, then the blocks are transformed into the frequency domain by using the FDCT. The equations for 8 x 8 forward and inverse discrete cosine transform (IDCT) is given by:

FDCT:

$$S(v,u)=\frac{C(v)}{2}\frac{C(u)}{2}\sum_{y=0}^{7}\sum_{x=0}^{7}s(y,x)\cos[(2x+1)u\pi/16]\cos[(2y+1)v\pi/16]$$

IDCT:

$$s(y,x)=\sum_{v=0}^{7}\frac{C(v)}{2}\sum_{u=0}^{7}\frac{C(u)}{2}S(v,u)\cos[(2x+1)u\pi/16]\cos[(2y+1)v\pi/16]$$

where

$C(i)=1/\sqrt{2}$ *for* $i=0$

$C(i)=1$ *for* $i>0$

s(y,x) = 2-D sample values

S(v,u) = 2-D DCT coefficients

The coefficient at the most top left corner of a DCT block is proportional to the average of the spatial domain samples, thus called the *dc* coefficient. The other coefficients, called *ac* coefficients, represent increasingly higher frequencies component as they progress away from the dc coefficient. For most natural images, there is not so much drastic change of content within an 8 x 8 block. The DCT hence concentrates most of the energy of the input samples into the first few coefficients at the top left corner.

The next step is quantization, which makes JPEG a lossy coding algorithm. An 8 x 8 quantization matrix is used to reduce the amplitudes of the transform coefficients and increase the number of zero-valued coefficients. The quantization and dequantization is done by:

Quantization: $Sq_{vu} = round\left(\frac{S_{vu}}{Q_{vu}}\right)$

Dequantization: $R_{vu} = Sq_{vu} \, x \, Q_{vu}$

where

S_{vu} : DCT coefficients

Q_{vu} : quantization steps

Sq_{vu} : quantized coefficients

R_{vu} : reconstructed coefficients

Rounding is to the nearest integer. This round function incurs loss of information. Larger quantization steps can produce smaller amplitudes and more zeros, hence a higher compression ratio but poorer image quality. Many JPEG implementations control the compression ratio using a Q-factor, which simply just a scale factor applies to the quantization matrix elements. For example, the JPEG implementation released by the Independent JPEG Group (ILG) [19] uses the quantization matrix:

$$\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}$$

A quality factor Q, range from 0 to 100, allows the user to adjust between the compression ratio and quality. The number Q is transformed to a percentage number by the equations:

$$P = \begin{cases} 200 - 2 * Q & if \quad 50 \le Q < 100 \\ 5000 / Q & if \quad 0 < Q < 50 \end{cases}$$

Say, if Q = 75, i.e. P = 50%, all the quantization matrix entries will be halved. If Q = 50, then P = 100%, the quantization matrix will be used as-is.

As mentioned before, the dc coefficient corresponds to the average intensity of an 8 x 8 block and adjacent blocks tend to have similar average intensities. So JPEG codes only the difference of the dc component between adjacent blocks. (This is an example of differential pulse coded modulation DPCM). Each differential value is coded by a variable length bit string, which comprises a **SIZE** symbol followed by exactly **SIZE** bits to specify the value. For example, a dc difference of +6 needs **3** bits: **110** to represent and this is translated to a token **(SIZE=3,110)**. JPEG uses the Huffman code, to code the symbol **SIZE**. A typical Huffman table is shown below:

SIZE	**Difference**	**Code length**	**Huffman code word**
0	0	2	00
1	-1,1	3	010
2	-3,-2,2,3	3	011
3	-7,...,-4,4,...7	3	100
4	-15,...-8,8,...,15	3	101
5	-31,...-16,16,...,31	3	110
6	-63,...-32,32,...,63	4	1110
7	-127,...-64,64,...,127	5	11110
8	-255,...-128,128,...,255	6	111110
9	-511,...-256,256,...,511	7	1111110

Table 1-1 A Typical Huffman Table for dc difference

The ac coefficients are zero-runlength (ZRL) coded. Zig-zag scanning is used because it tends to records longer zero runs by visiting lower frequency coefficients first.

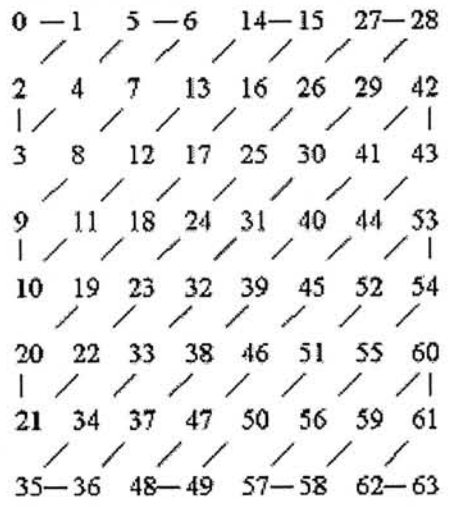


Figure 1-2 The Zig-Zag Sequence

The non-zero ac coefficients are coded similarly as the dc coefficients. However the tokens now become **(ZRL/SIZE, VLI)**. **ZRL** is the number of zeros since the last nonzero coefficient. **VLI** is the variable length integer of **SIZE** bits needed to specify the non-zero ac coefficient. The pair **(ZRL/SIZE)** is again entropy by Huffman code.

1.2 Video Coding Standard: MPEG

1.2.1 MPEG history

In 1992, the Moving Picture Expert Group (MPEG) has released an international standard, MPEG-1, for the compression of digital audio and video for transmission and storage. Uncompressed digital video requires an extremely high transmission bandwidth. For example, a PAL resolution television signal has a bit rate of approximately 100 Mb/sec. MPEG-1 can reduce this bit rate down to 1.5 Mb/sec, making it suitable for storage on compact discs. MPEG-1 is intended to be generic, it defines only the coding syntax and hence mainly the decoding procedure is standardized. MPEG-1 defines a DCT/DPCM hybrid-coding scheme with motion compensation method similar to the H.261 coding standard. It also provides functionality for random access required in digital media storage.

Studies on MPEG-2 began in 1990, with the initial target to issue a standard for coding of TV-pictures with CCIR Rec. 601 resolution (e.g. 352 pixels/line x 240 lines x 30 frames/sec) at data rates below 10 Mbps. In 1992 the scope of MPEG-2 was enlarged to suit coding of HDTV - thus making an initially planned MPEG-3 phase superfluous. The Draft International Standard (DIS) for MPEG-2 video was issued in early 1994.

The video coding scheme used in MPEG-2 is again generic and similar to the one of MPEG-1, however with further refinements and special consideration of interlaced sources. Furthermore many functionalities such as "scalability" were introduced.

Anticipating the rapid convergence of telecommunications industries, computer and TV/film industries, the MPEG group officially initiated a new MPEG-4 standardization phase in 1994. It targets to standardize algorithms and tools for coding and flexible representation of audio-visual data to meet the challenges of future multimedia applications requirements. In particular MPEG-4 addresses the need for:

- Universal accessibility and robustness in error prone environments
- High interactive functionality
- Coding of natural and synthetic data
- Compression efficiency

Bit rates targeted for the MPEG-4 video standard are between 5-64 kbps for mobile or PSTN video applications and up to 2 Mbps for TV/film applications.

1.2.2 MPEG video compression algorithm overview

All video compression algorithms achieve compression by exploiting the spatial redundancy and/or temporal redundancy exists in the video frames. The following paragraphs explain how does the MPEG algorithm exploit this redundancy [2],[5].

Spatial Redundancy

Video can be viewed as a sequence of still images, thus it can be compressed using the techniques defined in JPEG. A video frame is partitioned into non-overlapping 8 x 8 blocks. DCT is applied to every block, then the transform coefficients are quantized and coded. This technique exploits the spatial redundancy between pixels within a video frame and is referred as *intra-frame coding* because each video frame is independently compressed from other frames.

Temporal Redundancy

Intra-frame coding alone cannot compress the video down to the desired bit rate. There exists high correlation between adjacent video frames. MPEG exploits this temporal redundancy by using a block-based motion compensation approach.

A block of pixels (MacroBlock MB), called the target block, in the frame to be encoded is matched by a block of the same size in a reference frame (say the previous frame). The block that "best matches" the target block in the reference frame is used as a prediction. The prediction error is computed as the difference between the target block and the matching block. Associated with this matching block is a motion vector that describes the displacement of the matching block relative to the target block. The prediction error is then coded using the DCT approach as in intra-frame coding and transmitted to the decoder along with the motion vectors.

Smaller macroblock size allows us to find "better matching" block more easily and compress more efficiently. However, this also means that we have to transmit more motion vectors and overhead for the MB's. MPEG-1 chooses 16 x 16 macroblocks for motion compensations. This is a compromise between the compression efficiency and the storage overhead for the MB's.

MPEG has defined three frame types for temporal processing:

I-frames (intra-coded)

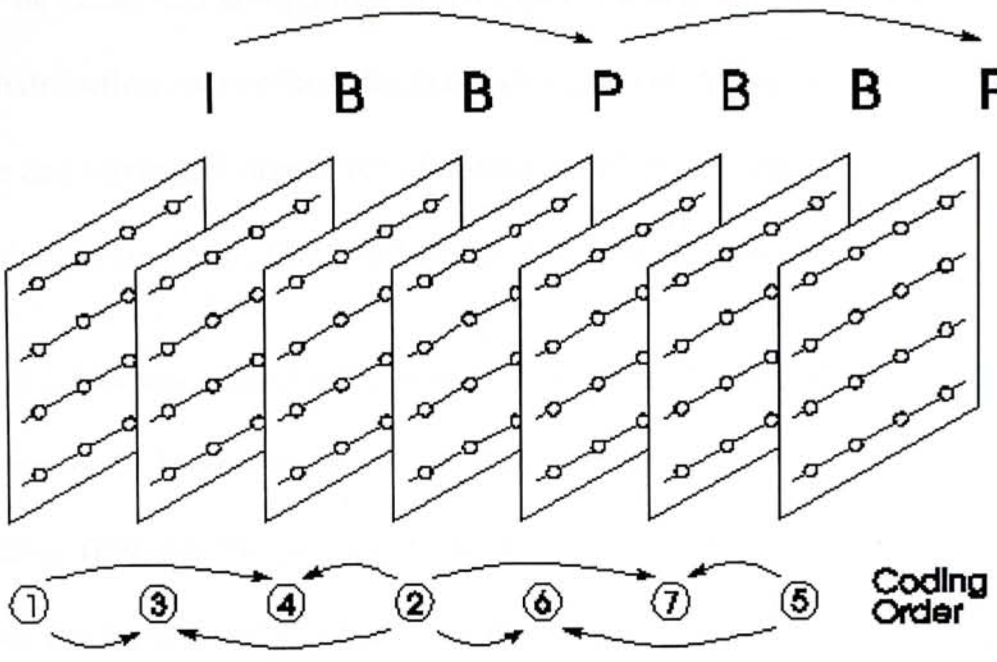
The I-frames are coded independently from other pictures. They provided random access and fast search points within the coded bitstream. The compression efficiency within the I-frames is moderate since inter-frame temporal redundancy is not used in encoding.

P-frames (forward predicted)

The P-frames are encoded based on the prediction from past I-frames or P-frames by using motion compensated prediction.

B-frames (bi-directionally predicted)

The B-frames are encoded based on the prediction from both past and future frames, thus named bi-directional. However, the B-frames themselves will not be used as reference for prediction. The use of bi-directional prediction gives better compression efficiency and video quality. This comes from the reduction in temporal redundancy and the ability to use future frames to address the unpredictable areas from the past reference.



The arrows indicate which pictures are used in prediction

Figure 1-3 The Relation between I-,P-,B-frames

1.2.3 More MPEG features

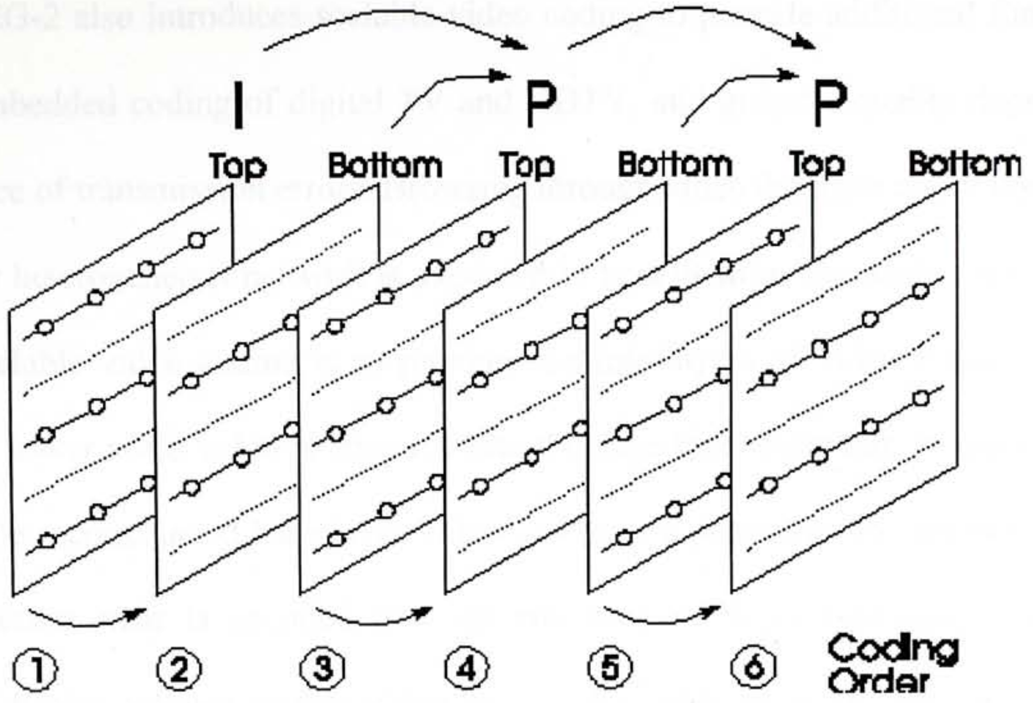
MPEG-1

MPEG-1 [15] was primarily targeted for multimedia CD-ROM applications. Important features provided by MPEG-1 include frame based random access of video, fast forward/fast reverse searches through compressed bit streams, reverse playback of video and editability of the compressed bit stream. MPEG-1 has been a very successful standard. It is the de-facto form of storing moving pictures and audio on the World Wide Web and is used in millions of Video CDs. Digital Audio Broadcasting (DAB) is a new consumer market that makes use of MPEG-1 audio coding.

MPEG-2

MPEG-2 can be seen as a superset of the MPEG-1 coding standard and was designed to be backward compatible to MPEG-1. Emerging applications, such as digital cable TV distribution, networked database services via ATM, digital VTR applications and satellite and terrestrial digital broadcasting distribution, were seen to benefit from the increased quality brought by the new MPEG-2 standardization.

MPEG-2 has introduced the concept of *frame pictures* and *field pictures* [5],[16] along with particular frame prediction and field prediction modes to accommodate coding of progressive (non-interlaced) and interlaced video. For interlaced sequences it is assumed that the input consists of a series of odd (top) and even (bottom) fields that are separated in time by a field period. Two fields of a frame may be coded separately (as *field pictures*). In this case each field is separated into adjacent non-overlapping Macroblocks and the DCT is applied on a field basis. Alternatively two fields may be coded together as a frame (*frame pictures*) similar to conventional coding of non-interlaced video sequences. Here, consecutive lines of top and bottom fields are simply merged to form a frame.



The concept of field-pictures and an example of possible field prediction. Each bottom field is coded using motion compensated Inter-field prediction based on the previously coded top field. The top fields are coded using motion compensated Inter-field prediction based on either the previously coded top field or based on the previously coded bottom field. This concept can be extended to incorporate B-pictures.

