Low Delay MPEG2 Video Encoding

by

Tri D. Tran

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in Electrical Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

February 3, 1997

© Copyright 1997 Tri D. Tran. All Rights Reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author .....
Department of ~~Electrical Engineer~~ing and Computer Science
February 3, 1997

Certified by .....
Andrew B. Lippman
Lecturer, Associate Director, MIT Media Laboratory
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAR 21 1997

LIBRARIES

Low Delay MPEG2 Video Encoding

by

Tri D. Tran

Submitted to the
Department of Electrical Engineering and Computer Science

February 3, 1997

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

High-quality and low-delay MPEG2 video coding can be achieved by avoiding the use of intra (I) and bidirectional prediction (B) pictures. Such coding requires intra macroblocks refreshing techniques for channel error propagation resilience and for compliance with the accuracy requirements of the MPEG2 standard. This project gives an overview of the MPEG technology, describes some of these low delay coding techniques and presents software simulation results of their performance in terms of image quality and their robustness to transmission channel errors. Trade-offs and comparisons among various coding strategies are analyzed. Based on the test sequence "Susie," it is shown that the simple low delay coding algorithm gives an overall visual quality acceptable to general viewers even at relatively low bit rate, desirable delay, and under severe transmission error environments.

Thesis Supervisor: Andrew B. Lippman
Title: Lecturer, Associate Director, MIT Media Laboratory

To my parents and to Maria

Acknowledgments

This thesis project is carried out at the IBM Thomas J. Watson Research Center located in Yorktown Heights, New York. It lasts from the beginning of June 1996 to the end of January 1997. I started writing the thesis document itself during early November and I owe many people, in and outside of work, for the completion of it. Please forgive me if I forget to list anyone.

Firstly, I would like to thank Cesar Gonzales at IBM Research for giving me the opportunity to work with his group on this particular project and guiding me through it. Without his help, I would not have a thesis to work on in the first place. I also would like to thank my MIT thesis advisor Andy Lippman for accepting me as his advisee, giving me advice and proofreading my thesis.

I cannot forget to thank my two mentors at IBM Research--Lurng-Kuo Liu and Peter Westerink. Lurng-Kuo has helped me through with many concepts about the MPEG technology in general and he also guided me through my low delay coding project. I normally meet with him on Monday mornings to discuss my progress and what to expect next. On the other hand, Peter has helped me tremendously with the MPEG software codec. Questions about coding techniques, data interpretation, and many other topics are answered. I often just run to either Lurng-Kuo's or Peter's office whenever I have a new question. In short, I owe much of the thesis completion to these two persons.

Other people in the group who have helped me through with my project are Rajesh Rajagopalan and Lascoe Allman. Rajesh has helped me proofread my technical paper and also pointed out helpful hints about my project. Lascoe readily mounts and dismounts any CDs and directories for me whenever asked. Other members of the department are always interested in knowing my progress.

Outside of work, I would like to thank Barbara Lynds for giving me the greatest hospitality throughout the fall and winter months. She has always been the person I can come up and talk to about my daily events, or about anything at all. She even fixes me food and medicine whenever I don't feel well.

Maria Su has been the love of my life, way before the project even started. She has always been loving and giving me support through whatever I decide to do. Please forgive me for those times when my work seems to take over our lives. You know better than anyone else that it does not. :-) I love you with all my heart!

My parents and my brother have always been supporting me throughout. My brother Trac has always given me advice about my work in general. My parents, being so far away in Vietnam, have given me all the emotional support and unsurpassed love any parents can give to a child they haven't seen for more than a decade. Hopefully, the day of our family reunion isn't so far away...

Table of Contents

1 Introduction

Started in 1988, the Moving Picture Experts Group (MPEG) committee set a goal to standardize a video coding strategy for video storage and retrieval for multimedia applications. Not only did the committee succeed with the standardization of MPEG1 [3], the introduction of MPEG2 near the end of 1994 surpassed MPEG's original goal and the video compression technology has grown to become one of the preeminent image representations, suitable for many high quality video applications including broadcasting, consumer electronics, and telecommunications.

In particular, many multimedia applications such as videoconferencing demand high quality real-time video with little or no noticeable lag. Previous works performed on low delay video coding, such as videoconferencing H.261 and H.263 standards, constrain many parameters as to achieve maximum efficiency at very low bit rates. For example, image size, frame rate, bit rate, and buffer size are all limited, which result in significantly lower quality than that of MPEG which requires higher bit rates. With the emergence of high-bandwidth communication channels, low delay applications will be able to take advantage of the advances in coding algorithms incorporated in the International MPEG Standards. We are motivated to further explore MPEG2 high video quality while keeping a minimum end-to-end delay--a requirement for low-delay applications such as videoconferencing--and fully conforming to the MPEG2 standard [4]. Such low delay coding was planned for in the MPEG2 standard but has been under-tested. More importantly, having an MPEG2 low delay encoder provides a superset of all other coders (i.e., H.261, H.263, and MPEG1); its broader degrees of freedom also supports higher quality and a wider range of applications.

As a brief overview, the MPEG standard IPB coding scheme is not well suited for low delay coding applications; this is particularly true for channels of constrained constant bit

rates. To achieve good picture quality, I pictures require large number of bits as compared to P and B pictures; this results in a requirement for large encoder compressed data buffers which, in turn, translates into a larger end-to-end delay than would be necessary if the buffers were smaller. B pictures, on the other hand, typically provide higher compression efficiency and hence require small buffers; however, they can introduce several picture delays by the frame reordering they require. In other words, to code a B picture we must first wait and encode a future reference picture (I or P). To decode, we also must wait until a future reference picture is decoded before the corresponding B pictures can be processed, thus introducing several picture delays. The elimination of I and B pictures is the key to MPEG2's low delay mode and also to the design of the low delay coding algorithm.

As suggested in Appendix D (not an integral part of the standard) of the MPEG2 standard, low-delay performance can be achieved by compression with P pictures only, as long as all macroblocks in the video pictures are periodically refreshed by coding with the intra-macroblock type. The standard suggests, for example, the use of intra slice updating (Appendix D.5 in [4]). Intra column refreshing is also suggested in [2]. Neither the MPEG2 standard nor the Test Model 5[1] [2] describes exactly how such intra macroblock updating should be done. In our design, we periodically refresh the video pictures by selectively encoding regions (rows or columns) of macroblocks using the intra type. Intra macroblock refreshing is necessary to avoid the propagation of errors due to either encoder/decoder incompatibilities in the implementation of the IDCT, and to avoid the propagation of channel data errors. Because the intra regions by definition are not predictively coded from prior pictures, they essentially stop any error propagation.

1. Test Models are documents formulated by collective works from various individuals and organizations participating in the standardization of the MPEG technology.

In this thesis project, I plan to study the low delay design, implement it on a software simulated MPEG2 encoder, and evaluate its performance. Various refreshing schemes using intra frames, intra slices, and intra columns will be implemented and compared. Attempts to reduce transmission error degradation by using larger slice/column sizes and constrained motion estimation will also be presented and discussed. Other strategies for video quality improvement such as usage of concealment motion vectors and dual-prime prediction will be investigated. Software simulation experiments that illustrate the performance of low delay coding in terms of PSNR, visual picture quality, and resilience to errors will be carried out and presented.

This thesis is organized as followed. Chapter 2 describes the technology and basic differences between the MPEG1 and MPEG2 video compression standards. Analysis on the standard IPB coding scheme in terms of low delay is also made. Chapter 3 provides the design and implementation of the low delay coding algorithm and its complimentary features. Chapter 4 presents simulation results that illustrate the performance of low delay coding in terms of PSNR, visual picture quality, and resilience to errors. Chapter 5 gives a detailed interpretation of the results and offers further discussions. Finally, Chapter 6 gives the conclusions and suggestions for future research.

2 Overview of the MPEG Technology

This chapter gives an overview of the fundamentals of the MPEG technology. Both MPEG1 and MPEG2 will be discussed along with their major differences. How the standard coding algorithm is not suited for low delay coding will also be pointed out in a separate subsection. It is assumed that the reader has experience with one-dimensional signal processing and two-dimensional image processing. For popular texts on one and two-dimensional signal and image processing, please refer to [15] and [14] respectively. Relatively short papers regarding overview of the MPEG technology can be found in [16] and [17]. For readers with access to the Internet, *http://www.crs4.it/~luigi/MPEG/mpeg2.html* provides answers to many frequently-asked-questions about the MPEG technology.

2.1 The MPEG1 Standard

MPEG is an acronym for the Moving Picture Expert Group, which was chartered by the International Standards Organization (ISO) to standardize a coding representation of video and its associated audio suitable to digital storage and transmission media. The goal of the group is to develop a generic coding standard that can be used in many digital video implementations varying from CD-ROM storage to telecommunications networks. MPEG1's primary requirement is to achieve the highest possible quality of the decoded motion video at a given bit-rate. Other requirements are random access capability, fast search, and support of a variety of video formats. Yet, a considerable amount of specification and tuning is still left for the designer of the specific application before it can be useful.

MPEG1 is an international standard for coding video and associated audio at bit-rates up to about 1.5 Mbits/sec [3]. Its official name is ISO/IEC 11172 (International Standards Organization/International Electrotechnical Commission). The specified 1.5 Mbits/sec bit

rate is not an upper bound. In fact, MPEG1 can be coded at rates as high as 100Mbits/sec. However, the 1.5 Mbits/sec bit rate turns out to be the optimal coding bit rate (with the highest level of compression and video quality) for progressive color sequences. Typical (but not limited to) Source Input Formats (SIF) are NTSC (352x240 pixels at 29.97 frames/sec) and PAL (352x288 pixels at 25 frames/sec). Both of these inputs must use 2:1 color horizontally and vertically subsampling (4:2:0 format).

2.1.1 Overview of MPEG1 Compression Algorithm

The MPEG video compression technology relies on two fundamental techniques: block-based motion compensation for the reduction of temporal redundancy and DCT transform based compression for spatial redundancy reduction.

2.1.1.1 Reducing Spatial Redundancy

A compression method called intraframe coding is used where each video picture is individually and independently encoded. Intraframe coding exploits the spatial redundancy that exist among the adjacent pixels of a picture. Such intraframe coded pictures are called *I pictures*. For such coding, MPEG1 video-coding algorithm employs a block-based NxN two-dimensional DCT, defined as:

$$F(u,v) = \frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}\cos\frac{(2y+1)v\pi}{2N}$$

with u, v, x, y = 0, 1, 2, ... N-1

where x, y are spatial coordinates in the sample domain

u, v are coordinates in the transform domain

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for u, v} = 0 \\ 1 & \text{otherwise} \end{cases}$$

A picture is first divided into 8x8 blocks of pixels, and the DCT is applied independently

to each of the picture block. The result is an 8x8 block of DCT coefficients in which most of the energy in the original input block is concentrated in the DC and lower-frequency coefficients at the top left of the block. The higher-frequency coefficients are concentrated at the bottom right of this block.

The DCT coefficients are then quantized. This technique is primarily responsible for the lossy nature of MPEG, which is implemented by simply dividing a coefficient by a step size and rounding the quotient to the nearest integer. For MPEG1, all intra DC coefficients (top left coefficient of each block) are quantized with a uniform quantizer with fixed step size (divisor) 8. The intra AC coefficients, which are the remaining coefficients of the block, are quantized with a nearly uniform quantizer having variable step size. With this process, unquantized DC coefficients having differential values from -2040 to 2040 will be scaled to a range of -255 to 255. A simplified example of quantization is illustrated in Figure 2.1. Several values on the x-axis are represented by a single value on the y-axis. The values on the y-axis are the resulting quantized values. For instance, any unquantized

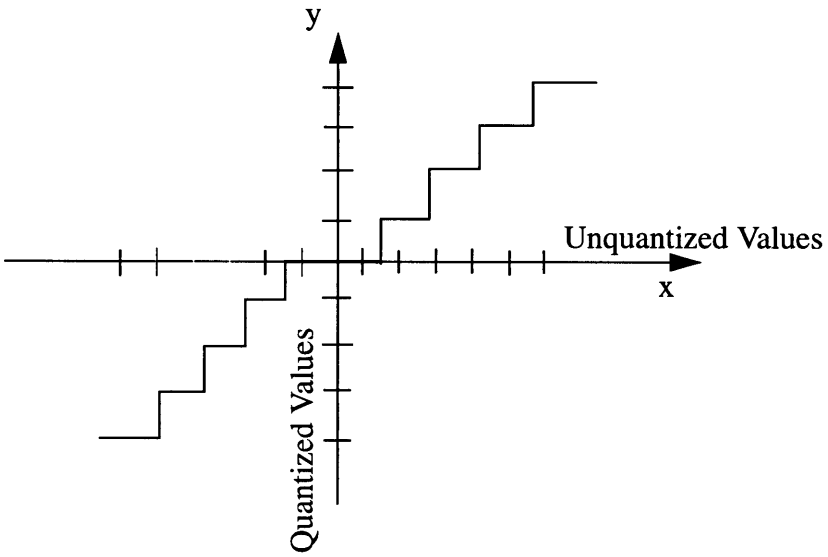


Figure 2.1: A Simple Example of Quantization.

value ranging from -1.5 to 1.5 is now 0. This "dead zone" located near the origin helps decrease noise sensitivity. Various types of quantization discussed in great details can be found in Chapter 4 of [13]. After quantization, many of these AC coefficients will typically become zero. Compression is achieved by passing on only nonzero coefficients which survived the quantization operation.

In order to convert these two-dimensional quantized data coefficients into one-dimensional data for further compression, a process called zigzag scanning to applied, where the default scanning order is illustrated in Figure 2.2. This operation rearranges the block of coefficients into the order shown for entropy coding, where shorter code words are assigned to quantized data which occur more often.

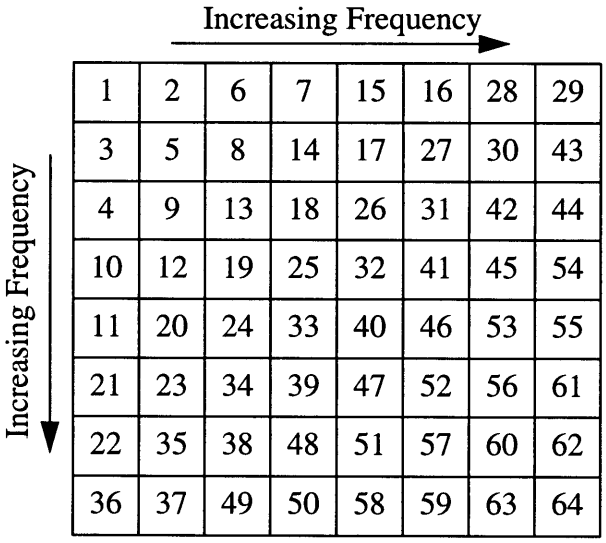


Figure 2.2: Default Zigzag Scanning Order for Transform Coefficients.

2.1.1.2 Reducing Temporal Redundancy

To further compress video signals at the desired bit rate of about 1.5 Mbits/sec, interframe coding is introduced. Temporal redundancy occurs when there is a strong correlation between adjacent pictures. For example, the background of a video sequence may not change at all through several video frames. MPEG1 exploits this redundancy by employ-

ing the technique of motion compensation and computing the interframe difference signal called the prediction error. A block of size 16x16 pixels called macroblock (MB) is introduced for motion compensation. This size is chosen as a trade-off between the compression provided by motion compensation and the overhead cost of transmitting the motion vectors.

For one type of motion estimation, called *forward prediction*, a *target* macroblock in the input picture to be coded is matched with a region of the same size in a past picture called the *reference* picture. The region in the reference picture that best matches the target macroblock is used as the prediction macroblock. The prediction error is computed by taking the difference between the target macroblock and the reference region. Note that this region usually does not align with coded MB boundaries in the reference frame (see Figure 2.3). The location of this best-matching prediction macroblock is determined by a motion vector that describes the displacement between it and the target macroblock. This motion vector is encoded and transmitted along with the prediction error which is coded with the DCT technique described above. Those pictures coded using this technique of motion compensation are called P pictures.

2.1.1.3 Bidirectional Temporal Prediction

A key feature of MPEG1 is bidirectional temporal prediction. Another name for it is motion-compensated interpolation. Pictures coded with this feature use two reference pictures--one in the past and one in the future. In other words, both forward and backward prediction are applied to code the target macroblock. Because a motion vector is associated with each prediction, two motion vectors are generated with bidirectional temporal prediction. This technique is illustrated in Figure 2.3. The two resulting motion vectors are averaged to obtain one effective motion vector.

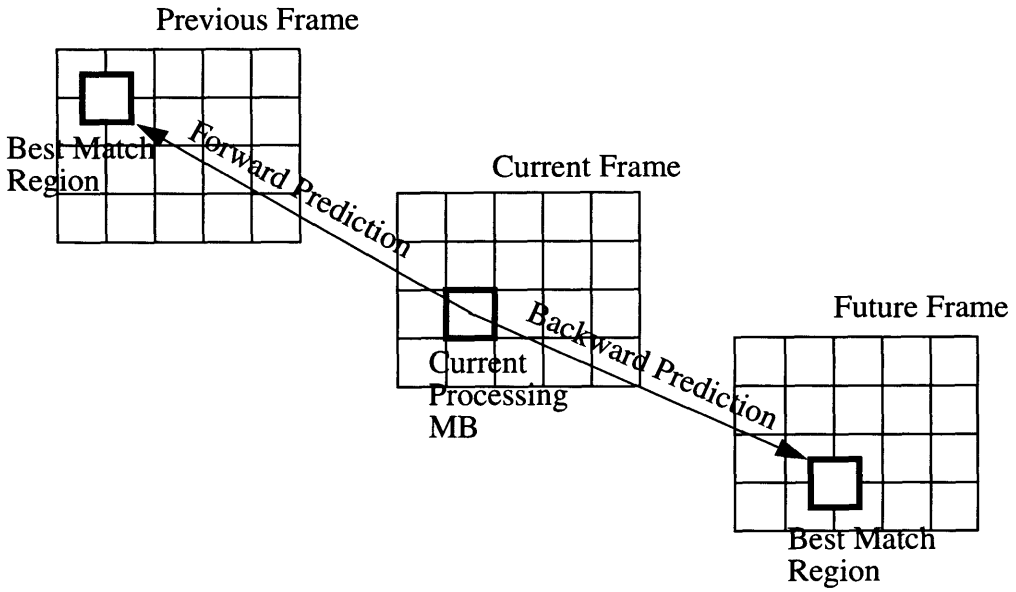


Figure 2.3: Bidirectional Prediction.

Pictures coded using bidirectional interpolation are called B pictures. These B pictures are never used as a reference picture to code any other picture. References for these B pictures must be either P or I pictures. Likewise, references for P pictures are also either P or I pictures. Recall that I pictures do not use any references.