Figure 1-4 Field pictures and field-prediction

New *field prediction* modes are introduced to efficiently code the field pictures. In field prediction, predictions are made independently for each field by using data from one or more previously decoded field. Usually, inter-field prediction from the decoded field in the same picture is preferred if no motion occurs between fields. An indication which reference field is used for prediction is transmitted with the bit stream. Within a field picture, all predictions are field predictions. Frame prediction forms a prediction for a frame picture based on one or more previously decoded frames. In a frame picture either field or frame predictions may be used and the particular prediction mode preferred can be selected on a Macroblock-by-Macroblock basis.

MPEG-2 also introduces scalable video coding to provide additional functionality, such as embedded coding of digital TV and HDTV, and graceful quality degradation in the presence of transmission errors. Browsing through video database and transmission of video over heterogeneous network is expected to benefit from this scalability. The basic idea of scalable video coding is to provide multiple layers of video signal at different scale. The lower scale video is encoded into the base layer bitstream at reduced bitrate. The upscale reconstructed base layer video serves as a prediction for the original video. The prediction error is encoded into the enhancement layer bitstream. Decoder can choose to display a lower quality video by decoding only the base layer bistream. Thus scalable coding can be used to encode video with a suitable bit rate allocated to each layer in order to meet specific bandwidth requirements of transmission channels or storage media.

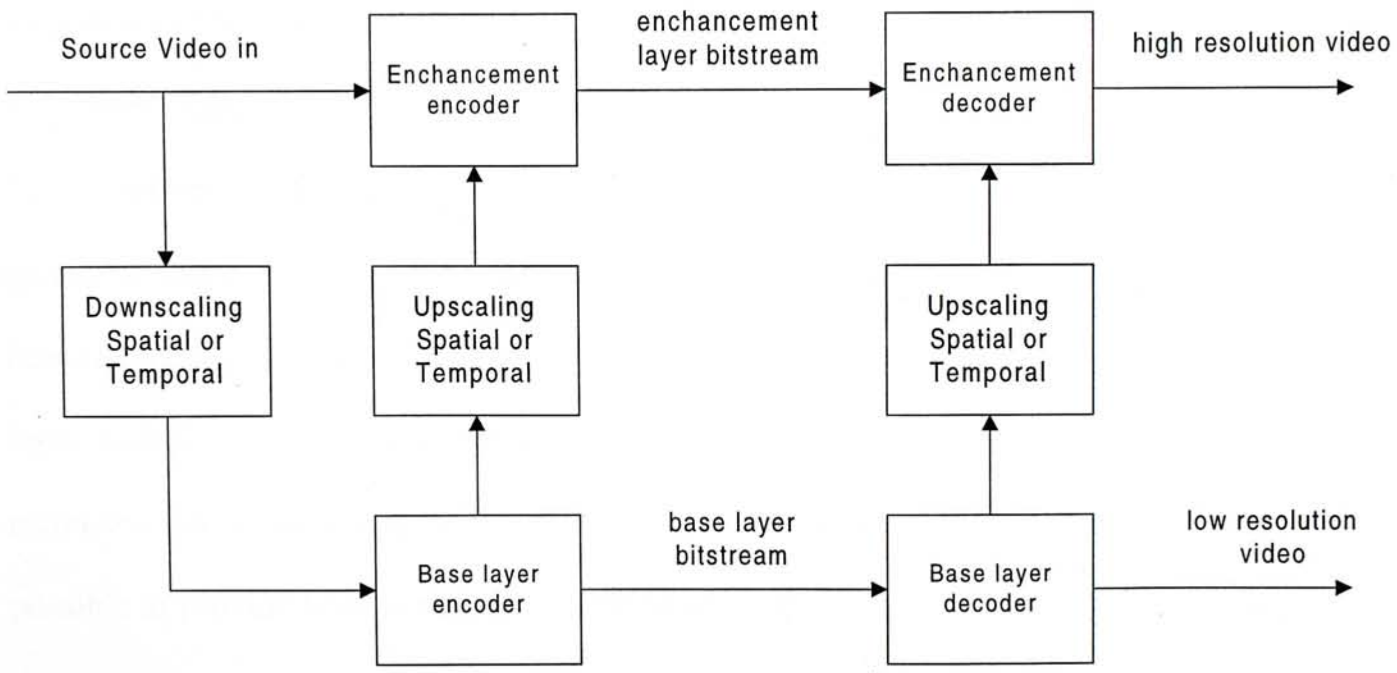


Figure 1-5 The scheme of Scalability

There are four basic scalable extensions defined in MPEG-2:

Spatial scalability

The algorithm is based on the classical pyramid progressive image coding. It is developed to support displays at different spatial resolution. This functionality is useful for application including embedding coding used in HDTV, allowing a migration from digital TV service to high spatial resolution HDTV service.

Temporal scalability

Layering is achieved by providing layers at different temporal resolution. The aim of this tool is similar to that of Spatial scalability.

SNR scalability

This tool has been developed to provide graceful degradation of the video quality in prioritized transmission media. If the base layer can be protected from transmission errors, a version of the video with gracefully reduced quality can be obtained by decoding the base layer signal only. One way to achieve SNR scalability is that at the base layer, the DCT coefficients are coarsely quantized and transmitted to achieve moderate image quality at reduced bit rate. The enhancement layer encodes and transmits the difference between the non-quantized DCT-coefficients and the quantized coefficients from the base layer with finer quantization stepsize. At the decoder the highest quality video signal is reconstructed by decoding both the lower and the higher layer bitstreams. It is also possible to provide SNR scalability by transmitting the first few DCT coefficients only in the base stream and the remaining coefficients in the enhancement layer.

Data partitioning

Data partitioning to designed to assist error concealment in the presence of transmission errors. It is not formally standardized with MPEG-2, but rather is referenced in the informative Annex of the MPEG-2 standard document. This is because this tool can be used entirely as a post-processing tool to any single layer coding technique. The algorithm is based on the separation of the DCT coefficients. Data partition can also be applied to separate the header information, such as the frame size and frame rate parameters, from the entropy coded stream.

It is possible to combine different scalability tools into a hybrid scheme. For example, services with different spatial resolution and frame rate can be supported by combining the Temporal Scalability and Spatial Scalability tools. Interoperability between HDTV and SDTV services can be provided along with a certain resilience to channel errors by combining the Spatial Scalability extensions with the SNR Scalability tool. With scalable video coding, decoder of various complexities can decode appropriate size replicas of the original video. Possible application areas include multi-party conferencing and video database browsing.

MPEG-4

MPEG is now working to produce MPEG-4, scheduled for completion in January 1999. The focus of MPEG-4 is the convergence of common applications for digital television, interactive graphics applications (synthetic content) and the World Wide Web (distribution of and access to content). MPEG-4 will provide the standardized technological elements enabling the integration of the production, distribution and content access of the three fields. Currently, MPEG-4 has identified the following key functions [6],[7]:

Content-based interactivity

New applications can be developed by the ability to interact with meaningful objects within an audio-visual scene. In existing standards such as MPEG-1 and MPEG-2, manipulation and editing are only possible in the original image domain. MPEG-4 will provide tools for content-based interaction and bitstream editing without transcoding. Otherwise, decoding before editing implies increased complexities and possible quality degradation. Possible fields of applications can be home video editing, video database queries, movies digital effects, online home shopping and video games.

Improved coding efficiency

MPEG-4 aims to provide better audio and visual quality at comparable bit rates of existing standards such as CCITT H.261. Better coding efficiency makes many applications more economically viable and competitive. Currently, core experiments on texture coding includes wavelet-based coding and matching pursuits.

Hybrid natural and synthetic coding

MPEG-4 will support for combining artificial object with natural video, thus allowing interactivity. It allows seamless integration of computer-generated graphics and natural scenes. Related techniques under experiment include shape coding, sprite coding, texture warping and wavelet coding.

Coding multiple concurrent data streams

Multimedia applications such as virtual reality, 3D movies, and multimedia presentations require efficiently coding multiple views and soundtracks of a scene.

Robustness in error-prone environments

Wireless communication requires error robustness for low bit-rate applications under severe conditions. MPEG-4 provides an error robustness capability to allow access multimedia applications over a variety of wireless and wire networks.

1.3 Summary

This chapter provides an overview of the JPEG and MPEG, the most popular standards for image and video coding respectively. We also present the basics of their compression algorithms and finally we review some of the features and functionalities developed by the different phases of MPEG, i.e. MPEG-1, MPEG-2 and MPEG-4. The knowledge of these materials will help the understanding on the issue of error resilience, which is going to be discussed in the next chapter.

Chapter 2 Error Resiliency

2.1 Introduction

With the rapid growth in mobile systems, error resilience is becoming increasingly important [7], [18]. Mobile communication channels in urban and suburban areas are subject to noise, fading, shadowing and interference [3]. All these make the wireless link an unreliable channel. Traditionally, channel coding is used to combat channel errors. However, when the channel condition is too bad, there are still uncorrectable errors. The compressed audio and video data are very vulnerable to errors: even a single bit error could result in significant quality degradation. Error resilience refers to the features of graceful degradation in the quality of the decoded data against deteriorating channel errors.

To demonstrate the problem of transmitting compressed data through noisy channel, Figure 2-1 shows a JPEG compressed image transmitted over a channel of BER 0.01%. A small number of bit errors can lead to great distortion in the decoded image. These few errors propagate over a large area of the image because of the use of Huffman code. A Huffman decoder is able to identify the correct separation of codewords from an error-free bitstream. However when there are errors, the decoder will either decoder longer or shorter codewords, leading to the loss of codeword synchronization and extended errors.

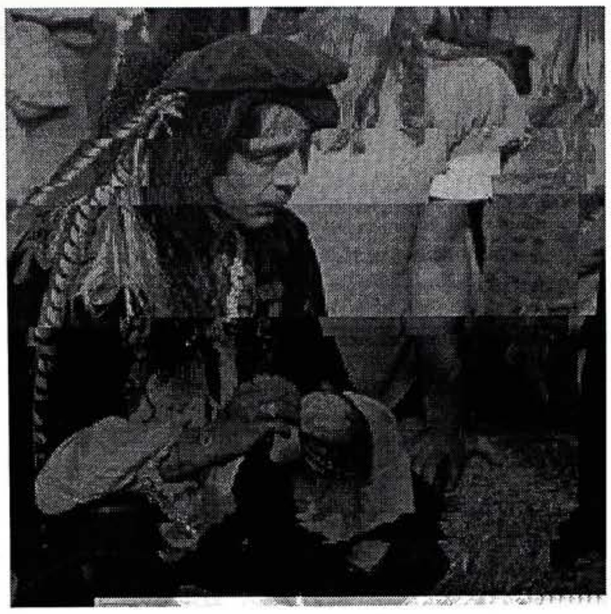


Figure 2-1 JPEG with 0.01% BER

The remainder of this chapter is organized as follows: in next section, we review several current existing methods in combating errors in video coding. Section 2.3 describes new ways to combat errors that are being considered by MPEG-4.

2.2 Traditional approaches

In this section will first review some prior methodology used to provide error resilience.

2.2.1 Channel coding

Forward error correcting codes (FEC) such as Reed-Solomon codes and BCH codes, can be used to correct certain amount of transmission errors. It involves the addition of extra parity bits to the compressed data, thus increases the total bandwidth required. Moreover, the FEC system is usually designed with the worst case scenario in mind. For channel with variable conditions, we need a very strong error correcting codes that adds much redundancy and hurts compression efficiency heavily. Besides, such a system will fail catastrophically when there are uncorrectable errors. This occurs when there is long burst of errors (as in a fading channel). One method to combat this is to use interleaving that turns long burst of errors into random errors, but this will introduce large delays.

2.2.2 ARQ

The Automatic Repeat Request (ARQ) protocol allows the decoder to request retransmission of the corrupted data. It is usually used with packet delivery systems. ARQ is effective in dealing with packet loss and long burst of errors. However, the retransmission of data can lead to significant delay and generate excess network traffic. This is undesirable for most real time applications such as video conferencing. Furthermore, ARQ is not efficient in dealing with random errors and short error bursts.

2.2.3 Multi-layer coding

The basic idea is to code the image/video in two or more layers. The base layer is used to code more important information, while subsequent layers are used to code the refinement information. The different layers can be given different error protection according to their priorities. This technique is also known as Unequal Error Protection (UEP). The bitstream from these layers may also be transmitted over different channels with different conditions.

2.2.4 Error Concealment

Error concealment actually is a post-processing technique. Here the decoder tries to detect errors within the received data using parity information or some statistical means.

For example, in the JPEG system, errors can be detected by discovering an illegitimate marker or an out-of-range DCT coefficient. Upon detecting an error, the decoder tries to conceal the errors by predicting the probable content of the corrupted data. A DCT codec, for example, can replace erroneous block by the average of its neighbouring blocks.

2.3 MPEG-4 work on error resilience

One of the targets identified by MPEG-4 is to provide error robustness and resilience to allow accessing image or video information over a wide range of storage and transmission media. The error resilience tools developed for MPEG-4 can be divided into three major areas. These areas or categories include resynchronization, data recovery, and error concealment [7]. These approaches address different areas in error resiliency. Resynchronization is almost indispensable in any coding system. MPEG-4 enhances the characteristics of these resynchronization tools. After synchronization is regained, the data recover tools attempt to recover the data that in general would be discarded. MPEG-4 also provides tools that help the error concealment methods, which are post-processing techniques to improve the decoded video quality.

2.3.1 Resynchronization

Resynchronization tools, as the name implies, attempt to enable resynchronization between the decoder and the bitstream after a residual error or errors have been detected. Generally, the data between the synchronization point prior to the error and the first point where synchronization is reestablished is discarded. If the resynchronization approach is effective at localizing the amount of data discarded by the decoder, then the ability of other types of tools that recover data and/or conceal the effects of errors is greatly enhanced.