Several advantages and disadvantages are associated with B pictures. The most important advantage is that B pictures allow for highest compression compared to P or I pictures. The same picture quality can be obtained using B pictures coded with fewer bits as to P or I pictures. Because B pictures are coded between two consecutive P pictures, the P pictures are now temporally located further apart, thus generally require more bits to code than the B pictures. Averaging two motion vectors for B pictures also contribute to noise reduction. As a result, most rate control algorithms allocate fewer bits to B pictures than P pictures, resulting in lower quality for B pictures compared to P ones, which appear as high frequency noise flicker and may be acceptable. Recall that B pictures are not used

as references and their lower quality does not propagate. Regarding disadvantages from using B pictures, coding a video sequence containing B pictures requires that the future reference picture is coded (decoded) before a B picture can be coded (decoded), causing frame reordering delay. Moreover, higher computation is associated with B picture coding since macroblock matching, a computationally expensive task, needs to be performed twice for each target macroblock--one with the past reference picture and one with the future reference picture.

2.1.2 The MPEG1 Bitstream Structures

There is no detailed specification for an MPEG1 encoder within the standard. Rather, the syntax of the encoded bitstream is given. The decoding process is also specified within the standard. As a result, a valid encoder is one which can generate a valid encoded bitstream. Details about algorithms such as the motion estimation matching process are not part of the standard and so an encoder designer has a high degree of flexibility. On the other hand, limitations are imposed for certain cases such as the constant bit rate coding, where the amount of bits generated for a video sequence may not underflow or overflow a Video Buffer Verifier (VBV). More explanation will be given about this VBV later.

In order for manufacturers to build a standard decoder for the most popular applications, MPEG1 has defined bounds for a set of parameters for the bitstreams, which are indicated in the Sequence Header parameters flag. These parameters bounds are listed in Table 2.1. However, the MPEG1 technology is not limited to these bounds.

The bitstream syntax is designed to support a wide variety of applications. Flexibility is a result of having multiple layers of headers and picture types, each having its own function. The layers are structured in a hierarchical fashion as shown in Figure 2.4 where each

layer begins with its unique start code and ends with its end code. These layers are illustrated in the subsections below and are summarized in Table 2.2.

Table 2.1: Upper Bounds for MPEG1 Constrained Parameters.

Parameter	Upper Bound
horizontal size	768 pels
vertical size	576 lines
number of MBs	396
MB rate	396*25 MB/s
frame rate	30 Hz
VBV Buffer	40 Kbytes
bit rate	1.856 Mbits/s

Table 2.2: Header Hierarchy of MPEG1 Video Bit-Stream Syntax.

Header	**Functionality**
Sequence	Context unit
Group of Pictures	Random access unit
Picture	Primary coding unit
Slice	Resynchronization unit
Macroblock	Motion compensation unit
Block	DCT unit

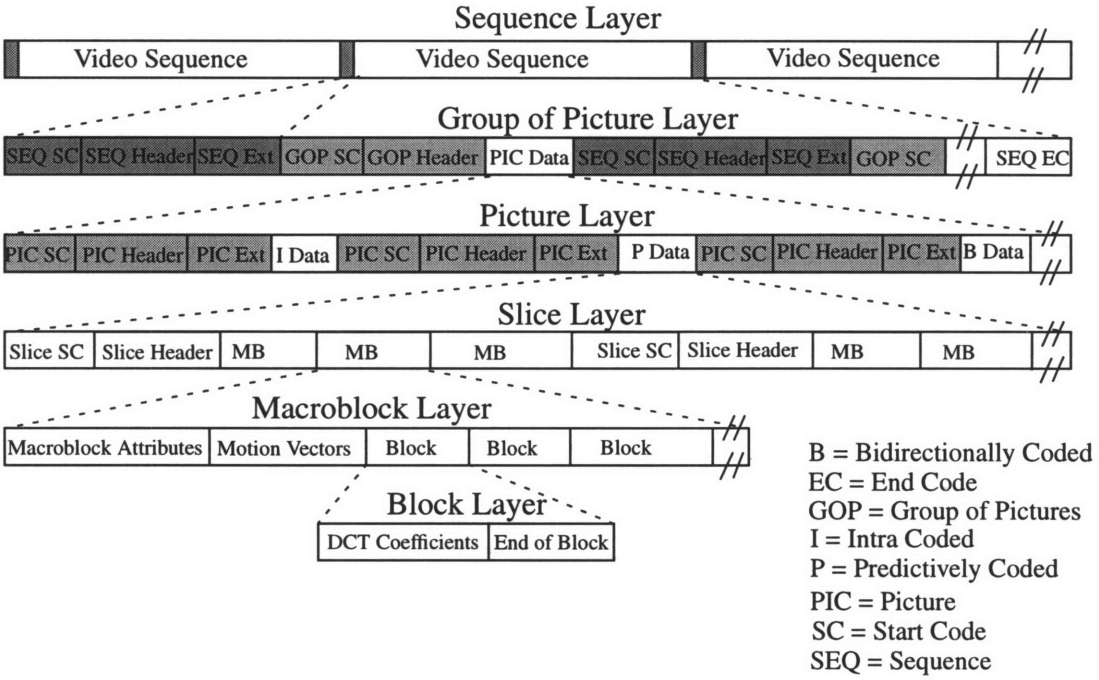


Figure 2.4: Hierarchical Structure of the MPEG Video Bitstream.

2.1.2.4 Video Sequence Header

The highest layer is the Video Sequence Header, which contains global parameters such as resolution of the video sequence, frame rate, initial VBV size, bit rate, etc... This header may also contain two quantizer matrices, one for Intra (I) pictures and the other for non-Intra (P or B) pictures. If no matrices are sent, the decoder should use the default matrices shown in Figure 2.5. These quantizer matrices show the step sizes used in the quantization process for the corresponding DCT coefficients. For intra coded pictures where DC and lower-frequency DCT coefficients (which has the most energy) are stored towards the top left corner of the block, note how the DC coefficient will be quantized with a much smaller step size (8) compared to the step size used for the highest AC coefficient (83), thus quickly eliminating the higher-frequency coefficients. On the other hand, a less step-size varying quantizer matrix is used to quantize prediction error coefficients for the inter

coded pictures[1]. Figure 2.6 illustrates the difference using 3-D plots.

Intra

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

NonIntra

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

Figure 2.5: Default Quantizer Matrices for Intra and Inter Pictures.

2.1.2.5 Group of Pictures (GOP) Header

Located within each Video Sequence are Group of Pictures (GOP) Headers and their data, which provide editing functions such as random access, and quick search. A bitstream is divided into a series of GOPs, where each GOP normally contains an I picture and several

1. Actually, the default quantizer matrix for inter coded pictures listed in the standard contains all 16's. However, the standard allows for the modification of that matrix to be the one presented.

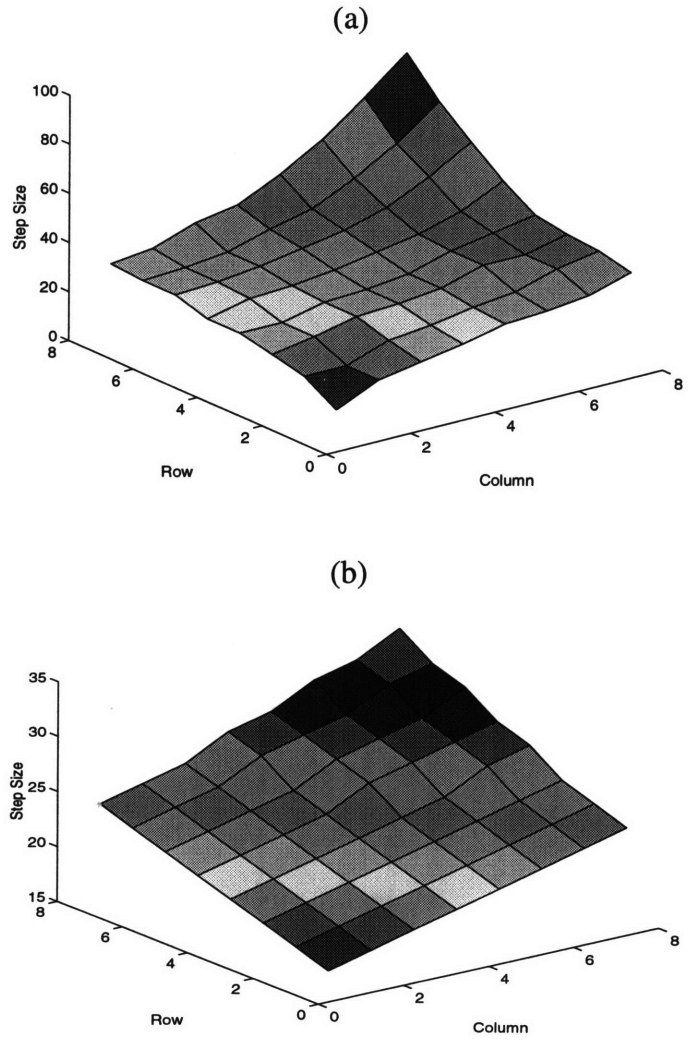
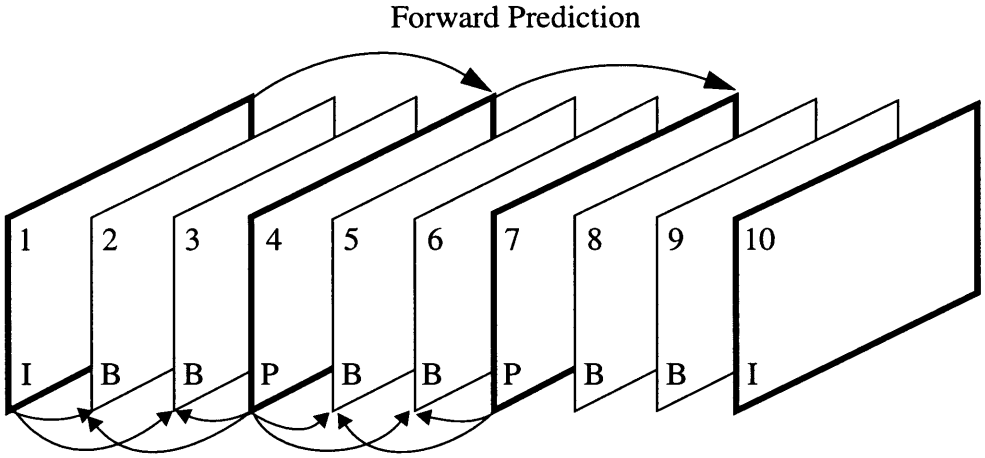


Figure 2.6: 3-D View of Default Quantizer Matrices: (a) Intra, (b) Inter.

forward predictive-coded pictures (P pictures) and bidirectionally predicted pictures (B pictures). Figure 2.7 shows an example of a typical GOP, which contains I picture 1 and 10, P pictures 4 and 7, and B pictures 2, 3, 5, and 6. The coding/transmission order and the displaying order are also shown. Note how B pictures 2 and 3 are coded after P picture 4, using both I picture 1 and P picture 4 as references. B pictures 8 and 9 are part of the next GOP because they are coded after I picture 10.



Display Order: 1 2 3 4 5 6 7 8 9 10 ...

Coding/Transmission Order: 1 4 2 3 7 5 6 10 8 9 ...
I P B B P B B I B B ...

Figure 2.7: A Typical Group of Pictures.

Random access is possible due to the availability of the I pictures. These pictures can be coded independently from the rest of the video sequence and can serve as starting point for further coding. Because MPEG1 does not specify any length for a GOP, the encoder designer has total control over how many random access points an encoded video sequence can have by setting how many I frames are to be coded.

2.1.2.8 Picture Header

Within the GOP picture data are the Picture Headers, which contain the type of the current picture, e.g. I, P or B, along with the temporal reference indicating the position of the picture relative to the display order. Included is a very important parameter called *vbv_delay*, which indicates how long to wait after a random access before starting to decode. This parameter is used to keep the decoding buffer from underflow or overflow.

2.1.2.9 Slice Header

The picture data of the Picture Layer contain the Slice Layer. Consecutive macroblocks of arbitrary length running in raster scan order make up a slice. Figure 2.8 shows an example of MPEG1 slice structuring. The Slice Header is often used for resynchronization from transmission bit errors. DC Intra coefficients and other prediction parameters are reset at the beginning of a slice. How large the slice should be usually depends on the expected bit error rate (BER) of the transmission environment. In general, smaller slice sizes are chosen for higher BER. The first and last MBs of a slice cannot be skipped MBs (not coded for), and no gaps are allowed between slices. The Slice Header also contains the vertical position of the slice within the picture, as well as a parameter called *quantization_scale*. This parameter is used to adjust the coarseness of the quantization of the slice.

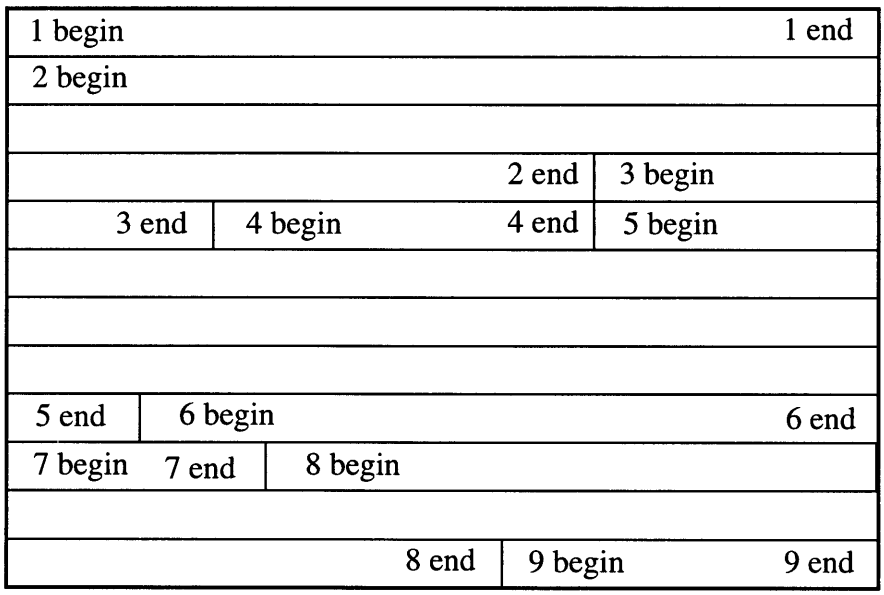


Figure 2.8: Possible Slice Arrangement for a 256x192 Picture.

2.1.2.9 Macroblock Header

As described before, a macroblock (MB) is a unit of 16x16 pixels used for motion com-

pensation. The Macroblock Layer is contained within the Slice Layer. Included in its header are the horizontal position (address) of the first MB for each slice which is coded with Variable Length Coding (VLC), MB type (Intra, NonIntra, etc.), *quantizer_scale*, motion vectors, and coded block pattern. The position for other transmitted MBs are coded differentially with respect to the most recently transmitted MB, also using VLC. Note that skipped MBs are not allowed in I pictures. In P pictures, skipped MBs are interpreted as NonIntra with zero coefficients and zero motion vectors. In B pictures, skipped MBs are assumed to be NonIntra with zero coefficients and the same motion vectors as the previous MB.

2.1.2.10 Block

A Block contains data for the quantized DCT coefficients of size 8x8 pixels in the macroblock. The blocks transmitted are coded using VLC; skipped blocks are assumed to have zero DCT coefficients.

2.1.3 A Typical MPEG1 Encoder

Figure 2.9 shows a typical MPEG1 video encoder[2]. It is assumed here that the reordering of the video pictures are done before they are fed into this encoder. In other words, I and/or P pictures used as references for B picture prediction must be coded and transmitted before processing of the corresponding B pictures. The reordered video input is fed into the Motion Compensation Estimator/Predictor which feeds a prediction to the minus input of the Subtracter. A decision is made by the Inter/Intra classifier for each MB as to whether or not to code it Intra. Usually, if the mean square prediction error exceeds the mean square pel value (indicating that the prediction isn't as accurate), then an Intra MB is coded.

2. Block diagrams of MPEG1 codec are taken from [12].

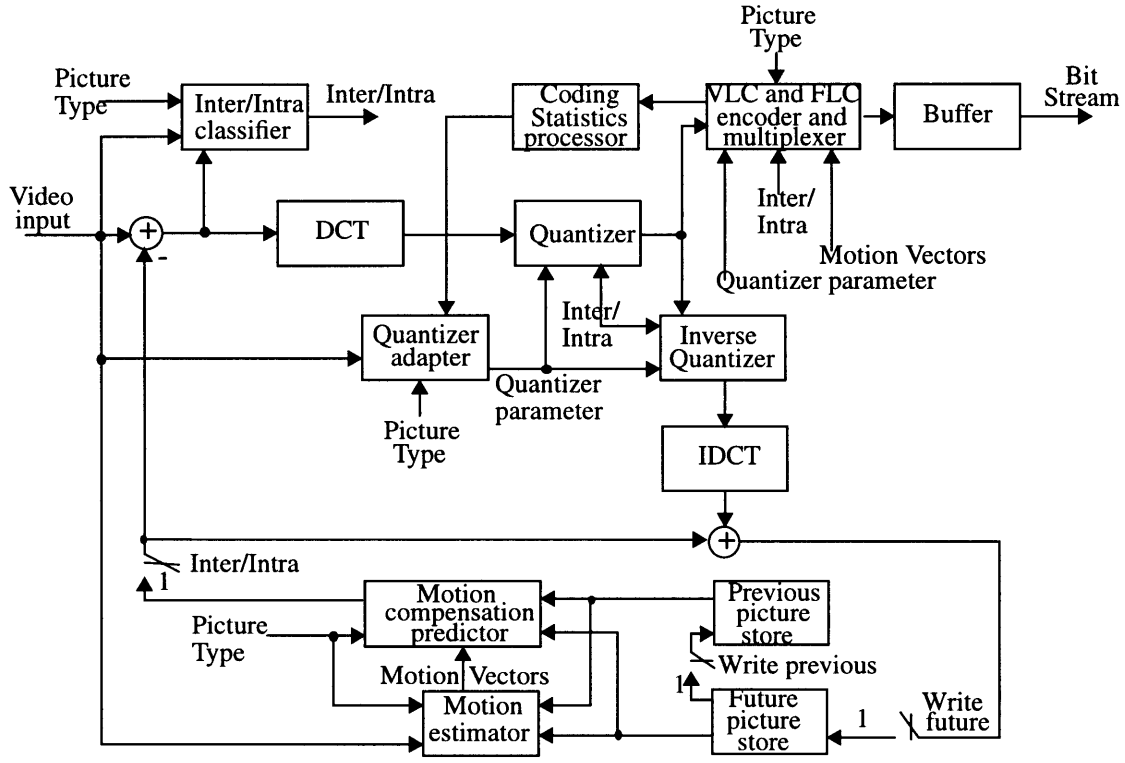


Figure 2.9: A Typical MPEG1 Encoder.