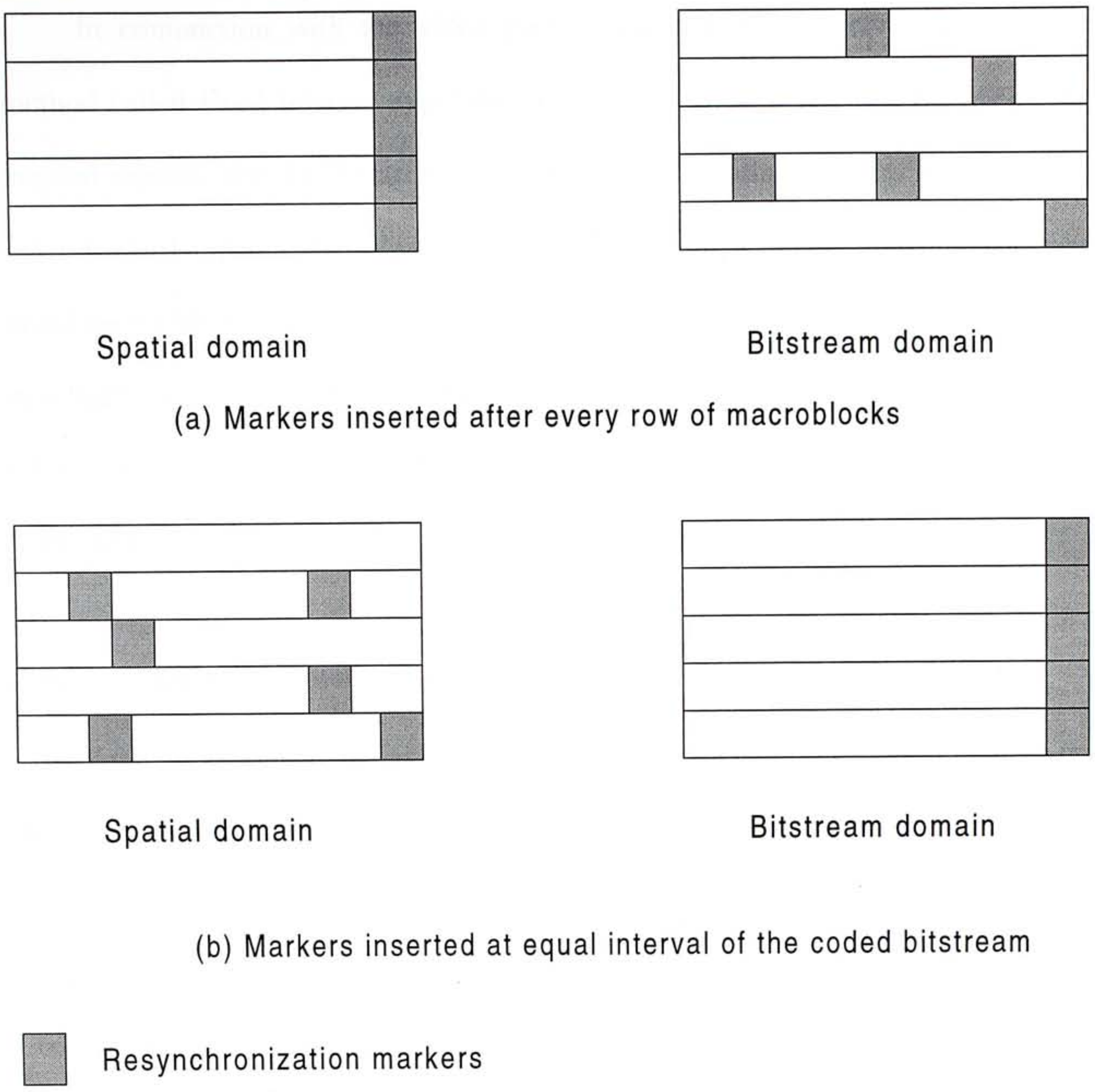
The resynchronization approach adopted by MPEG-4, referred to as a packet approach, is similar to the Group of Blocks (GOBs) structure utilized by the ITU-T standards of H.261 and H.263. In these standards a GOB is defined as one or more rows of macroblocks (MBs). At the start of a new GOB, information called a GOB header is placed within the bitstream. This header information contains a GOB start code, which is different from a picture start code, and allows the decoder to locate this GOB.

Furthermore, the GOB header contains information which allows the decoding process to be restarted (i.e., resynchronize the decoder to the bitstream and reset all predictively coded data). The GOB approach to resynchronization is based on spatial resynchronization. That is, once a particular macroblock location is reached in the encoding process, a resynchronization marker is inserted into the bitstream. A potential problem with this approach is that since the encoding process is variable rate, these resynchronization markers will most likely be unevenly spaced throughout the bitstream. Therefore, certain portions of the scene, such as high motion areas, will be more susceptible to errors, which will also be more difficult to conceal.

The video packet approach adopted by MPEG-4 is based on providing periodic resynchronization markers throughout the bitstream. In other words, the length of the video packets are not based on the number of macroblocks, but instead on the number of bits contained in that packet. If the number of bits contained in the current video packet exceeds a predetermined threshold, then a new video packet is created at the start of the next macroblock. This creates slices of macroblock with variable lengths.

A resynchronization marker is used to distinguish the start of a new video packet. This marker is distinguishable from all possible VLC codewords as well as the VOP (Video Object Plane) start code. Header information is also provided at the start of a video packet. Contained in this header is the information necessary to restart the decoding process and includes: the macroblock number of the first macroblock contained in this packet and the quantization parameter necessary to decode that first macroblock. The macroblock number provides the necessary spatial resynchronization while the quantization parameter allows the differential decoding process to be resynchronized.

It should be noted that when utilizing the error resilience tools within MPEG-4, some of the compression efficiency tools are modified. For example, all predictively encoded information must be confined within a video packet so as to prevent the propagation of errors.


Spatial domain
Bitstream domain
(a) Markers inserted after every row of macroblocks
Spatial domain
Bitstream domain
(b) Markers inserted at equal interval of the coded bitstream
Resynchronization markers

Variable MacroBlock Slice Tehnique:
Markers are inserted at equal interval of the bitstream, instead of after a fixed number of macroblocks. The slice of macroblocks between two adjacent markers become variable in length.

Figure 2-2 The use of variable Macroblock slice to provide error resilience

In conjunction with the video packet approach to resynchronization, a second method called fixed interval synchronization has also been adopted by MPEG-4. This method requires that VOP start codes and resynchronization markers (i.e., the start of a video packet) appear only at legal fixed interval locations in the bitstream. These help to avoid the problems associated with start codes emulation. That is, when errors are present in a bitstream it is possible for these errors to emulate a VOP start code. In this case, when fixed interval synchronization is utilized the decoder is only required to search for a VOP start code at the beginning of each fixed interval.

2.3.2 Data Recovery

After synchronization has been reestablished, data recovery tools attempt to recover data that in general would be lost. These tools are not simply error correcting codes, but instead techniques that encode the data in an error resilient manner. For instance, one particular tool that has been endorsed by the Video Group is Reversible Variable Length Codes (RVLC) [8]. In this approach, the variable length codewords are designed such that they can be read both in the forward as well as the reverse direction. Example of such codewords is 111, 101, 010. Codewords such as 100 are not used.

Figure 2-3 shows an example illustrating the use of a RVLC. Generally, in a situation such as this, where a burst of errors has corrupted a portion of the data, all data between the two synchronization points would be lost. However an RVLC enables some of that data to be recovered.

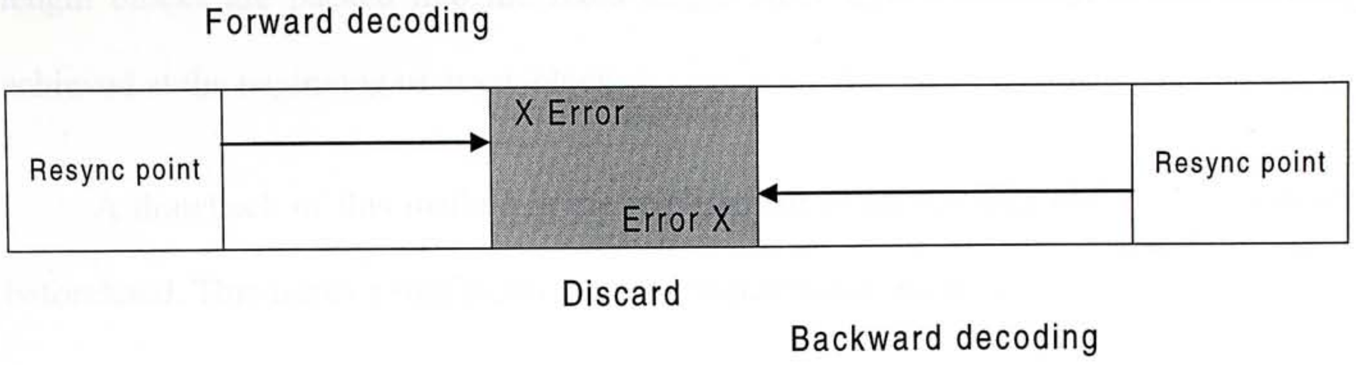


Figure 2-3 Example of RVLC

Other error resilient tools under consideration by MPEG-4 that enable data recovery include:

1) *Robust DC coefficient encoding*

For example, the median predictor is used instead of simple DPCM. In DPCM error propagates until the next reset of prediction. On the other hand, the median predictor has the ability to reject out-lier and is thus more robust to errors.

2) *Packaging VLCs into fixed length packets*

This is primarily to avoid the catastrophic loss of codeword synchronization. A technique of this category is the error-resilient entropy code (EREC) [17]. The basic operation of the EREC is to rearrange the *n* variable length blocks of data into fixed length slotted structure. In this way, the decoder can independently find the start of each block and start decoding it.

An example of the algorithm with 6 blocks is shown in the figure 2-4. At each stage k, whenever a block i has surplus bits over the fixed length slot, it will search in slot *(i+k) mod n* for empty space to put the remaining bits. After at most *n* stages, all the variable length blocks are packed into the fixed length slots. Synchronization is automatically achieved at the beginning of every block.

A drawback of this method is the requirement of all the data blocks to be known beforehand. This implies significant memory requirement and delay.

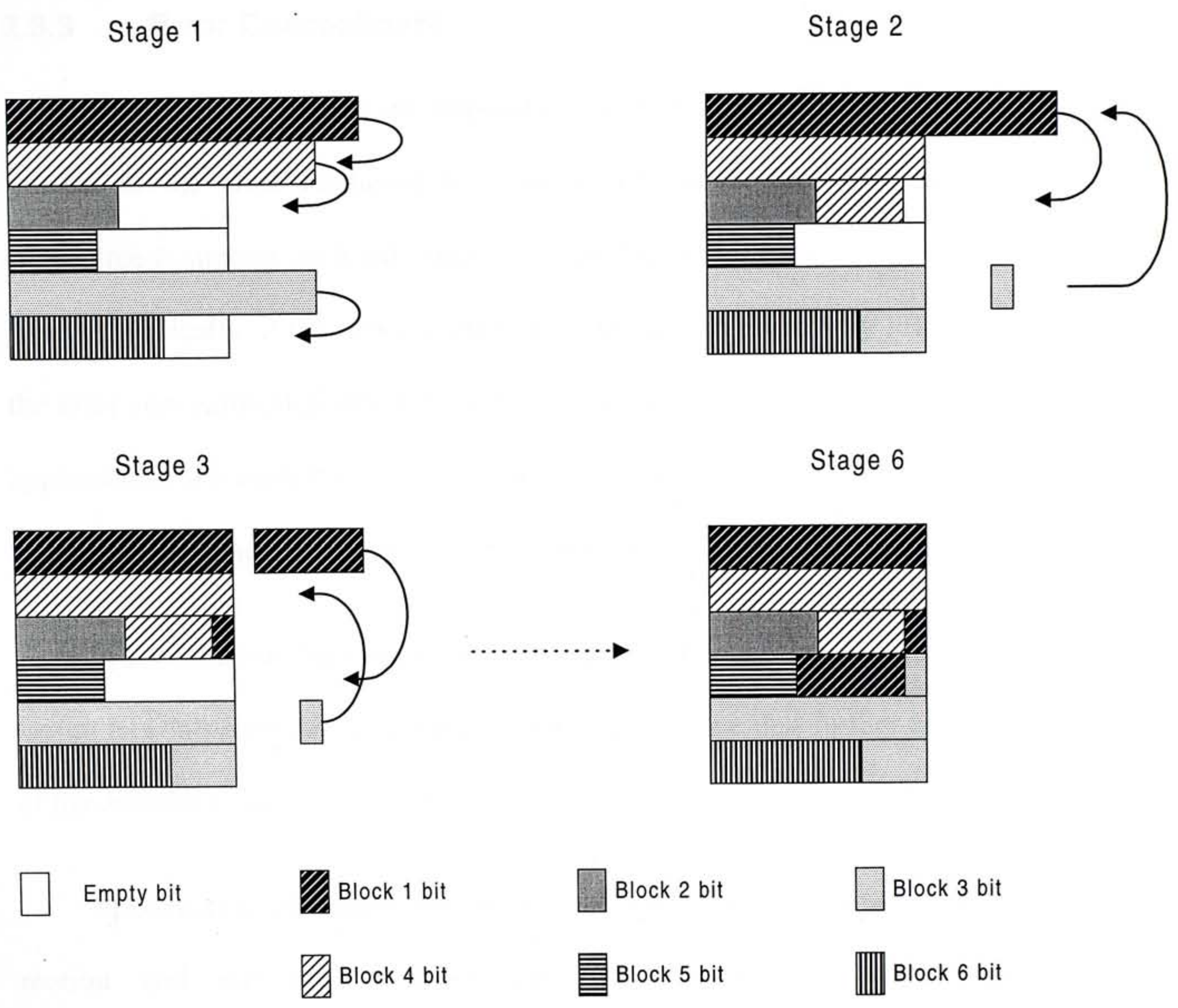


Figure 2-4 Example of the EREC algorithm

3) Multiple transmission within the bitstream of key information

In each video packet, a 1-bit field called *Header Extension Code* (HEC) is introduced. If this bit is set, the important header information (e.g. spatial dimension, motion vectors) which describes this video frame is repeated.

2.3.3 Error Concealment

Error concealment is an important component of an error robust video codec. Similar to the error resilience tools discussed above, the effectiveness of an error concealment strategy is highly dependent on the performance of the resynchronization scheme. Basically, if the resynchronization method can effectively localize the error then the error concealment problem becomes much more tractable. For low bitrate, low delay applications, the current resynchronization scheme provides very acceptable results with a simple concealment strategy, such as copying blocks from the previous frame.

In recognizing the need to provide enhanced concealment capabilities, the Video Group has developed an additional error resilient mode that further improves the ability of the decoder to localize an error.

Specifically, this approach utilizes the data partitioning tools by separating the motion and the texture information. This approach requires that a second resynchronization marker (Motion Boundary Marker MBM) be inserted between motion and texture information. If the texture information is lost, this approach utilizes the motion information to conceal these errors. That is, due to the errors, the texture information is discarded, while the motion information is used to motion compensate from the previous decoded VOP.

2.4 Summary

This chapter discussed the importance of error resilience in image and video communications. Some prior approaches of providing error resilience are described. Thereafter we introduced the current work by MPEG-4 on error resilience. These tools are mainly on the area of resynchronization, data recovery and error concealment.

Chapter 3 Fixed length codes

3.1 Introduction

In this chapter, we present a variable-to-fixed length coding scheme to entropy code the DCT coefficients and compare the results to the baseline JPEG Huffman system.

Traditional entropy coding of the DCT coefficients, such as Huffman, employs fixed-to-variable length coding scheme, in which a fixed length block of symbols is matched to a variable length bit string. Here, we experiment a variable-to-fixed length coding scheme in which a variable number of DCT coefficients are matched to a fixed length binary string. This is a feasible solution to the error resiliency problem because transmission errors will not propagate beyond the boundaries of the fixed length codes. With our modified Lempel-Ziv algorithm, we have implemented a variable-to-fixed length coder which cost about 20% additional bit-rate than the baseline JPEG Huffman system without special error resiliency consideration, but cost less than the reverse Huffman code proposed to the MPEG-4 study group.

A variable-to-fixed length encoding is a mapping from a dictionary of variable-length strings of source symbols to a set of codewords with a given length. A variable length code, like Huffman code, achieves compression by assigning shorter codewords to more frequent source symbols and longer codewords to less frequent source symbols. A variable-to-fixed length code, on the other hand, chops the input source sequence into segments of variable length and represents each segment with an index from its dictionary of possible segments. To achieve compression, it tries to maximize the average number of source symbols represented by each dictionary index. The Tunstall code and Lempel-Ziv code are examples of variable-to-fixed length codes.

Fixed length codes are inherently more resilient to errors. A variable length decoder may output codewords of wrong length in case of errors, thus lead to the loss of codeword synchronization. Codeword synchronization is not necessary for fixed length codes since we know the exact positions of the codeword boundaries. The effect of errors is always confined within the codeword corrupted and does not propagate.

The remainder of this chapter is organized as follows: section 3.2 and 3.3 briefly reviews the Tunstall and Lempel-Ziv coding algorithms. Section 3.4 gives the details of our variable-to-fixed length coder and presents the simulation results.

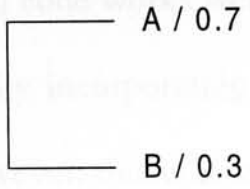
3.2 Tunstall code

This section reviews the Tunstall coding algorithm. The Tunstall code [9] construction procedure starts with a coding tree with all single-letter string as leaves. The leaf-nodes will correspond to the strings in the Tunstall dictionary. Associates with each string is the probability of occurrence of that string.