For Intra MBs, the prediction is set to be zero. Else, the prediction error is then passed through the DCT process and the Quantizer before being VLC coded, multiplexed and sent to the buffer. Reconstruction is performed by passing quantized levels to the Inverse Quantizer and Inverse DCT to produce a coded prediction error. The prediction is then added to the prediction error, and clipped to produce the desirable range of 0 to 255. Note that previous and future reference pictures are stored and used for Inter-coded pictures. They remain unchanged during B picture coding. For I and P pictures, the coded pels output are written to the Future Picture Store while the old pels are duplicated from the Future Picture Store to the Previous Picture Store.

Both the Coding Statistics Processor and the Quantizer Adapter control the output bit rate to conform to the Video Buffer Verifier (VBV) and help achieve the highest picture

quality possible. It is common to have a target number of bits or buffer fullness to be generated for each picture. The parameter *quantizer_scale* is adjusted periodically in attempt to meet this targeted value. More complex system can adjust this parameter for each MB.

2.1.4 A Typical MPEG1 Decoder

The decoder architecture is basically identical to the reconstruction portion of the encoder. Figure 2.10 shows a typical setup for such a decoder. As before, it is assumed that reordering of the encoded pictures is done before entering the decoder. Note that memory is required for such reordering to take place.

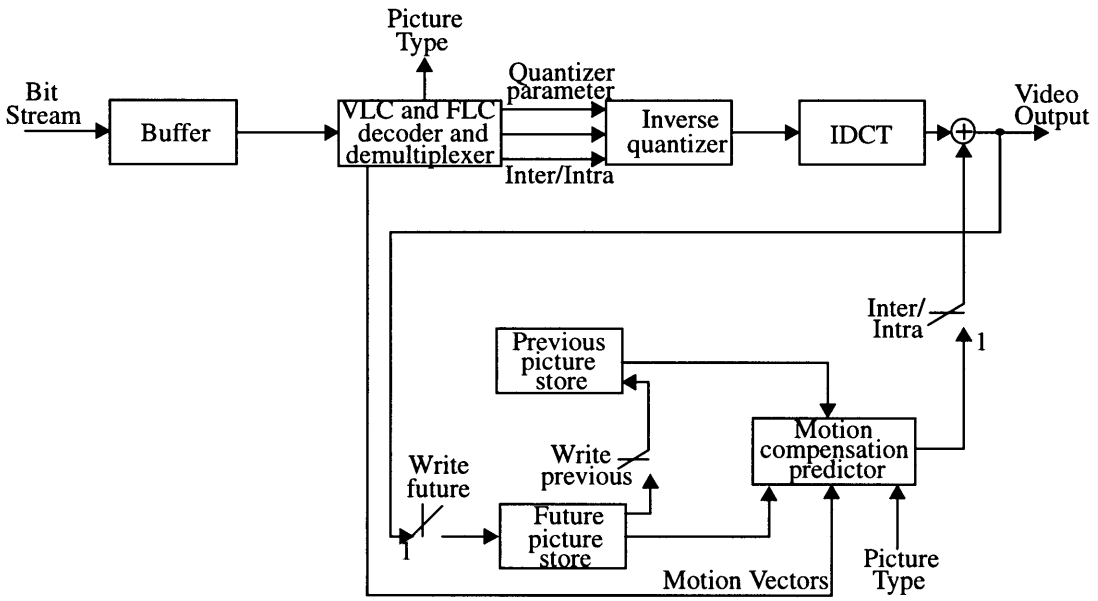


Figure 2.10: A Typical MPEG1 Decoder.

Decoders are often designed to handle transmission bit errors. Normally, when errors are detected, the decoder replaces the erroneous data with previous MBs until the next slice is received. However, this technique recovers the errors well only for intra pictures.

Many other strategies can be implemented for Intra pictures and other cases of errors. Further implementation of error recovery is developed for MPEG2, which will be discussed subsequently.

2.1.5 Overall Performance of MPEG1

With bit rate of 1.1 Mbits/sec and SIF resolution input pictures, many engineers feel that MPEG1 achieves image quality close to that of VHS videotape playback. Recall that higher bit rate and a much wider range of resolution are also supported by the syntax. Therefore, MPEG1 has provided a generic bit-stream syntax that can be beneficial to a wide variety of applications. The complete syntax and informative sections are provided in the standard [3].

2.2 The MPEG2 Standard

MPEG2 is designed to code video at bit-rates above 2 Mbits/sec; its official name is ISO/IEC 13818. Its original goal is to code interlaced CCIR 601 video at a bit rate that meets a wider variety of consumer applications. This is the key difference between MPEG1 and MPEG2--MPEG2 handles interlaced video more efficiently. Two picture structures are specified for this purpose--Field Pictures and Frame Pictures. For Field Pictures, each picture consists of fields that are coded independently. Frame Pictures, on the other hand, are composed of pairs of fields which are combined into frames before coding. Also, since the input video is CCIR 601, which has about 4 times the resolution as MPEG1's SIF, the optimal bit rate for MPEG2 is 4 Mbits/sec. Much higher bit-rates are also allowed.

In order for CCIR 601 color samples to be better compressed, a new video format is defined where 2:1 subsampling of both the vertical and horizontal directions are performed. This results is the MPEG2 4:2:0 video format, as shown in Figure 2.11. For the case of interlace video, the odd (first, third, fifth, etc.) rows of 4:2:0 chrominance CbCr

samples are temporally the same as the odd rows of the luminance Y samples. Similarly, the even rows of chrominance samples are temporally the same as the even rows of luminance samples. Other options are available, including supporting other video formats such as 4:2:2 and 4:4:4.

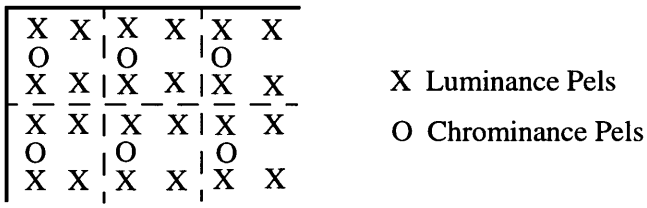


Figure 2.11: MPEG2 4:2:0 Color Format.

To meet the demand of many more applications, further options are provided in MPEG2. For example, communication over packet networks requiring prioritization so that low priority data can be dropped to relieve congestion is possible. Sending the same program to different decoders which handle different picture quality is also an option. However, not all applications will require all these options. Therefore, MPEG2 has designed several sets of constrained parameters listed through a two-dimensional ranking order. The first ranking is called *Profile*, where coding features are listed. The other dimension is called *Level*, where information such as picture resolutions, bit-rates, etc. are provided. The most important combination is the Main Profile at Main Level (MP@ML), of which parameters are listed in Table 2.3. For other profiles and levels, please refer to [4]. Documentations regarding system integration and audio compression standards can be found in [5] and [6] respectively.

Table 2.3: Upper Bounds for MPEG2 MP@ML Parameters.

Parameter	Upper Bound
horizontal size	720 pixels
vertical size	576 lines
frame rate	30 Hz
pel rate	10.368 MHz
VBV Buffer	224 Kbytes
bit rate	15.0 Mbits/s
color subsampling	MPEG2 4:2:0

2.2.1 The MPEG2 Bit-Stream Syntax

The syntax of MPEG2 MP@ML bitstream is a superset of the syntax of MPEG1. There are many additional extensions adjacent to the Headers. The use of VBV to maintain constant bit-rates is kept from MPEG1. Yet, unlike MPEG1, picture skipping is allowed. For the Video Sequence Header, information about which Profile/Level and Interlace/Progressive Display is also given in addition to other parameters listed in the MPEG1 Video Sequence Header. For GOP Header, more information is given regarding open or closed GOP. The Picture Header contains additional parameters such as the 3:2 pulldown flag, alternative scan to the DCT zigzag scan, and presence of error concealment. The Slice Header also allows for indication of slices containing only Intra MBs, which is used for the purpose of quick search operation. Unlike MPEG1, each horizontal row of MBs must begin with a new slice, and the first and last MBs of a slice may not be skipped. There are many more additional MB types available to be included in the Macroblock Header. These result from the new coding options of interlaced video.

2.2.2 Exploiting Spatial Redundancy

Similar to MPEG1, block-based two-dimensional DCT is employed by the MPEG2 algorithm. Quantization is applied to each DCT coefficient and further compression is achieved by transmitting the nonzero quantized coefficients and their entropy-coded locations and amplitudes.

To better process interlaced video, a new "zigzag" scanning order is employed to replace the one used for progressive video. This new scanning order, called the Alternate Scan, is shown in Figure 2.12, and may be specified in the Picture header. In addition to the new scanning order, a capability for field coding within a MB is also possible. In other words, the encoder may reorder the luminance lines within a MB so that the first 8 lines are from the top field and the last 8 lines are from the bottom field. Chrominance reordering is not done. The overall effect of this reordering is to increase the vertical correlation within the luminance blocks and thus decrease the DCT coefficient values.

1	5	7	21	23	37	39	53
2	6	8	22	24	38	40	54
3	9	20	25	35	41	51	55
4	10	19	26	36	42	52	56
11	18	27	31	43	47	57	61
12	17	28	32	44	48	58	62
13	16	29	33	45	49	59	63
14	15	30	34	46	50	60	64

Figure 2.12: MPEG2 Alternate Scan.

2.2.3 Exploiting Temporal Redundancy

In addition to MPEG1's temporal redundancy reduction methods of interframe, MPEG2

also uses other methods to deal with interlaced video. These are listed below. Experiments done on these prediction modes can be found in Appendix L of [1].

2.2.3.1 Frame Prediction for Frame Pictures

This is the same prediction methods that MPEG1 uses for reducing temporal redundancy. Motion compensated MBs and the use of P and B pictures is also kept. MPEG2 classifies these techniques as Frame Prediction for Frame Pictures and as in MPEG1, allows one motion vector for each forward predicted target MB and two motion vectors for each bidirectionally predicted target MB. As before, predictions are made from reference pictures which can be either P or I pictures. Overall, this prediction method is well suited to code video with slow or moderate motion and detailed background, and is not used for field pictures.

2.2.3.2 Field Prediction for Field Pictures

Another mode of prediction that MPEG2 employs is the Field Prediction for Field Pictures. This method is similar to Frame Prediction but the target MB pels all come from the same field, but is not constrained to polarity (top or bottom field) as the target MB field. For P pictures, the prediction MBs can come from either of the two most recently coded I or P fields. For B pictures, the prediction MBs can come from the two most recently coded I or P frames. As before, one motion vector is assigned to forward predicted target MBs and two motion vectors may be used for bidirectionally predicted target MBs. This mode is specifically designed for field pictures and is not used for frame pictures.

2.2.3.3 Field Prediction for Frame Pictures

A third mode of prediction that MPEG2 uses to handle rapid motion interlaced video is called Field Prediction for Frame Pictures. Using this mode, the target MB is first divided into two fields--top field pels and bottom field pels. Field prediction is then carried out as before, independently from each of the two parts. Hence, up to two motion vectors can

result from a forward predicted target MB and up to four motion vectors can result from bidirectionally predicted target MB.

2.2.3.4 Dual Prime for P Pictures

This fourth mode of Dual Prime prediction can only be used with P pictures where one motion vector per MB is assigned; the reference pictures cannot be B pictures. Two predictions are initially made and then averaged together to form the resulting prediction. The first initial prediction is a field prediction as before, except that the prediction pels all come from the previously coded fields having the same polarity as that of the target pels. Prediction pels come from one field for field pictures and from two fields for frame pictures. The second initial prediction is computed by using a motion vector plus a motion vector correction. For this second case, the prediction pels come from the opposite polarity field. The computed motion vectors are obtained through temporal scaling of the transmitted motion vector to match the field where the prediction pels are located. The correction can be up to half a pel and is included in the MB Header.

This prediction mode provides image quality comparable to that of using B pictures. Its gains the advantage of lower coding delay since no frame reordering is required as in B pictures. However, higher memory bandwidth is usually required for this prediction mode.

2.2.3.5 16x8 MC for Field Pictures

The fifth mode of prediction, called 16x8 MC for Field Pictures, splits the Field Picture MB into two halves. A field prediction is applied to each half separately, resulting up to two motion vectors for P picture MB and up to four motion vectors per B picture MB.

2.2.4 Typical Codec Architecture

The structure for the MPEG2 codec is similar to the MPEG1 codec illustrated in Figure

2.9 and Figure 2.10. The key difference is the additional complexity in the Motion Estimation/Prediction and the Coding Statistics Processor, where many new options are available.

2.2.5 MPEG2 Scalability Extensions

Several extensions has been made to the Main Profile to increase MPEG2 functionality, particularly towards error concealment. The following Scalability Extensions provide two or more separate bitstreams called Layers for the same video input, which can be combined to provide a high quality video signal. The *Base* Layer bitstream can be decoded by itself and can provide a lower quality video signal. With the addition of the *Enhancement* Layer, the overall image quality can be increased significantly. These scalability extensions are particularly useful for packet transmission systems, such as Asynchronous Transfer Mode (ATM) networks, which allow for dual priority data packetization. The Base Layer is sent over the high priority channel with little errors and the Enhancement Layer is sent over the low priority channel where errors are more likely to occur. In the case of low priority packet loss, a reasonable quality video signal can still be recovered. Each of these extensions is further discussed below.

2.2.5.1 SNR Scalability

SNR scalable coding provides two layers with the same spatial resolution but different image quality. This is done by allowing the enhancement layer to have corrective information regarding the accuracy of the DCT coefficients. These corrective values are added to the base layer prior to its IDCT operation. Besides concealing errors when the enhancement layer is lost, SNR scalability can also be used for video distribution networks where the base layer is used to broadcast nonedited video and the fully enhanced video is used for editing.

2.2.5.2 Data Partitioning

Data partitioning is a simplified version of SNR scalability where the base layer contains the low frequency DCT coefficients while the enhancement layer contains the remaining high frequency ones. The key feature of Data Partitioning is its simplicity and low overhead required, relative to other scalable coding. The only extra information needed is the codeword counter to control the switching from one layer to another. Because of its low overhead cost, the fully enhanced image quality is very close to that of the single layer coding.

2.2.5.3 Spatial Scalability

The base layer of spatial scalable coding has lower spatial resolution than the original video. Full original resolution is obtained after the enhancement layer is added. Also, the lower layer is coded without regard to the enhancement layer. When the enhancement layer is lost, the upconverted lower layer can be used directly. This lower layer data is relatively free of nonlinear distortions like blocking effects and motion jerkiness.

2.2.5.4 Temporal Scalability

Temporal Scalability allows layering of video frames with the same spatial resolution as the original video but with lower temporal rates (frame rates); the combined video has full temporal resolution. When the enhancement layer data is lost, the lower layer can still maintain full spatial quality and resolution, but at half the temporal rate.

2.3 Regarding MPEG and Low Delay Coding

As previously suggested in this chapter, MPEG standard coding using I, P and B pictures is not well suited for applications requiring low delay coding. This is particularly true for channels with constrained constant bit rates. Because I pictures are coded by spatial redundancy reduction technique which does not take advantage of temporal redundancy,

they require many more bits than P and B pictures to achieve good image quality. High amount of generated bits requires larger encoder compressed data buffers which directly translates to longer delay than by using smaller buffers.

B pictures bring about significant delay due to the necessary frame reordering. For encoding, reference pictures (I or P) required for coding these B pictures must be coded and transmitted before the corresponding B pictures are processed. On the decoding end, the reference pictures must be decoded before the corresponding B pictures are decoded. Therefore, the key to low delay coding is the elimination of I and B pictures. Coding with only P pictures provides the necessary end-to-end delay that meets the demand of most applications such as videoconferencing.

Two major obstacles are encountered for coding with only P pictures--risking the propagation of channel errors, and facing encoder/decoder incompatibilities in the implementation of the IDCT algorithm. Channel errors can often be introduced into the compressed data bitstream by random noise, ATM packet loss, etc., depending on the communication channel and the transport system. Because coding of a P picture must use the previous P picture as a reference, any MB having errors will most likely propagate. If a prediction is "lucky," it will not use the region containing the erroneous MB for prediction, but that will not likely be the case. It is more possible that an error can be passed on throughout the entire video sequence if the error is not somehow corrected.

The second obstacle of encoder/decoder incompatibilities occurs because the reconstructive IDCT operation of the MPEG encoders is often not identical to that of a decoder. In other words, the encoder's IDCT operation often yields very slightly different coefficient values than those generated by a decoder's IDCT operation, purely due to different codec's arithmetic accuracy. This incompatibility often arises when the encoder and decoder are not designed by the same person.

The solution to both problems discussed above is to periodically refresh all macroblocks in the video pictures by coding them intra. When an erroneous MB is replaced by an Intra MB of which no reference has been used to code it, the error is eliminated. Also, meeting the requirement of refreshing the entire picture using intra MBs every 132 frames will meet MPEG2 specification for eliminating encoder/decoder incompatibilities. Another possible solution is to use leaky prediction where the predicted frame is added by a small factor of the original input image, thereby updating the image gradually. We concentrate on the first solution and leave the second solution to be explored in future research.

Recall that coding an entire frame Intra would contradict the decision to eliminate I pictures for low delay coding. Thus, only a small number of macroblocks can be refreshed per picture. The MPEG2 standard vaguely mentions using intra slices as a macroblock refreshing method (Appendix D in [4]). However, exactly how this is done is not specified in the standard. Other methods are also possible and will be discussed in the following chapter. Additional specific problems arise for low delay coding. These will also be addressed in the following chapter where the design and implementation of the low delay coding algorithm are given in details.

3 Low Delay Coding Design and Implementation

In this chapter, details about the design and implementation of the low delay coding algorithm are given. But first, works on H.261 and H.263, two videoconferencing low delay coding standards, will be described briefly and compared with the MPEG2 low delay coding algorithm. The complete documentation for H.261 and H.263 standards can be found in [7] and [8], respectively. Also, a more thorough comparison between H.261 and H.263 can be found in [10].

3.1 The H.261 and H.263 Videoconferencing Standards

H.261 is a well known and widely used videoconferencing standard in today's industry, particularly in Europe. Recently, it has been superseded by the improved H.263. Both of these coding schemes utilize similar spatial and temporal redundancy reduction techniques as MPEG1 and MPEG2, namely, the DCT algorithm and interframe prediction. They both also use quantization, zigzag scanning, and VLC to further improve the compression ratio. Further works with these standards such as multipoint videoconferencing and video bridging are included in [20] and [21].