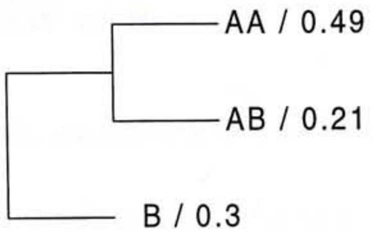
The algorithm chooses the leaf node with the largest probability and split it down by one more level. (Ties in choosing the maximum probability are broken arbitrarily.) This is equivalent to appending a single letter to the string currently with the largest probability. The probabilities of the newly added strings are updated accordingly. This process iterates until the total number of leaf-nodes exceeds the desired dictionary size. An index is then assigned to every leaf-node, according to their lexicographical order.

As an example, consider a binary memoryless source with alphabet {A,B}, and Pr{A}=0.7, Pr{B}=0.3. We construct a Tunstall code with 8 strings in the dictionary, thus we use 3 bits ($2^3=8$) per index. Figure 3-1 shows the first few steps and the final step of the codebook construction. The final codebook is:

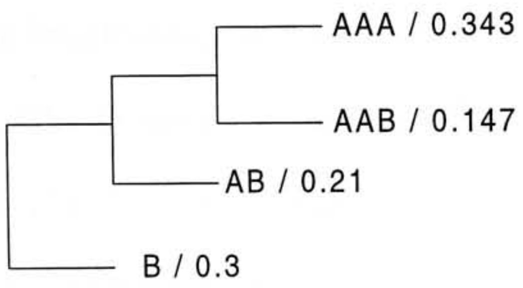
Index	**Strings represented**
000	AAAAA
001	AAAAB
010	AAAB
011	AAB
100	AB
101	BAA
110	BAB
111	BB



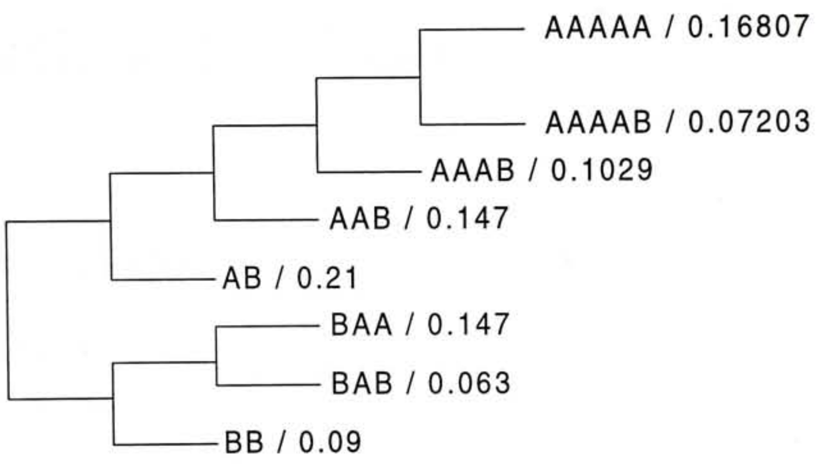
Step 1 Initial tree



Step 2



Step 3



Step 7

Figure 3-1 Example of Tunstall code construction

The Tunstall code works well with memoryless sources. Of course, we can improve the performance by incorporating higher order probability estimates when updating the probabilities. However, this will increase the complexities drastically.

3.3 Lempel-Ziv code

In this section, we review two famous compression algorithms, by A. Lempel and J. Ziv, commonly referred to as LZ77 and LZ78.

The Lempel-Ziv code belongs to a class of dictionary-based compression methods. It encodes variable length strings into a single token and these tokens form an index to a phrase dictionary. The Lempel-Ziv code was first invented in 1977 [10] by Abraham Lempel and Jacob Ziv. They published another paper [11] in 1978 describing another dictionary-based compression method. These two techniques developed are called LZ77 and LZ78. LZ77 is a "sliding window" technique in which the dictionary consists of a set of fixed-length phrases found in the previously processed window. LZ78 builds up its dictionary one at a time, adding a new symbol to an existing phrase whenever a match occurs.

In the following sub-sections, we will review the LZ77 and LZ78 coding algorithms respectively.

3.3.1 LZ-77

The LZ77 maintains two sliding windows: the just encoded window of length **N** and the to-be-encoded window of length **K**. The compressor loops the following three steps:

Parse: the compressor tries to find the longest prefix in the to-be-encoded window as matched by a substring inside the just encoded window.

Encode: the matching information is encoded by emitting the token

(go back **n**, copy **k**, append **'A'**)

where 'A' is the first letter following the matched prefix string inside the to-be-encoded window. This token tells the decoder to go back **n** letters in the just encoded window, start from there, copy **k** letters and append a letter **'A'**.

Update: both windows slide forward by k+1, i.e. the length of the matched string plus the appended letter.

The LZ77 is a fixed length code because all the parameters within the token (n, k, 'A') are represented by bit strings of given lengths.

Example:

Assume we are already in the middle of the compression process and we will encode:

...H.261, H.262, H.263...

with N=7, K=4.

step	**just encoded window**	**to-be-encoded window**	**output token**
1	"H.261, "	"H.26"	back 7, copy 4, append '2'
2	", H.262"	", H."	back 7, copy 4, append '2'

The first two steps of encoding are shown.

Note that the just encoded window in LZ77 implies the dictionary used. All the k-sub-strings within the just encoded window are members of the dictionary.

3.3.2 LZ-78

The LZ78 algorithm maintains a dictionary of phrases it has seen before, all the phrases are distinct and each is given a unique index. The initial dictionary contains only one entry: the NULL phrase with an index of 0.

Similar to its counterpart, LZ78 loops these three steps:

Parse: find a phrase P in the current dictionary that matches the longest prefix of the to-be-encoded string.

Encode: the compressor output the token

(copy i-th phrase, append 'A')

where the i-th phrase is P and 'A' is the letter P in the to-be-encoded string.

Update: the new phrase (P, 'A') is added to the dictionary.

The pair of dictionary indexes **i** and the appending letter '**A**' is again coded with bit string of given lengths respectively.

Example:

We will encode the same input sequence:

...H.261, H.262, H.263...

Assume the LZ78 dictionary is already occupied with some single letters:

index	string
0	NULL
1	“H”
2	“.”
3	“2”
4	“6”
5	“1”
6	“,”
7	“ ”

The encoding steps are shown below:

Steps	Matched string	Output token	New string added
1	“H”	1, ‘.’	index=8, “H.”
2	“2”	3, ‘6’	index=9, “26”
3	“1”	5, ‘,’	index=10, “1,”
4	“ ”	7, ‘H’	index=11, “ H”
5	“.”	2, ‘2’	index=12, “.2”
6	“6”	4, ‘2’	index=13, “62”
7	“,”	6, ‘ ’	index=14, “, ”
8	“H.”	8, ‘2’	index=15, “H.2”
9	“6”	4, ‘3’	index=16, “63”

In both LZ77 and LZ78 algorithms, the dictionary need not be sent to the decoder. The decoder can infer the dictionary from the decoded tokens and update its dictionary in pace with the encoder. This dictionary update process depends on the previously decoded result. Hence, a single error can result in a wrong entry in the dictionary and lead to many future errors.

3.4 Simulation

We have modified the original LZ78 algorithm to produce a stationary codebook. This variable-to-fixed length codebook will be used to code the stream of DCT coefficients and we have simulated our algorithm on a series of standard test images. The compression efficiency is about 20% worse than the baseline JPEG Huffman, and this is the cost to provide the error resiliency. In comparison, our scheme compares favorably to the reverse Huffman code which costs about 50% extra redundancy [20].

3.4.1 Experiment Setup

To use the LZ78 algorithm, we need some modifications:

(1) To avoid the problem of loss of dictionary synchronization between the encoder and decoder, the LZ78 dictionary is trained by an image beforehand. The trained dictionary will be used to encode the other images. This sacrifices some compression efficiency, but is justified by the error resiliency it brings.

(2) The starting LZ78 dictionary is initially loaded with all single-coefficient string. This ensures that we can always find a match: at least a string of length one.

(3) Now, we do not have to update the dictionary and hence we shall no longer append a letter after the matching index. The encoder just has to search for the longest matched prefix from the trained dictionary and output the index found.

In the training stage of the dictionary, we proceed as usual in the original LZ78 algorithm with the DCT coefficients from the training images as input sources. We grow the dictionary by one string at a time. The maximum size of the dictionary is chosen to be 2^{16} (64K), and a 16-bit string represents every dictionary index. When the dictionary is filled up to 64K entries, we will stop the training process. The choice of a 16-bit index makes the parsing of the compressed stream much easier since we do not have to deal with the byte alignment problem. This allows a simpler and faster decoder. Moreover, the 2^{16} entries provide us a sufficient number of string candidates to match the incoming DCT coefficient stream.

In this experiment, the LZ78 algorithm is preferred to Tunstall code because the alphabet size of the transform coefficients is too large for Tunstall. Moreover, there exists correlation between the coefficients, they do not imitate a memoryless source. Keeping the higher order probability statistics of these coefficients means too much computation. For the LZ77 sliding window technique, the previously processed text becomes the source of the dictionary. A single bit error will lead to loss of synchronization between the windows content being kept by the encoder and decoder. This creates extended errors in the decoded coefficients. Therefore the LZ78 algorithm is chosen.

3.4.2 Results

Every image is DCT transformed and quantized with a quality factor of 50. The DPCM on the dc coefficients is switched off. This is to avoid the problem of cascaded errors due to the use of prediction from previously decoded coefficients. The ac coefficients are run-length coded as usual. The quantized coefficients and the run-length symbols are then coded using the modified LZ78 algorithm. Four different training images are used to train up the dictionary. These training images include landscapes and portraits. Table 3-1 records the result bit rates.

		Training Images			
	JPEG	baboon	barb	boat	bridge
Couple	29188	34539	35717	36451	34833
Crowd	28810	35655	36279	37001	34751
Girl	22511	27407	28209	28095	27577
Goldhill	27449	32917	33813	34845	32465
Lake	29396	36403	37389	37767	35673
Lena	20921	26197	26063	26691	25775
Man	28102	33583	34577	35239	33281
Peppers	21310	26395	26703	27279	26211
Plane	22602	30469	29569	29769	28767
Tiffany	25642	30087	31371	32037	30483
Woman	14323	20177	18899	19117	18537
Zelda	17262	21341	21405	21959	21063
Total Bytes	287516	355170	359994	366250	349416
Vs. JPEG		+23.5%	+25.2%	+27.4%	+21.5%

Table 3-1 16-bit LZ-78 on DCT coefficients of the test images

The bit rate by the baseline JPEG Huffman system *(see section 1.1)* is shown also. Here the same quantization parameters are used as in our modified LZ-78 algorithm. Hence, we are comparing the efficiency at the stage of losslessly code the quantized DCT coefficients. The baseline JPEG and our LZ-78 decoder should give identical decoded pictures in an error-free condition.

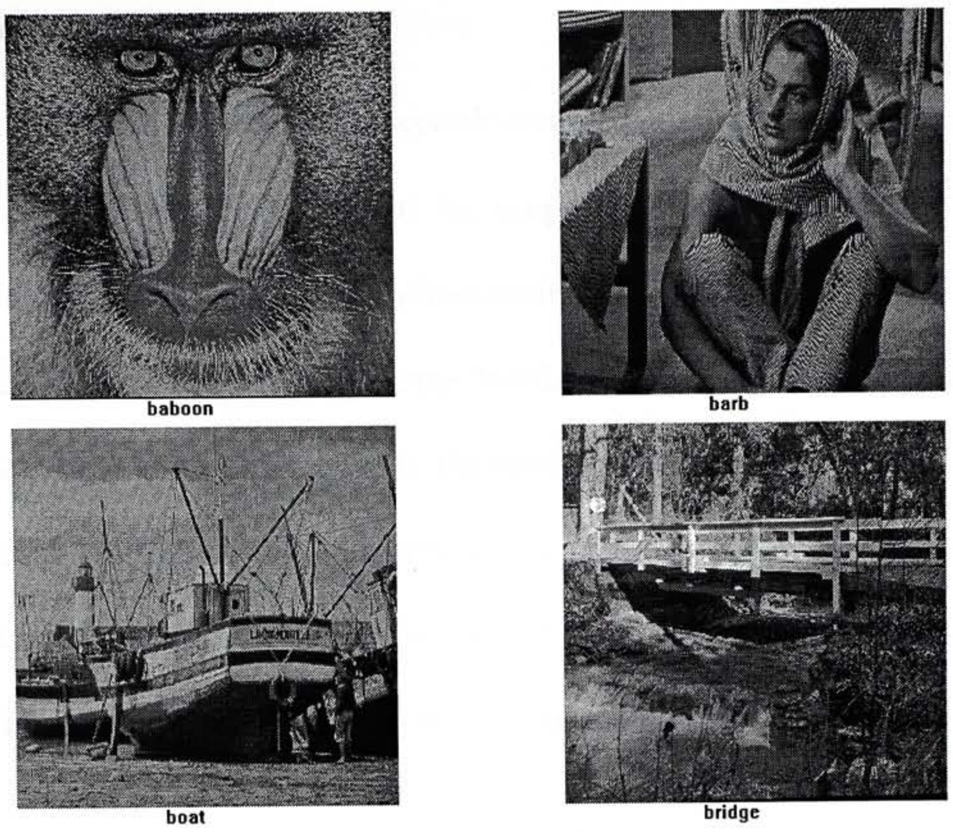


Figure 3-2 The training image set

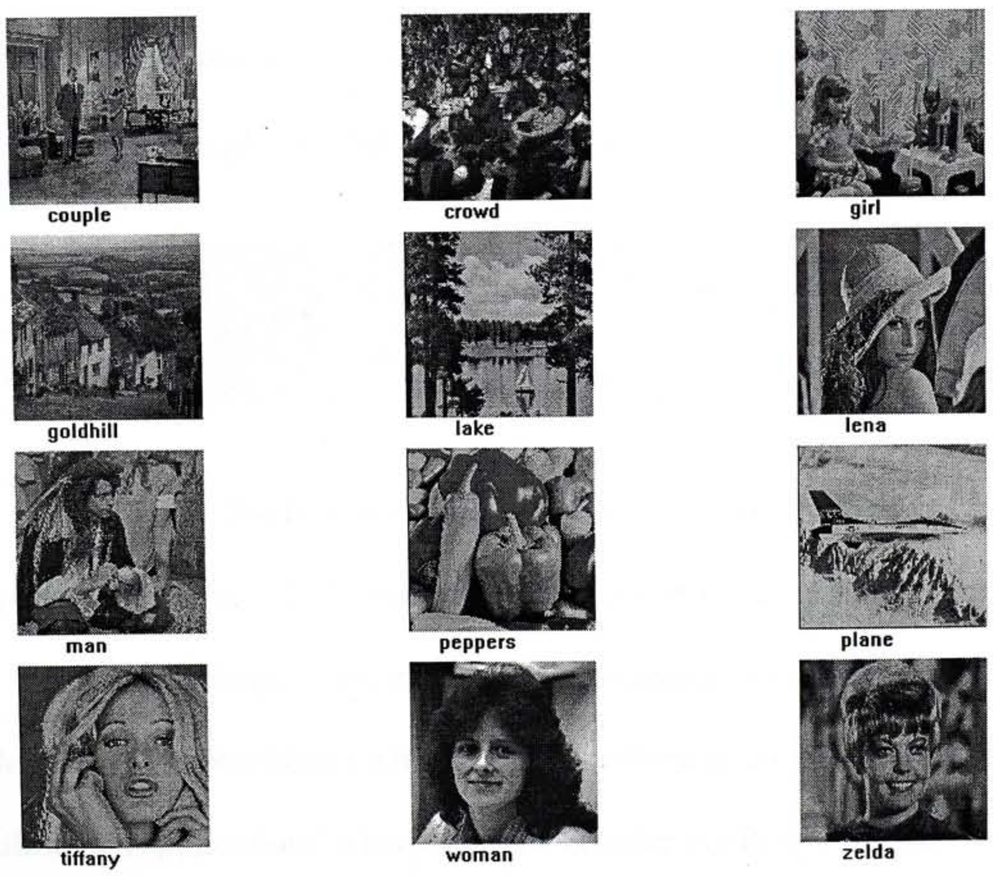


Figure 3-3 The images coded by the LZ-78

3.4.3 Concluding Remarks

The modified LZ-78 algorithm records over 20% bitrate than the baseline JPEG Huffman encoder. However, it gives us the property of codeword synchronization and can be used in conjunction with the bi-directional decoding as in the RVLC of MPEG-4. Comparing LZ78 with the simple reverse Huffman codes to achieve reversibility, we have improved the coding efficiency. In the construction of a reversible Huffman code, we take a usual Huffman code and mirror extend those codewords that are not self-symmetric (i.e. codewords that read the same in both forward and reverse directions). This usually introduces a redundancy from 50% to 100% [20].

An advantage of the variable-to-fixed length code is that we can further recover those data between two error points. In contrast to simple RVLC, the decoder still outputs a series of wrong words when error occurs. With fixed length codes, the error is confined within the codeword corrupted so that we can continue the decoding process.

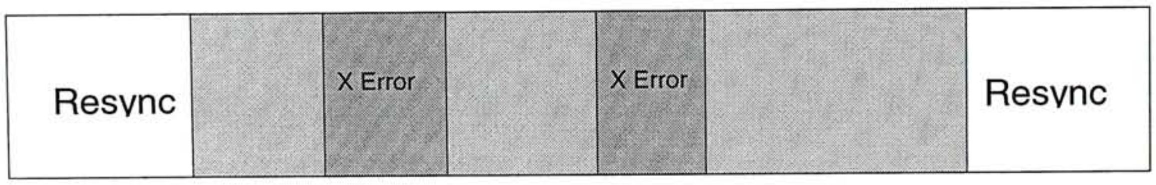


Figure 3-4 Recover data between two error points

Compare with the standard Huffman, our encoder requires training overhead and the trained codebook is large. This imposes a large memory requirement. On the other hand, the decoder is simpler because it just has to perform a straightforward table lookup. In most multimedia applications, a less complex decoder is often preferred.

Although there are papers proving that fixed-length codes are better than variable length codes asymptotically [12], their conditions require a word length and codebook size that tends to infinity, which is not viable in practical applications. Nevertheless, the use of fixed-length codes provides good error resilience by avoiding the catastrophic damage caused by errors, as in the case of Huffman code.

In the next chapter, we will use a class of self-synchronizable code to achieve codeword synchronization. This class of self-synchronizable codes is variable in length and we will show that it further improves the coding efficiency.

Chapter 4 Self-Synchronizable codes

4.1 Introduction

In this chapter, we present another solution to the error resiliency problem with the application of Scholtz's class of self-synchronizable code (or synchronizable code) to the entropy coding of the DCT coefficients. These codes have the property that synchronization of variable-length codewords can be quickly recovered after errors. This property effectively limits error propagation and provides error resiliency.

This class of synchronizable code offers self-synchronizability of the bitstream and good compression efficiency on the DCT coefficients. It provides good error resiliency because the synchronization procedure can start anywhere within the bitstream and does not depend on what comes before. We have implemented an encoder using Scholtz's synchronizable code and shown that its compression efficiency is about 10% worse than the baseline JPEG Huffman. Combining with the technique of reversible decoding, we have demonstrated an improvement in the decoded image quality under various error conditions.

The organization of this chapter is as follows: section 4.2 gives a tutorial on Scholtz synchronizable code, including its construction and synchronization procedure. Section

4.3 gives the simulation results of our Scholtz's encoder on the DCT coefficients and also the bi-directional decoder we have implemented. Section 4.4 is the concluding remarks.

4.2 Scholtz synchronizable code

4.2.1 Definition

A synchronizable code (**SCs**) has the property that the punctuation, i.e. the comma separating the codewords, can always be determined by observing at most ***s*** neighbouring code symbols of the position in question. The number ***s*** is called the *synchronization delay* of the code.

When a transmission error occurs, the decoder can temporarily misjudge the boundaries between codewords, but the property of self-synchronizability ensures that codeword synchronization can be recovered within a short time.

4.2.2 Construction procedure

Scholtz showed in his paper in 1966 that one can construct a synchronizable code from another [13], [14]. Consider a synchronizable code C consists of codewords c_1, c_2, ... ,c_n. A new synchronizable code can be constructed with the following procedure:

Suffix Construction Procedure

1. Remove a codeword $\mathbf{c}_i$ from C.
2. Create new codewords by appending $\mathbf{c}_i$ as a suffix, with an arbitrary number of times, to the remaining codewords. Thus the new codewords added are:

$c_1 \mathbf{c}_i$	$c_1 \mathbf{c}_i \mathbf{c}_i$	$c_1 \mathbf{c}_i \mathbf{c}_i \mathbf{c}_i$
$c_2 \mathbf{c}_i$	$c_2 \mathbf{c}_i \mathbf{c}_i$	$c_2 \mathbf{c}_i \mathbf{c}_i \mathbf{c}_i$
...	...	...
$c_{i-1} \mathbf{c}_i$	$c_{i-1} \mathbf{c}_i \mathbf{c}_i$	$c_{i-1} \mathbf{c}_i \mathbf{c}_i \mathbf{c}_i$
$c_{i+1} \mathbf{c}_i$	$c_{i+1} \mathbf{c}_i \mathbf{c}_i$	$c_{i+1} \mathbf{c}_i \mathbf{c}_i \mathbf{c}_i$
...	...	...
$c_n \mathbf{c}_i$	$c_n \mathbf{c}_i \mathbf{c}_i$	$c_n \mathbf{c}_i \mathbf{c}_i \mathbf{c}_i$

The only restriction is that we do not exceed the maximum desired word length.

The code derived from this procedure is also synchronizable by the following argument. After observing s letters, we must be able to determine the punctuation for the original code C at a particular point of the code stream. If the punctuation is a comma, we must determine whether or not the following word is c_i from C. This requires an observation of an additional k_i symbols, where k_i is the length of c_i. Hence the new synchronization delay s' is given by $s' = s + k_i$.

Let us look at an example to see how the procedure works. We start with the simplest synchronizable dictionary SC_0:

$C^{(0)}$: 1, 0

Using **0** as the suffix word and setting the maximum word length to be 5, we derive

$C^{(1)}$: 1

10

100

1000

10000

Repeat the suffix construction procedure, with **1** as the suffix word this time. We have

$C^{(2)}$: 10

100 101

1000 1001 1011

10000 10001 10011 10111

Further modification using **10** as the suffix word results

$C^{(3)}$: 100 101

1000 1001 1011

10000 10001 10011 10111 10010 10110

The synchronization delay of the code $C^{(3)}$

= length of **0** + length of **1** + length of **10**

= 4

Let us call the codewords **0**, **1**, **10** the *atoms* of $C^{(3)}$ since every word in $C^{(3)}$ is a (repeated) concatenation of these atoms. The final synchronizable code can be fully specified by the statement of the initial dictionary ($C^{(0)}$), the atoms chosen and their order, and the maximum allowed word length.

4.2.3 Synchronizer

Following the above example, we shall demonstrate how synchronization can be achieved for the code $C^{(3)}$. Similar to the construction procedure, the synchronization process for $C^{(3)}$ relies on the synchronization capability of the base codes from which it is derived.

As a test sequence, let us use

…00,100,101,10111,10010,10…

The first synchronizer for $C^{(0)}$ is trivial, we just have to insert a comma between every letter.

The next operation is to synchronize the sequence for the dictionary $C^{(1)}$. This involves erasing commas preceding the codeword suffix **0** which is a member from $C^{(0)}$. No word in $C^{(1)}$ (and hence no words in $C^{(2)}$ and $C^{(3)}$) begin with a **0**.

Next, we have to erase commas preceding **1**'s, but only when **1** appears as a complete word. The synchronization of the original sequence is finally completed by erasing the comma preceding the **10** word in $C^{(2)}$.

Notice that the erasure of a comma only depends on what follows the comma, this whole procedure does not depend on where one starts the synchronization process in a stream of code symbols. We have put a '?' at the punctuation position of the stream because the synchronizer must observe more symbols to make a decision. To be exact, we have to inspect 4 letters to decide the punctuation at the position of '?' for the code $C^{(3)}$.

The codes derived from the Suffix Construction Procedure are suffix codes: no codeword being a suffix of another codeword. Of course one can construct a self-synchronizable prefix code by the Prefix Construction Procedure, i.e. the atoms are being added to the head of the other codewords in each derivation. The synchronizer for this prefix code shall be changed to remove the comma *following* the atoms in each step of synchronization.

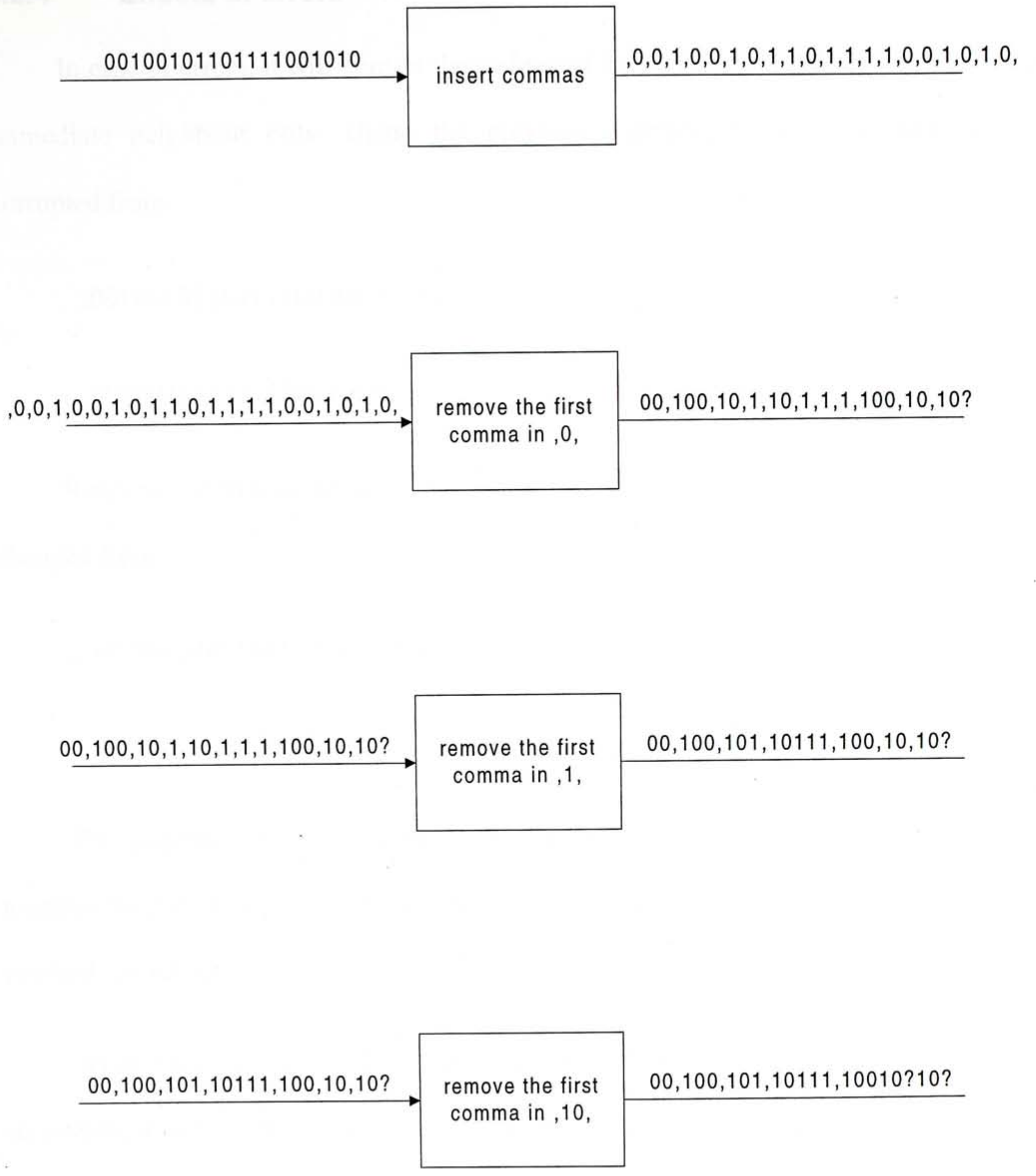


Figure 4-1 The synchronizer for the self-synchronizable code $C^{(3)}$

4.2.4 Effects of errors

In case of errors, it will corrupt the codeword that contains the erroneous bit and its immediate neighbour only. Using the previous example, suppose the sequence is corrupted from

…001001011**0**1111001010… to

…001001011**1**1111001010…

Running the synchronization procedure as described before, the parsing result is changed from:

…00,100,101,1**0**111,10010?10?… to

…00,100,1011**1**111,10010?10?…

The corrupted bit is highlighted. We see that two of the codewords are merged together but the other codewords remain intact. The synchronization points (commas) are inserted correctly.

When there are channel erasures, Scholtz's synchronizable code is capable of recovering synchronization very quickly too. This is because the synchronization process can start from anywhere within the bitstream.

4.3 Simulation

We have adapted the Scholtz algorithm to code the DCT coefficients from a collection of test files. The compressed stream is subjected to various levels of transmission errors and we have used both forward decoding alone and bi-directional decoding to reconstruct the pictures. The decoded pictures are visually examined and the PSNR are calculated. It is found that our algorithm performs only slightly worse than Redmill and Kingsbury [17] in terms of bit-rates and visual quality, but are competitive on other features such as buffering and decoding complexity.

4.3.1 Experiment Setup

We construct a codebook consists of a set of variable length codewords by the procedure reviewed in the previous section. The construction parameters are chosen as follows:

Initial dictionary:	{1,0}
Atoms chosen:	{0,1,1000}
Maximum code length:	12

To assign these codewords to the DCT coefficients, we obey the basic principle of data compression, i.e. use shorter codewords for more frequent symbols and longer codewords for less frequent symbols. For most images, there are more coefficients of smaller magnitudes after quantization. Therefore the shorter codewords are assigned to them first. For the zero runlengths, shorter runs of zeros are more frequent except for the EOB symbol. Empirical probabilities of the coefficients are collected and we assign shorter codewords to the more probable codewords in order. We have separate codebook for the dc and ac coefficients because they have different range and exhibit different distribution. This is also true in the baseline JPEG Huffman model.

The Scholtz algorithm comes with some compression inefficiency in order to provide synchronizability. One is the lack of total flexibility to change the length of the codewords. Another is that it is not a full-tree code, i.e. some nodes of the coding tree have only one child. Therefore we use the following adaptation to increase the compression efficiency. We code every non-zero coefficient (both dc and ac) into two numbers. The first number is the quotient of the coefficient when divided by 8; the second number is the absolute-valued remainder when the quotient is non-zero. The sign of the remainder is dropped because the sign of the coefficients can be inferred from the quotient.