To achieve low lag, codings using H.261 can choose various low bit rates with multiples of 64 Kbits/sec. The multiples can range from 1 to 30, giving H.261 another name, p*64. Besides these extremely low bit rates, H.261 constrains the luminance resolution to be in CIF (Common Intermediate Format) which has a spatial resolution of 352x288 pixels. QCIF (Quarter CIF) is also supported where the resolution is 176x144 pixels. Because the video buffer is extremely limited, H.261 also allows up to 3 frames to be skipped between two coded frames. Frame rate is also dropped down to 10-15 frames/sec. For better contrast, a comparison between MPEG1 and H.261 can be found in [9].

H.263 aims at bit rates below 64 Kbits/sec but achieves similar quality as H.261. In addition to supporting CIF and QCIF picture formats, H.263 supports a third format called sub-QCIF where the luminance resolution is merely 128x96 pixels, especially for extremely low bit rates, e.g., below 10 Kbits/sec. Other major changes made are unrestricted motion vector mode, advanced prediction mode (e.g., four motion vectors per MB, overlapped motion compensation), and PB-frames mode.

These standards are designed particularly for low delay videoconferencing applications. Image resolution and quality are often severely sacrificed to achieve extremely low bit rate and low delay, suitable for low bandwidth transmission channels. On the other hand, MPEG2 low delay coding aims towards higher bit rates for much higher image quality, thereby requiring higher bandwidth transmission channels. At very low bit rates, H.261 and H.263 are the clear winners in terms of image quality because MPEG has high overhead; however, remarkable improvement is achieved using MPEG at higher bit rates. The MPEG2 technology also provides much more freedom with input video formats and other fine-tuning options to support a wider range of high-end applications. With the advent of many high bandwidth transmission channels, MPEG2 and low delay coding will represent a superset of all existing general video and videoconferencing standards.

3.2 Design of the Low Delay Coding Algorithm

The key design of the MPEG2 low delay coding algorithm is described in this section. Several major issues such as error propagation and compliance to the MPEG2 standard are also covered.

3.2.1 Refreshing Strategy Using Intra Slices and Columns

To keep low encoder/decoder buffer requirement and to avoid frame reordering delay, the very first frame of our low delay coding algorithm is an I frame but the remaining of the

video sequence is composed of only P pictures. Together, the first I frame and all future P pictures fulfill the MPEG2 bitstream syntax. Although the low buffer requirement will result in relatively poor picture quality for the very first I frame, being the only I frame ever coded in the entire video sequence allows such distortion to quickly disappear. In our design, we insert intra slices or columns into each P frame in a sequential order as shown in Figure 3.1. Intra slices are inserted in a top-to-bottom fashion while intra columns traverse from left to right. The size of an intra slice (or intra column) may be selected to be from one to several rows (or columns) of macroblocks per frame. Notice that the entire screen is eventually refreshed in what we will refer to as a "refresh cycle" (3 frames in each of the examples of Figure 3.1). This refresh cycle consists of all frames starting from the one with the upper most intra slice (or left most intra column) to the frame with the lower most intra slice (or right most intra column). This refresh cycle resembles a group

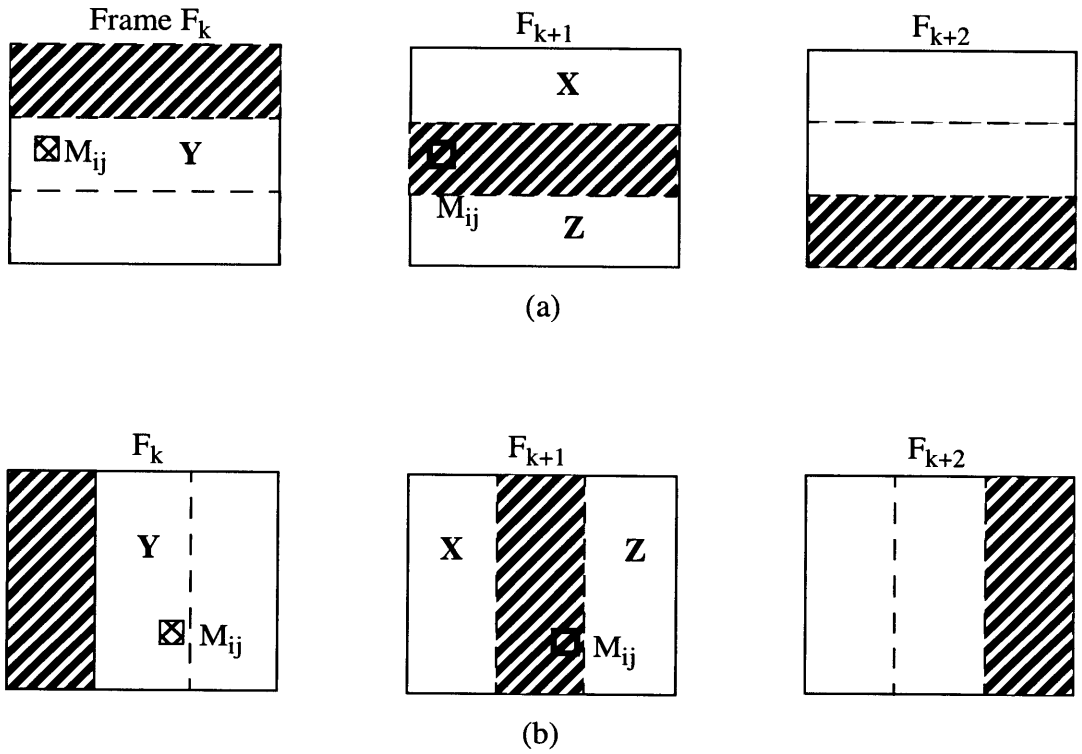


Figure 3.1: Low Delay Coding Algorithm Using (a) Intra Slices, and (b) Intra Columns.

of pictures (GOP), a standard MPEG structure where a series of pictures is typically headed by a single Intra frame. Please note that since a column is not a specified MPEG2 bitstream structure, it should be regarded as a series of MBs aligned in the same column.

How many frames should a refresh cycle have is the decision that the user must decide. As designed, the algorithm gives the user the ability to select the size of the intra region to be any integral number of rows or columns of macroblocks; but how large is typical? In a typical standard IPB MPEG2 coding, an Intra frame is inserted every 15 frames to refresh all the macroblocks at once. To attempt simulating such refreshing rate for low delay coding, a user may specify the size of the intra slice for a 720x480 pixel CCIR 601 resolution video sequence to be 2 rows of macroblocks. This number is easily derived from the 30 rows of macroblocks that this sequence has. If 2 rows of macroblocks are sequentially refreshed every frame, then after 15 frames, the entire picture is refreshed. Likewise, 3 columns of macroblocks to be coded intra may be specified to obtain the same refreshing rate if intra columns are used for updating. Because there are 45 columns of macroblocks in this video sequence, the entire picture is refreshed after 15 frames where each frame has 3 sequential intra columns. In short, how large the intra region is directly controls the refreshing rate (or cycle)--smaller intra region for longer refreshing interval and larger intra region for shorter refreshing interval. In addition, note that this design is only for frame prediction mode instead of field prediction mode. Recall that frame prediction is better suited for low motion video, which is typical of low delay applications such as videoconferencing.

An alert reader should notice that there is a slight problem with the notation of intra "slices." As mentioned in the previous chapter, MPEG2 requires that each horizontal row of MBs must start with a new slice. Therefore, it is noncomplying to call an intra slice of size larger than one row of MBs as an intra "slice." So an intra region which occupies, say,

two rows of MBs, actually composes of two intra slices, each having the size of one row of MBs. However, for the sake of length and distinguishing between an intra slice and an intra column, "intra slice of size 2" will continue to be used to notate such an intra region. Similarly, "intra column of size 2" will be used to notate an intra region composing of two columns of MBs. Because a column of MBs is not a specified structure of the MPEG2 bitstream, notating an intra region composing of 2 columns of MBs as an "intra column of size 2" not a noncompliance as long as the coding of it does not violate the syntax.