The final codebook to be used is:

codebook														
DC			AC								
symbol	codeword	length	symbol	codeword	length						
0	10	2	1	10	2	ZR20	100000011	9	16	10000111111	11
1	100	3	EOB	100	3	ZR21	100000111	9	ZR38	10001111000	11
2	101	3	2	101	3	8	100001000	9	-16	10001111111	11
3	1001	4	ZR1	1001	4	ZR22	100001111	9	ZR39	10010001000	11
4	1011	4	3	1011	4	-8	100011000	9	ZR40	10011111000	11
5	10000	5	ZR2	10000	5	ZR23	100011111	9	ZR41	10011111111	11
6	10001	5	4	10001	5	9	100111000	9	ZR42	10110001000	11
7	10011	5	ZR3	10011	5	ZR24	100111111	9	ZR43	10111111000	11
-1	10111	5	5	10111	5	-9	101111000	9	ZR44	10111111111	11
-2	100000	6	ZR4	100000	6	ZR25	101111111	9	ZR45	100000000000	12
-3	100001	6	6	100001	6	10	1000000000	10	ZR46	100000000001	12
-4	100011	6	ZR5	100011	6	ZR26	1000000001	10	ZR47	100000000011	12
-5	100111	6	7	100111	6	-10	1000000011	10	ZR48	100000000111	12
-6	101000	6	ZR6	101000	6	ZR27	1000000111	10	ZR49	100000001000	12
-7	101111	6	-1	101111	6	11	1000001000	10	ZR50	100000001111	12
8	1000000	7	ZR7	1000000	7	ZR28	1000001111	10	ZR51	100000011000	12
-8	1000001	7	-2	1000001	7	-11	1000011000	10	ZR52	100000011111	12
9	1000011	7	ZR8	1000011	7	ZR29	1000011111	10	ZR53	100000111000	12
-9	1000111	7	-3	1000111	7	12	1000111000	10	ZR54	100000111111	12
10	1001000	7	ZR9	1001000	7	ZR30	1000111111	10	ZR55	100001111000	12
-10	1001111	7	-4	1001111	7	-12	1001111000	10	ZR56	100001111111	12
11	1011000	7	ZR10	1011000	7	ZR31	1001111111	10	ZR57	100011111000	12
-11	1011111	7	-5	1011111	7	13	1010001000	10	ZR58	100011111111	12
12	10000000	8	ZR11	10000000	8	ZR32	1011111000	10	ZR59	100110001000	12
-12	10000001	8	-6	10000001	8	-13	1011111111	10	ZR60	100111111000	12
13	10000011	8	ZR12	10000011	8	ZR33	10000000000	11	ZR61	100111111111	12
-13	10000111	8	-7	10000111	8	14	10000000001	11	ZR62	101110001000	12
14	10001111	8	ZR13	10001111	8	ZR34	10000000011	11		101111111000	12
-14	10011000	8	ZR14	10011000	8	-14	10000000111	11		101111111111	12
15	10011111	8	ZR15	10011111	8	ZR35	10000001000	11			
-15	10111000	8	ZR16	10111000	8	15	10000001111	11			
16	10111111	8	ZR17	10111111	8	ZR36	10000011000	11			
-16	100000000	9	ZR18	100000000	9	-15	10000011111	11			
			ZR19	100000001	9	ZR37	10000111000	11			

code generated by Scholtz's recursive method.
Max code length = 12
Start: {0,1}
Atoms: {0,1,1000}

The coding order of DCT coefficients is:

{DC coefficients of all blocks} followed by

{AC coefficients of block 1} followed by

{AC coefficients of block 2} followed by

{AC coefficients of block 3} … etc.

We output all dc coefficients first because we do not want the decoder to switch reading the dc and ac coding tables back and forth. Otherwise when there are errors, the decoder may read the wrong tables intermittently.

4.3.2 Results

	Q = 50		Q = 75	
Images	**Scholtz**	**JPG**	**Scholtz**	**JPG**
Baboon	50511	45753	76298	68933
Barb	32805	29648	47316	43393
Boat	27897	24367	40769	36272
Bridge	46202	41317	69811	62923
Couple	32526	29188	48432	43840
Crowd	32980	28810	46740	41862
Girl	25595	22511	37129	33310
Goldhill	31525	27449	47228	42004
Lake	33591	29396	50286	44917
Lena	23925	20921	36214	32568
Man	31727	28102	47427	42908
Peppers	24441	21310	37588	33722
Plane	26303	22602	38776	33826
Tiffany	28734	25642	44638	40879
Woman	17606	14323	26104	22235
Zelda	20170	17262	30329	27008
total bytes	486538	428601	725085	650600
Vs jpeg	+13.52%	-	+11.45%	-

Table 4-1 Image Coding Using Scholtz's code

Table 4-1 records the bit rate of using the Scholtz code on the DCT coefficients. The bit rate of the baseline JPEG Huffman model is also included for comparison. As in last chapter of the application of LZ78, both the JPEG and our Scholtz system are losslessly compressing the identical set of quantized DCT coefficients.

In terms of coding efficiency we have improved over the modified LZ78. The extra bytes, compared with JPEG, dropped from over 20% to 13% at a quality factor of 50. At a quality factor of 75, the extra bytes are about 11%. This coding gain comes from the use of variable word length. The DCT coefficient exhibit highly skewed distributions with most coefficients are of small magnitudes. Thus, the shorter codewords (mostly with length less than 5) of the synchronizable code already contribute to the most share of the bit count. This also explains the choice of the atoms in the construction of our synchronizable code. In every suffix construction procedure, if we remove a shorter word we can add more new codewords to our dictionary (by appending more repetitions). Thus, we remove 0 and 1 in the first two steps. After that we want to keep the shorter codewords since it will contribute most to the bit count, therefore we choose to remove a codeword of length 4 rather than length 2 or 3. On the other hand, if we remove a codeword of length 5, the result dictionary size will not be large enough.

Next we will demonstrate the use of self-synchronizable codes in conjunction with the bi-directional decoding technique as introduced in the RVLC (reversible variable length code). Conditions with random channel erasures and errors are both simulated. Erasures are also important because when there are uncorrectable error-patterns, a block channel decoder (say a RS decoder) may output erasures. While a convolution decoder (which finds the most probable code path by the Viterbi algorithm) always outputs a codeword, which may contain random errors. All the images will be subjected directly to the applied errors and no channel coding is used.

Images	Fwd only	Bi-directional
Baboon	22.826736	25.551426
Barb	24.803165	28.721564
Boat	26.218367	31.098192
Bridge	23.356335	26.073067
Couple	25.944527	28.955846
Crowd	25.290572	29.646251
Girl	28.846805	32.568283
Goldhill	27.683424	31.839993
Lake	24.283356	28.347524
Lena	27.695152	33.145045
Man	27.068229	30.542521
Peppers	26.814051	31.111188
Plane	24.807193	29.968025
Tiffany	27.226883	29.758860
Woman	29.483727	35.763666
Zelda	29.690868	33.472342

(a)

Images	Fwd only	Bi-directional
Baboon	18.536957	22.843518
Barb	20.545385	24.312417
Boat	21.911535	26.629913
Bridge	19.071453	21.951153
Couple	22.377877	26.119982
Crowd	20.108135	22.415568
Girl	23.549027	26.515470
Goldhill	21.031646	24.393998
Lake	20.957058	22.653602
Lena	22.758831	25.450534
Man	21.95351	25.414182
Peppers	23.153029	26.163731
Plane	22.905153	25.026695
Tiffany	21.228916	24.820463
Woman	24.207779	26.010601
Zelda	23.780936	25.624487

(b)

Table 4-2 Forward decoding vs Bi-directional decoding of images coded at Q=50 PSNR(dB) at (a) 0.1% erasures (b) 0.1% errors

Table 4-2 gives the peak signal-to-noise ratio (PSNR) of the decoded images at 0.1% random erasures and 0.1% random errors respectively. With bi-directional decoding, we can recover part of those data that would be discarded when only forward decoding is used. Hence, we see that there is an improvement in the PSNR for all the test images.

Figure 4-3 to figure 4-11 shows some decoded images of Lena and Boat at different error rates for visual comparisons. The recovery of data brought by the use of bi-directional decoding is visualized as the removal of those blurred blocks in the case of erasures or checkers-like artifacts in case of errors.

From the PSNR vs BER curves, we see that there is a consistent improvement in the PSNR over a wide range of error rates for both images Lena and Boat. The power of bi-directionally decoding is more significant, in terms of both the PSNR number and perceptual quality, at error rate not less than 0.1%. The comparative advantage of bi-directional decoding diminishes as the error rate increases. For instance, the image Boat at a bit error rate of 0.1%, has a gain of about 5dB from bi-directional decoding. While this gain drops to 2dB when the BER rises to 1%. This is because when the error rate is too high (~1%), the errors will appear near the ends of the synchronization points more probably. This limits the amount of data recovered by the bi-directional decoding. Yet, at such high error rate, the decoded picture is already at rather low quality (the Boat is at a PSNR of 14dB to 16dB), an improvement by 2dB does not help much perceptually in this case.

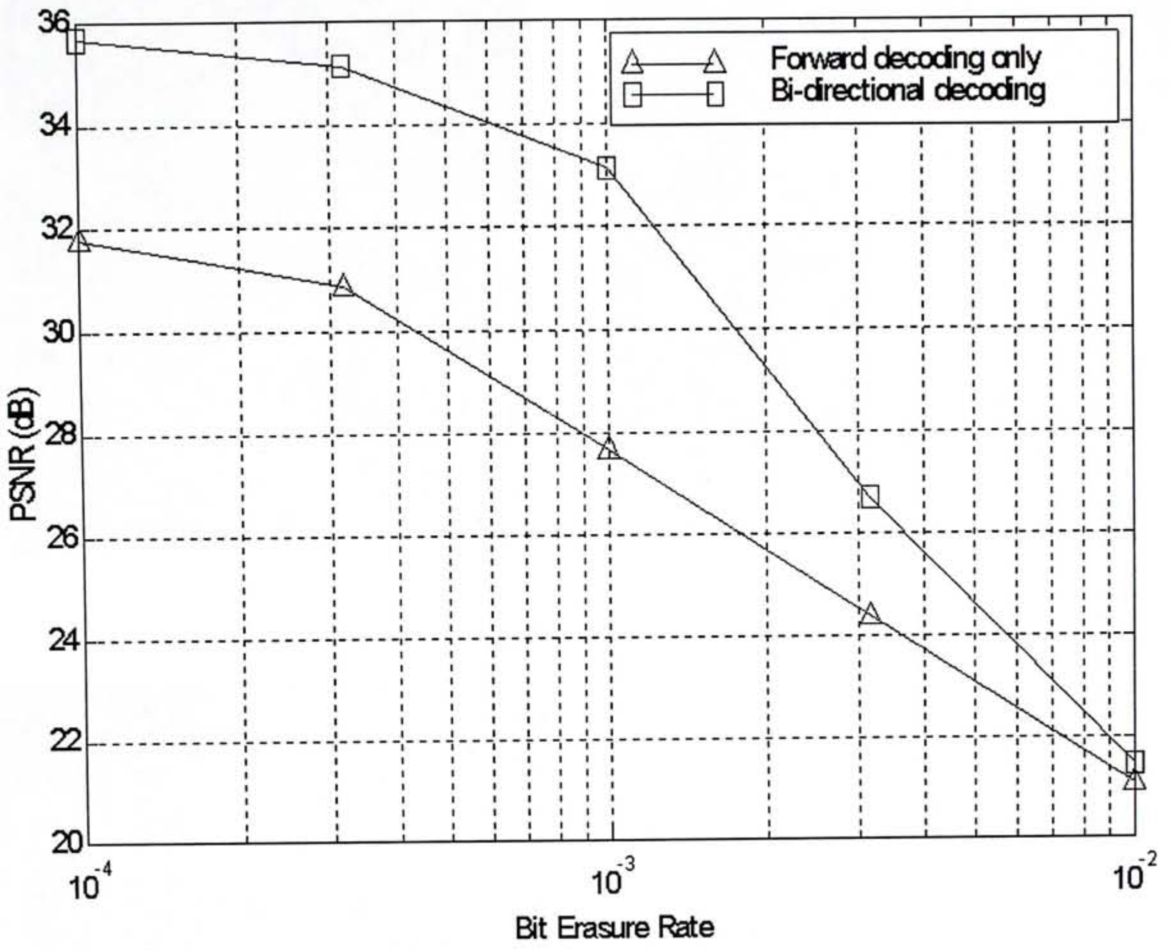


Figure 4-2 PSNR at different Bit Erasure Rate of Lena coded at Q=50 (0.73 bpp)

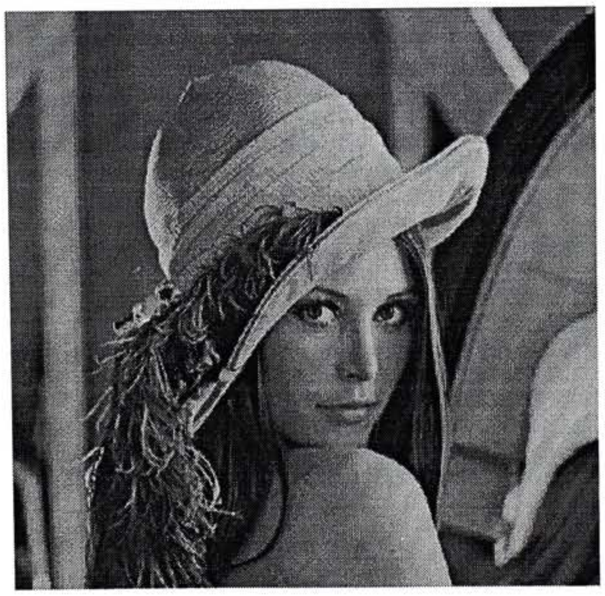
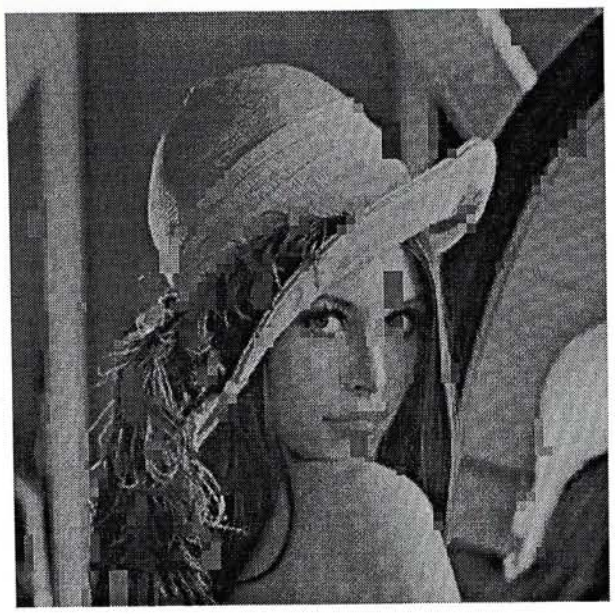
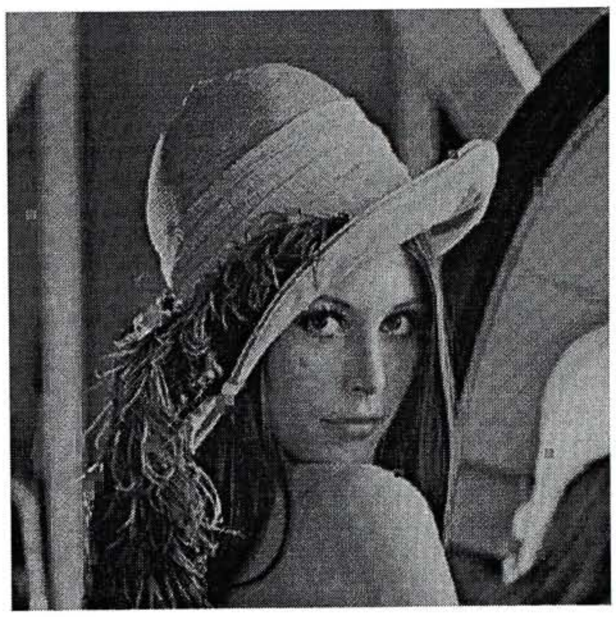
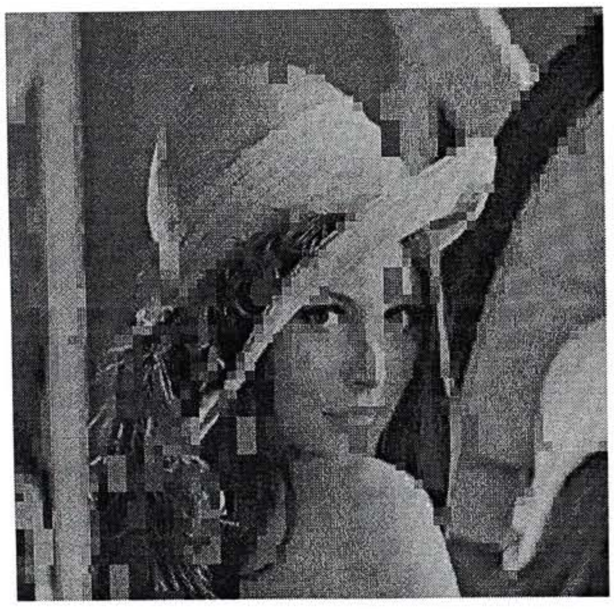


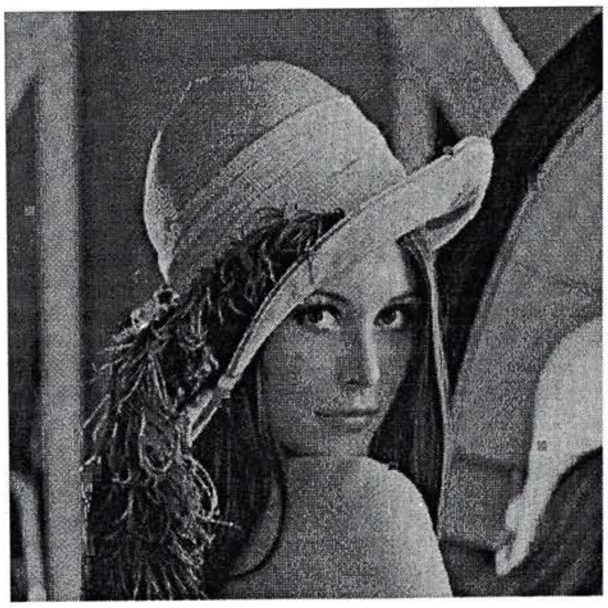
Figure 4-3 Error free Lena at Q=50





(a) (b)





(c) (d)

Figure 4-4 Lena

With 0.1% erasures (a) forward decoding only (b) bi-directional decoding

With 0.3% erasures (c) forward decoding only (d) bi-directional decoding

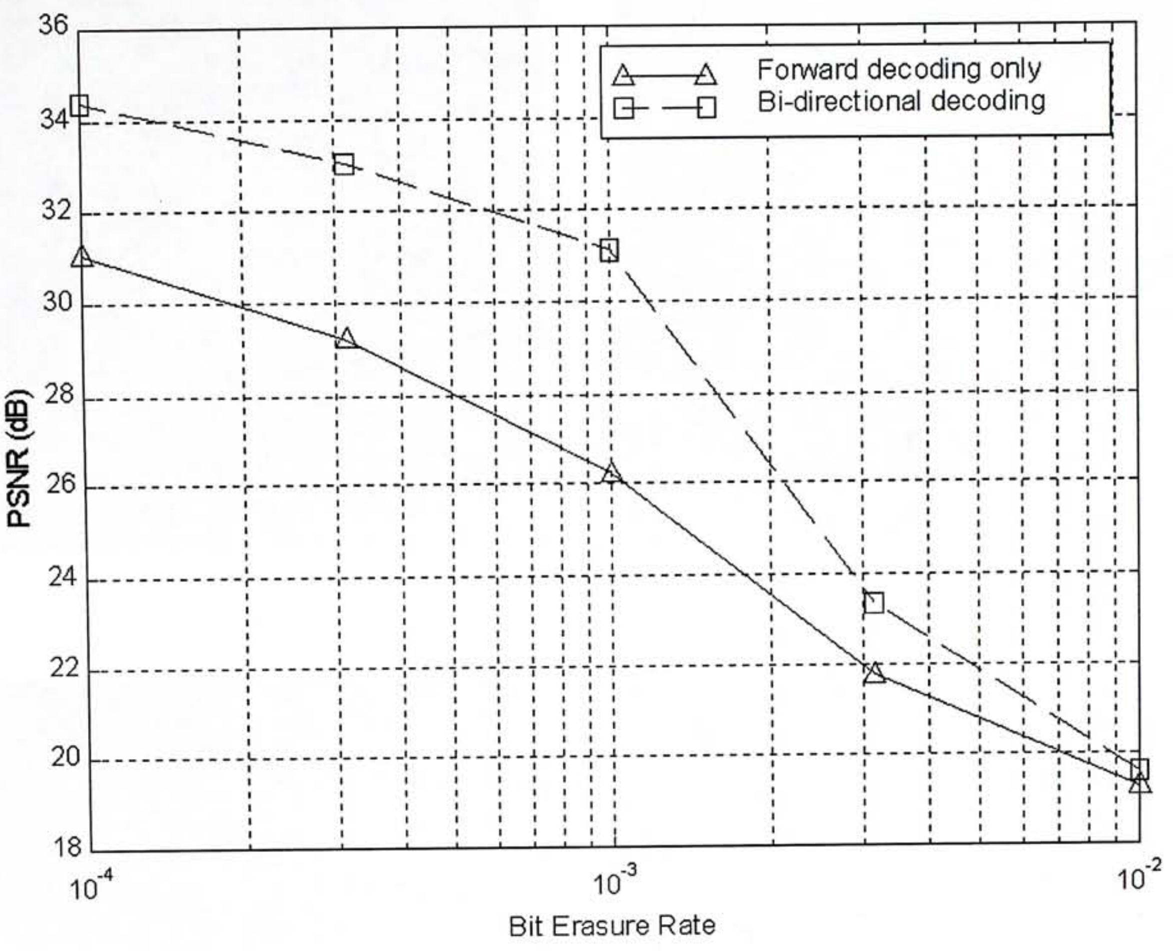


Figure 4-5 PSNR at different Bit Erasure Rate of Boat coded at Q=50 (0.85 bpp)

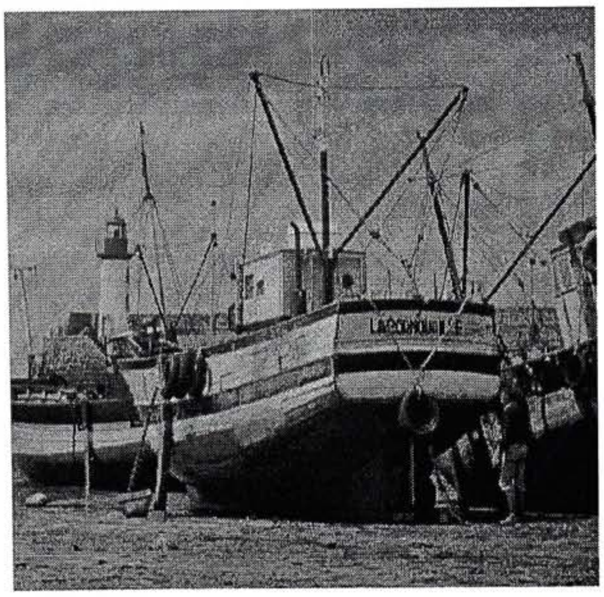
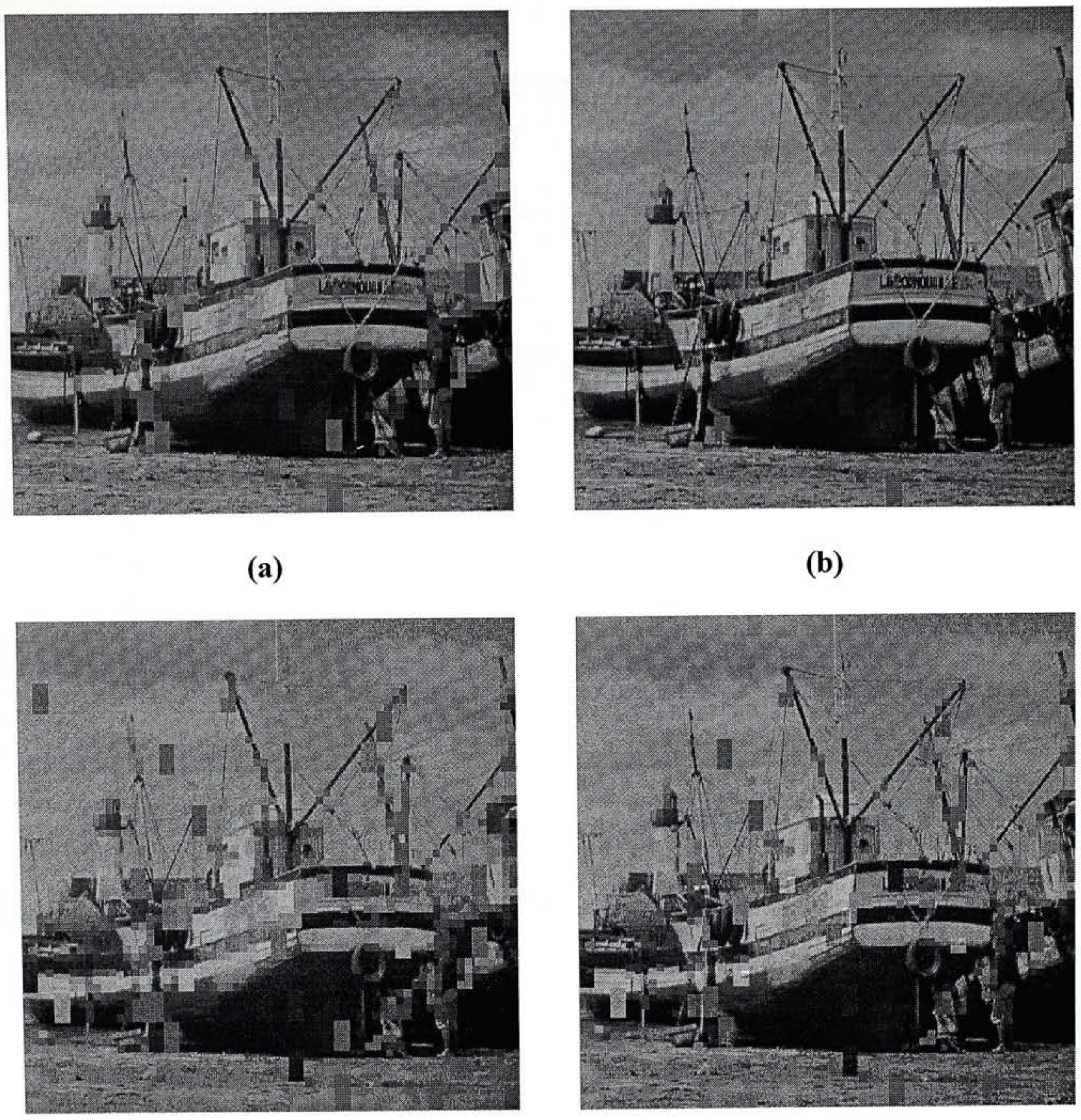


Figure 4-6 Error free Boat at Q=50



(a) **(b)** **(c)** **(d)**

Figure 4-7 Boat

With 0.1% erasures (a) forward decoding only (b) bi-directional decoding

With 0.3% erasures (c) forward decoding only (d) bi-directional decoding

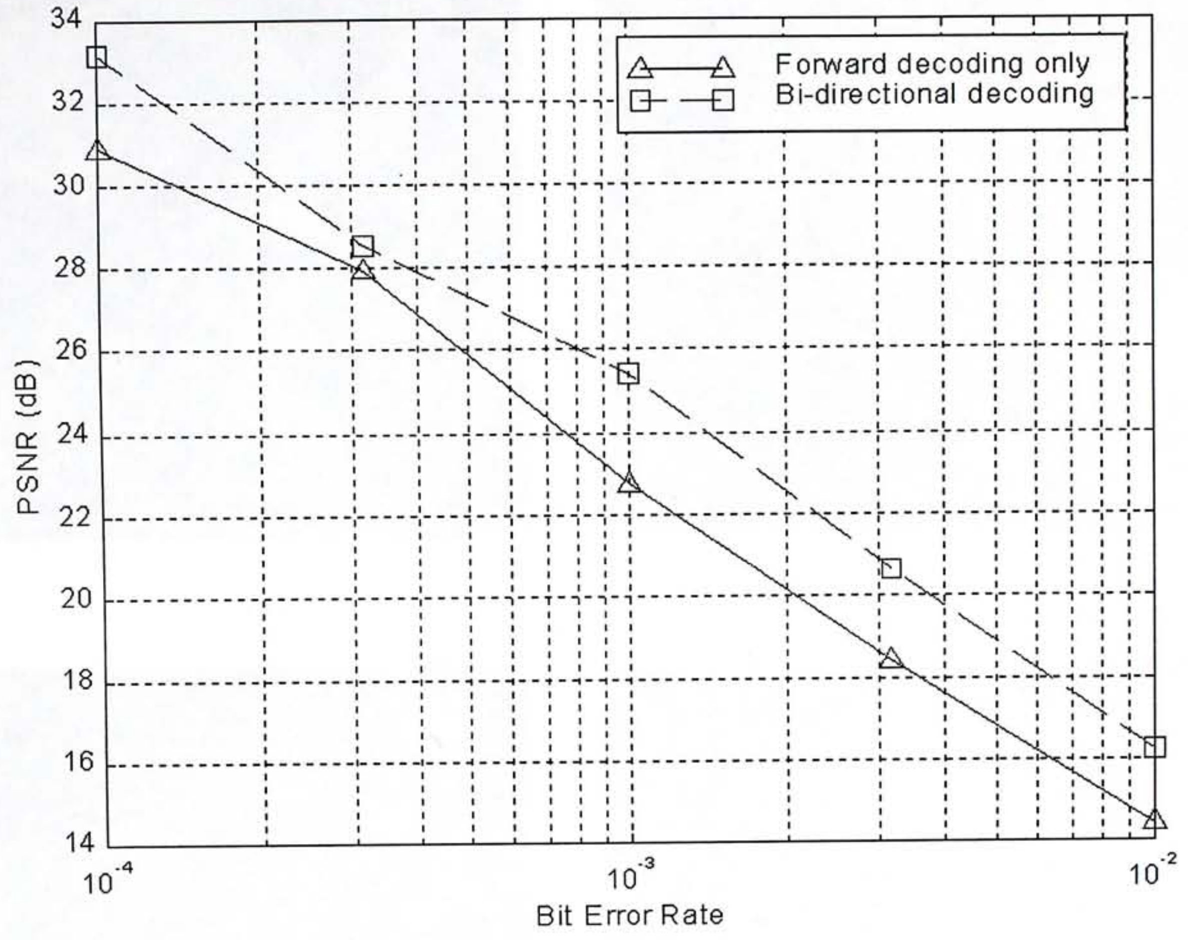
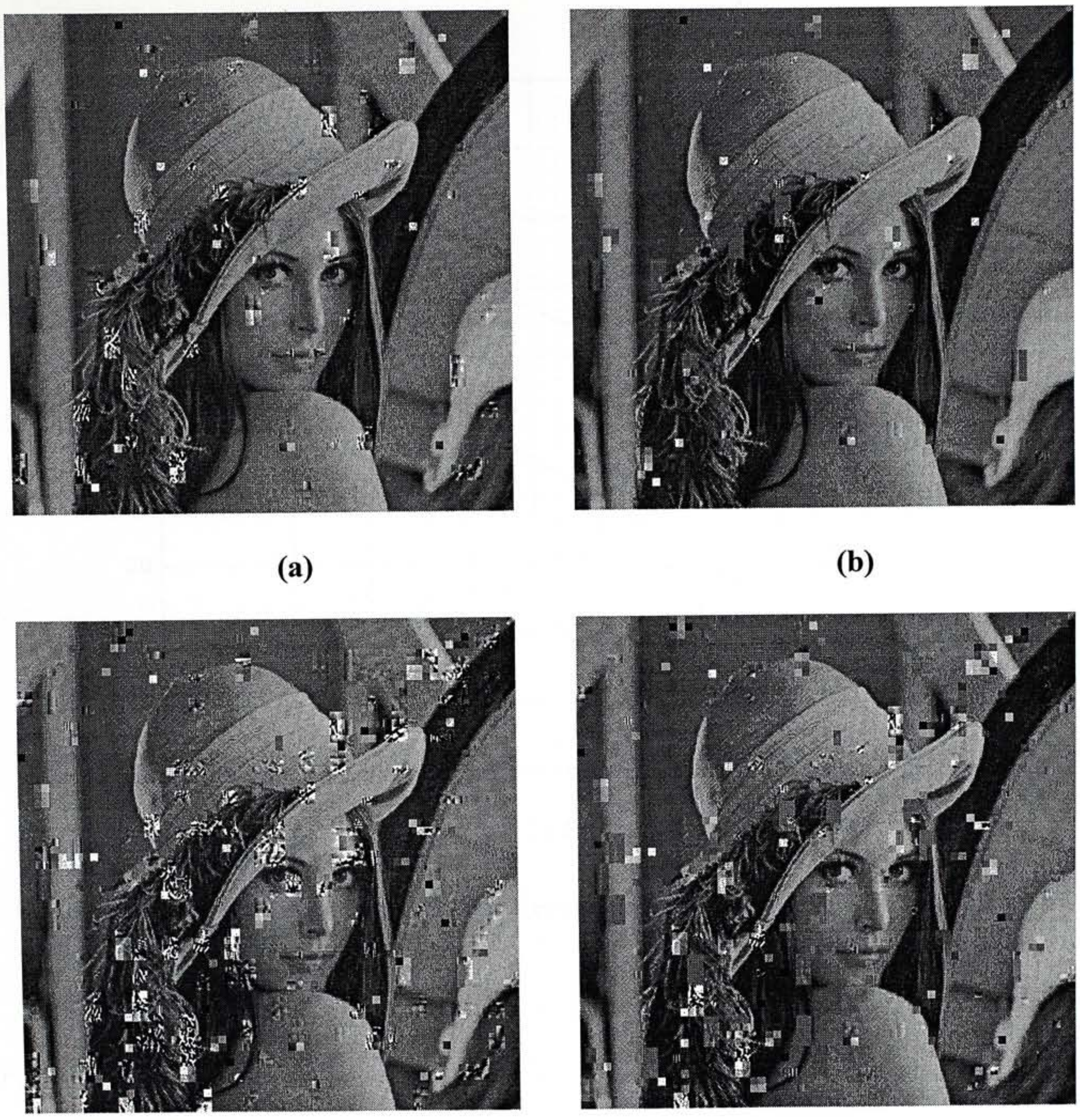


Figure 4-8 PSNR at different Bit Error Rate of Lena coded at Q=50 (0.73 bpp)



(a) (b)

(c) (d)

Figure 4-9 Lena

With 0.1% errors (a) forward decoding only (b) bi-directional decoding

With 0.3% errors (c) forward decoding only (d) bi-directional decoding

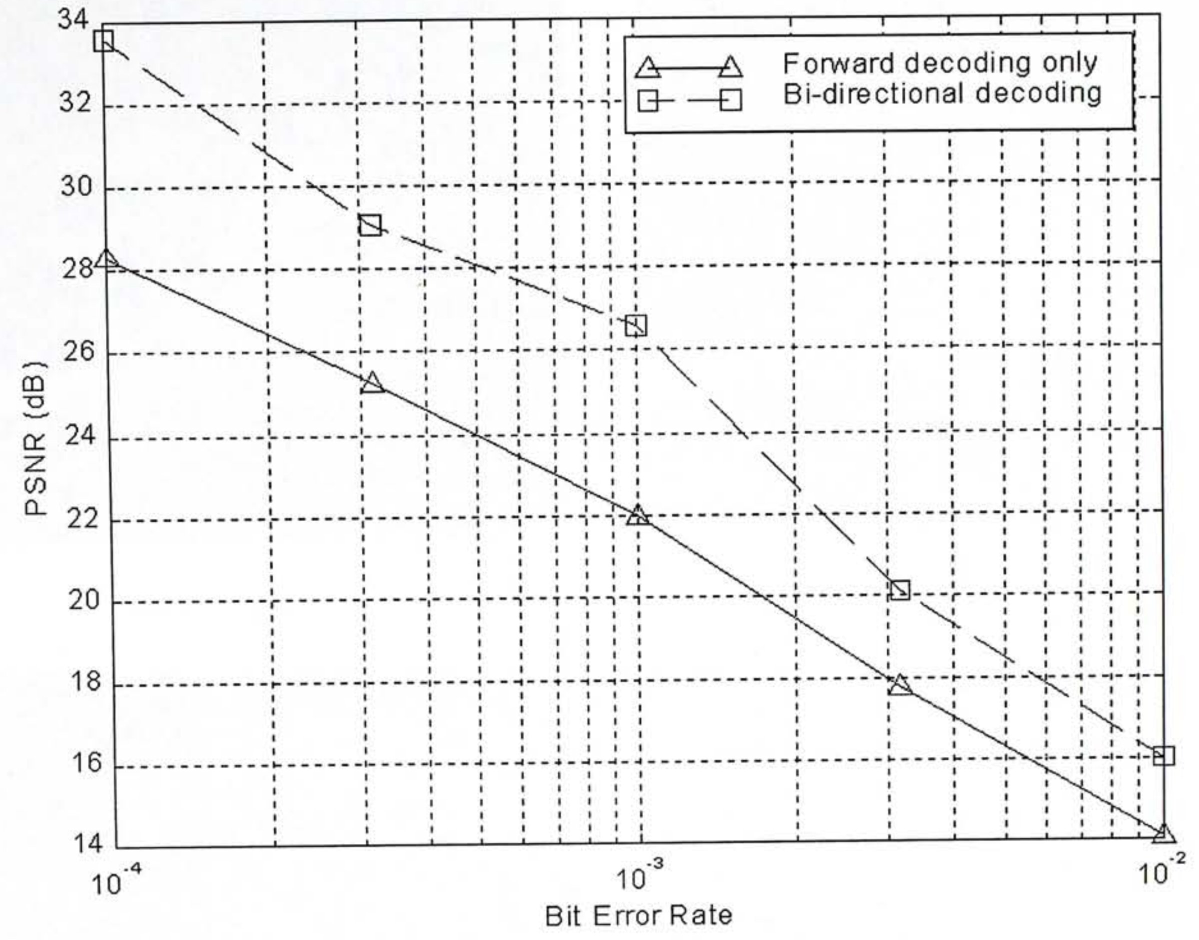
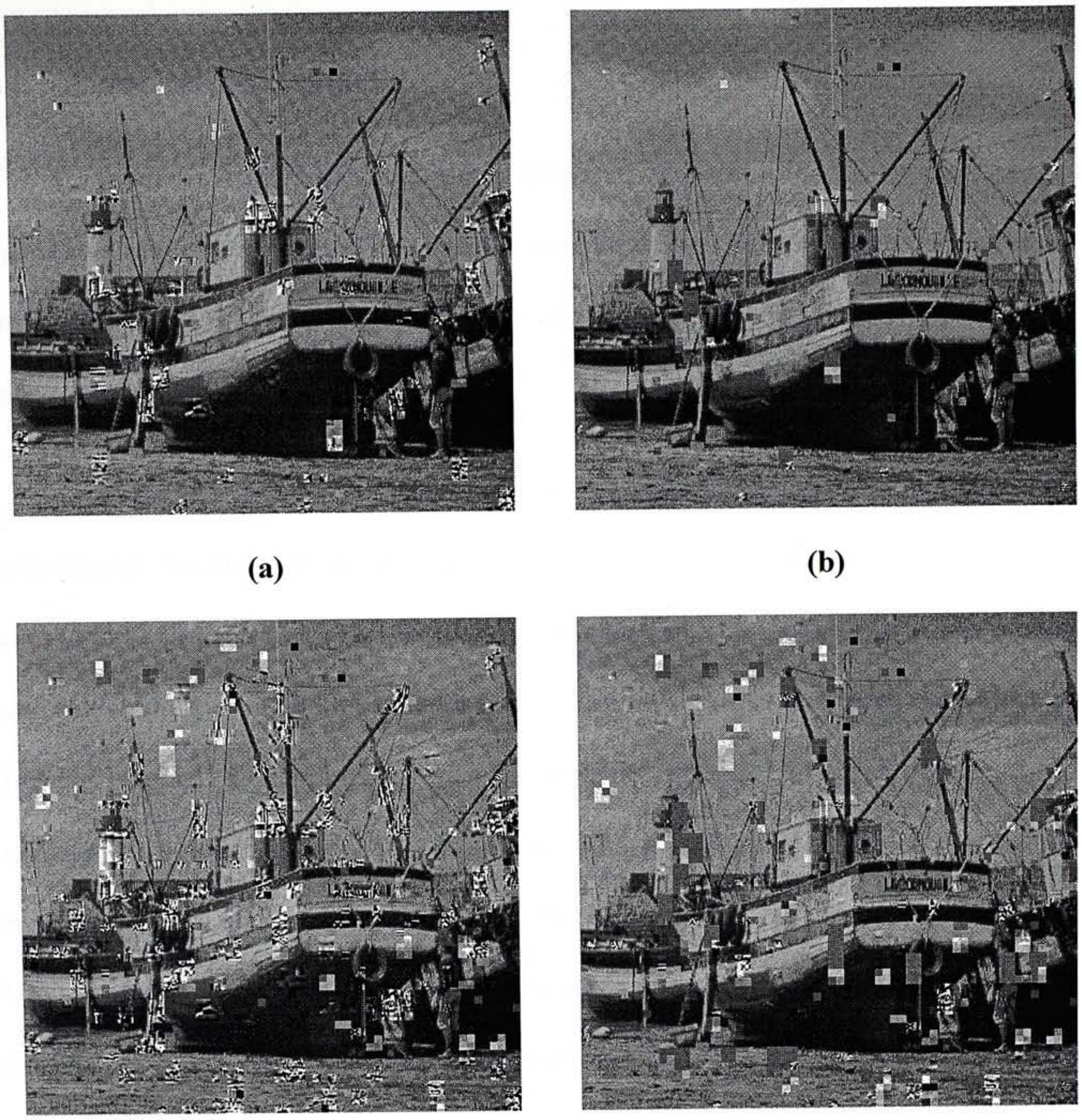


Figure 4-10 PSNR at different Bit Error Rate of Boat coded at Q=50 (0.85 bpp)



(a) (b)

(c) (d)

Figure 4-11 Boat

With 0.1% errors (a) forward decoding only (b) bi-directional decoding

With 0.3% errors (c) forward decoding only (d) bi-directional decoding

4.4 Concluding Remarks

Compare with the EREC algorithm proposed by Redmill and Kingsbury [17], our algorithms introduce about 10% more bit rates. However, our algorithms do not have to buffer the whole set of DCT coefficients and has simpler decoding complexities and operate faster.

In terms of codeword synchronizability, the self-synchronizable code offers similar performance. We still can decode bi-directionally or recover those data between two error points. With respect to limitation of error propagation, the self-synchronizable code is less stringent than fixed length codes. An error may affect the next codeword also. However, the self-synchronizable code has an extra advantage of being not afraid of the insertion or deletion errors. This is because the synchronization procedure can start anywhere we like and does not depend on what comes before. In some channels, like CD-ROM, which employs some form of run-length coding, bit insertions and deletions are possible. The self-synchronizable codes will provide protection against these kinds of errors.

Chapter 5 Conclusions

In this thesis, we described the error resilience tools provided in MPEG-4. These tools enable the robust transmission of video in error-prone environments. These techniques include variable macroblock slices, RVLC, partitioning of texture and motion data etc. These error resilient tools are mainly applied to the data layer while we assume that FEC codes are already provided in the system layer. They all aim to mitigate the effect of residual errors (after channel decoding) on the decoded video frames.

Compressed data is vulnerable to errors because of the loss of codeword synchronization when error occurs. We have studied two methods to deal with this problem. We first try the LZ78 code, which is a variable-to-fixed length code.

Next, we use Scholtz's class of self-synchronizable code. The major advantage is we can start the synchronization procedure from anywhere within the bitstream. Although it is inflexible to change Scholtz's code length distribution, it does match the DCT coefficient distribution satisfactorily. In our experiment, we only lag the baseline Huffman JPEG (in terms of bitrate) by 13.5% at a quality factor of 50 and 11.5% at 75 respectively.

Together with the technique of bi-directional decoding, we are able to recover more data in case of errors. This is beneficial to other error concealment methods that further increase the effective quality.

References

[1] Weidong Kou, *"Digital Image Compression Algorithms and Standards,"* Kluwer Academic Publishers.

[2] Hseuh-Ming Hang, John W. Woods, *"Handbook of visual communications,"* Academic Press.

[3] Theodore S. Rappaport, *"Wireless Communications, Principle & Practice,"* Prentice Hall.

[4] William B. Penneaker, Joan L. Mitchell, *"JPEG Still Image Data Compression Standard,"* Van Nostrand Reinhold 1993.

[5] Thomas Sikora, *"More on MPEG-1 and -2,"* available online at http://wwwam.HHI.DE/mpeg-video/papers/sikora/mpeg1_2/mpeg1_2.htm

[6] Thomas Sikora, *"MPEG-4 Overview,"* available online at http://wwwam.HHI.DE/mpeg-video/standards/mpeg-4.htm

[7] Yih-Fang Huang, Che-Ho Wei, *"Circuits and Systems in the Information Age,"* Short Courses at the 1997 IEEE International Symposium on Circuits and Systems.

[8] Wen and J. Villasenor, *"A Class of Reversible Variable Length Codes for Robust Image and Video Coding,"* Proceedings of the IEEE International Conf. on Image Proc., Santa Barbara, CA, Vol 2, pp 65-68, Oct. 1997.

[9] B. P. Tunstall, *"Synthesis of noiseless compression codes,"* Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, 1967.

[10] J. Ziv and A. Lempel, *"A universal algorithm for data compression,"* IEEE Trans. Inform. Theory, vol. IT-23, pp337-343, 1977.

[11] J. Ziv and A. Lempel, *"Compression of individual sequences via variable-rate coding,"* IEEE Trans. Inform. Theory, vol. IT-24, pp530-536, 1978.

[12] Serap A. Savari, Robert G. Gallager, *"Generalized Tunstall Codes for Sources with Memory,"* IEEE Trans. Information Theory, vol. 43, No. 2, March 1997.

[13] A. Scholtz, *"Codes with synchronization Capability,"* IEEE Transactions on Information Theory, vol. IT-12, No. 2, pp. 135-142, April 1966.

[14] Robert A. Scholtz, *"Maximal and Variable Word-Length Comma-Free Codes,"* I IEEE Transactions on Information Theory, vol. IT-15, No. 2, pp. 300-306, March 1969.

[15] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg and Didier J. LeGall, *"MPEG Video Compression Standard,"* Chapman and Hall.

[16] Barry G. Haskell, Atul Puri and Arun N. Netravali, *"Digital Video: An Introduction to MPEG-2,"* Chapman and Hall.

[17] David W. Redmill and Nick G. Kingsbury, *"The EREC: An Error-Resilient Technique for Coding Variable-Length Blocks of Data,"* IEEE Trans. On Image Processing, vol. 5, No. 4, April 1996.

[18] Raj Talluri, *"Error-Resilient Video Coding in the ISO MPEG-4 Standard,"* IEEE Communications Magazine, June 1998.

[19] Independent JPEG Group (ILG), *Portable C code for JPEG Compression,* available at ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz

[20] Dr. Wei-ping Li, *private communications*