In terms of low lag time, this algorithm in conjunction with the MPEG2 software codec allows the user to specify the desirable maximum end-to-end delay time, not including any processing or channel delay. This delay is associated with the parameter vbv_delay (Video Buffer Verifier Delay) mentioned in Chapter 2 and it is determined by

$$Delay = \frac{BufferSizeLimit}{BitRate}, \tag{3.1}$$

where *BufferSizeLimit* is the user-specified constraint on the encoder's maximum buffer size and *BitRate* is the constant bit rate that the input video will be coded at, which is also user specified. This parameter is calculated from the rate control process and is sent to the decoder end within the Picture Header of each frame. Figure 3.2 illustrates this process. The decoder then has to wait for that calculated time before it starts to decode the compressed bitstream since the encoder would have finished transferred one entire frame by that time. Normally, the decoder utilizes this parameter whenever there is a random access but typically, the so-called random access occurs only once at the very beginning. Nevertheless, this design does not limit the decoder from performing subsequent random accesses by allowing the encoder to send a sequence header every N frames. N can be

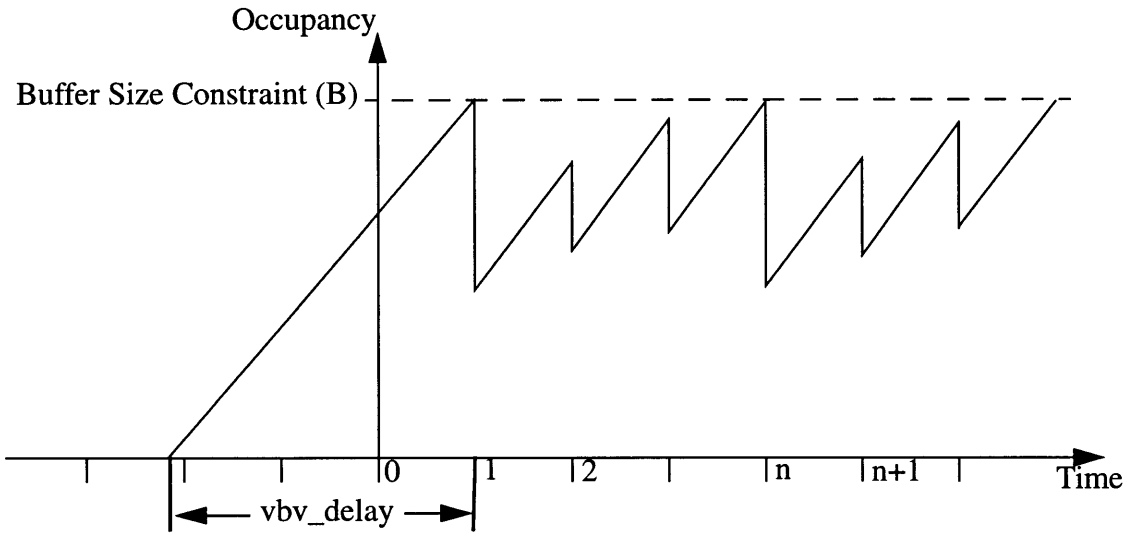


Figure 3.2: VBV Buffer Occupancy with Constant Bit Rate.

adjusted to accommodate the user's specific application.

Figure 3.2 describes a typical behavior of the VBV buffer fullness. VBV (Video Buffer Verifier) is a hypothetical decoder located within the encoder in an attempt to mimic what an actual decoder might do. Starting from an empty buffer, compressed video frames are consistently filled at the specified bit rate and instantaneously removed from the video buffer. The exact level depends on how many bits are generated for each picture, and when they are removed. This is controlled by the rate control process which ensures that the level will never become zero (underflow) or exceed (overflow) the given buffer size constraint B. Two common techniques used for such control are bit stuffing (to prevent underflow) and MB skipping (to prevent overflow). As shown, once the first frame has filled and is instantaneously removed from the buffer even though the second frame is being filled at the same time, the decoder can start decoding. The time it takes to fill the VBV buffer and then remove that first frame is the vbv_delay. Because each picture is filled with the chosen constant bit rate, it is simple to deduce that the maximum vbv_delay is simply the quotient of the buffer size constraint and that bit rate. Once the decoder has

started, it simply waits for 1/30th of a second before decoding the next frame, given that 30 frames/sec is the encoded frame rate and that the displaying time isn't varying. Note that the buffer occupancy level does not ever have to reach the maximum buffer size as shown in Figure 3.2 (so the delay time can actually be shorter than the maximum time specified). The occupancy level may be steady below the maximum level without sacrificing image quality as long as data bits are always available for any coded picture.

3.2.2 Reducing Error Propagation

If a channel error occurs in macroblock M_{ij} of frame F_k, then an error-free intra slice (or column) containing macroblock M_{ij} in frame F_{k+1} will eliminate that error immediately. Such a case where the error is quickly removed will not always happen. Most of the time, it will take on average F/2 frames before the error is removed, given that F is the number of frames in a refresh cycle. Prediction of where these errors are located cannot be made due to their random nature. Some ideas about MB or slice updating pseudo-randomly have originated from this random behavior of bit errors.

One key advantage of the intra frame updating (IP coding) is to refresh all the macroblocks during a single picture, instead of having to go through a complete refresh cycle before every MB location is refreshed. Channel errors may only propagate through F frames where F is the maximum number of frames in the GOP; once an error-free intra frame is decoded, the errors are immediately removed. However, as pointed out before, the codec's small buffer requirement do not allow for all macroblocks in a single picture to be updated smoothly all at once; intra coding requires many more bits to acquire equivalent image quality compared to inter coding. Its trade-off is its ability to remove errors quickly.

Some errors are also capable of propagating through the video sequence despite the usage of intra slices and intra columns [2]. For Figure 3.1, when a recently refreshed macroblock in region X uses an erred region in Y for motion compensation, error propagates. This happens whenever objects in the video picture are moving opposite to the direction of the refresh. As stated in the MPEG2 standard, it would require up to 2F frames to remove such an error, where F is the total number of frames in a refresh cycle. To avoid this problem, we also tested low delay encoding with constrained motion estimation. This strategy constrains the motion estimation vectors of macroblocks already refreshed in the current refresh cycle to only point to pixels already refreshed in the same cycle. Macroblocks not yet refreshed have no restrictions. For instance, region X may not use region Y for motion estimation for frame F_{k+1} in Figure 3.1, but region Z can be motion compensated from anywhere in the previous picture. Such constraint may cause higher prediction errors but is a trade-off for error propagation reduction. Going back to the idea of updating MBs or slices pseudo-randomly, it is clear that errors are more likely to propagate uncontrollably (i.e., much longer than 2F frames) with such technique. Trying to apply constrained motion estimation to such random updating strategy will result in even higher prediction errors.

Without regard to low coding delay, note how IPB standard coding has a major advantage in terms of error propagation reduction. Any error is removed immediately upon encountering an error-free I frame. Also, if the error occurs on a B frame, it will disappear the very next frame (i.e., in 1/30th of a second for 30 frames/sec mode) because that B frame is never used as a reference picture for future prediction. However, an argument can be made that the chance an entire I frame is error-free is lower than that of an intra slice or column since an I frame takes up many more bits in a bitstream than an intra slice or column.

3.2.3 Compliance With MPEG2 Standard

Because the IDCT operation is usually not identically implemented on any two encoder and decoder, there is a slight mismatch among the coefficients that the operation produces. These mismatch errors result from arithmetic imprecision of the IDCT. The errors are usually small, but they accumulate over time and if left unfixed, they can cause noticeable artifacts. Therefore, macroblock refreshing/updating operation is necessary to eliminate such mismatch between the IDCT reconstruction process of an encoder and that of a decoder.

As stated in Appendix A of the MPEG2 standard (and H.261), every macroblock in a picture is required to be refreshed before it is coded 132 times as predictive macroblocks. Skipped macroblocks are not included because they do not contribute to mismatch errors. Also in order for the number 132 to be valid, the standard also specified that the precision of the IDCT should be sufficient so that errors do not occur in the final integer values. With the design of our low delay coding algorithm, both of these requirements are easily met. Because all other aspects of the design do not violate any specified requirement of the MPEG2 standard, the bitstream generated from the low delay coding algorithm is decodable by any standard MPEG2 decoder.

3.3 Implementation of the Low Delay Coding Algorithm

The design of the low delay coding algorithm is implemented on an MPEG2 simulating software encoder written in the C programming language. The existing standard MPEG2 simulating software encoder, provided by the department of Video and Image Technologies at the IBM T. J. Watson Research Center at Yorktown Heights, New York, is modified to add the new options of low delay coding using intra slices and/or intra columns. The encoder was originally implemented by Elliot Linzer et al. at IBM Research and later

modified by Peter Westerink, also from IBM Research. In the following subsections, relevant files of the encoder and some major highlights of the new implementation will be discussed, including the implementation of intra slices/columns and constrained motion estimation. The source codes for the entire MPEG2 simulating software encoder are not included in this thesis due to its extremely large size relative to this document and also somewhat due to its irrelevancy to this project. The low delay coding algorithm is designed to be implemented on any encoder, and is not an ad hoc option to this particular encoder. However, more complete details about the major and relevant low delay coding functions can be found in Appendix A. Also note that compressed bitstreams are decoded by an MPEG2 simulating software decoder, also written in C. Because of its standard features and also because such software decoder can easily be found on the Web (for example, at Web site *http://sun1.bham.ac.uk/eee5nlm6/mpegt.html*), the source codes for it are also not included in this document.

It is also important to note that the software encoder/decoder mentioned above do not provide real-time MPEG2 coding/decoding. Depending on the speed of the computer that the codec is running on, it is capable of coding at about 3 pictures of a CCIR 601 video sequence per minute and about twice as fast if precomputed motion vectors are used. The decoder can decode and display at roughly 15 pictures per minute. If real time video displaying is desired, the decompressed file is transferred to a much faster mainframe and displayed there. The software codec to be discussed is mainly used for research purposes and because real-time MPEG2 codecs require very high computing power, they are often found in hardware instead. Many MPEG2 hardware codec systems exist today, such as the IBM MPEG2 codec. More information about such hardware system can be found at the Web site: *http://www.chips.ibm.com/products/mpeg/.*

3.3.1 Relevant Files of the MPEG2 Encoder

Several key files of the existing MPEG2 software encoder are modified to incorporate the new option of low delay coding and its complimentary feature of constrained motion estimation. Specific modifications are discussed in the subsections below. For the purpose of relevancy, minor modifications, such as specific declaration of the new functions and their parameters in *.h* files and other syntactic techniques employed to keep the program manageable and executable, will not be discussed. The key files of the encoder where changes are made are itemized below along with the description of the major functions that they carry out.

- ***dvdenc.c:*** This file contains the main function which controls the entire encoding process. Here, function calls made to initialize various parameters are located. Several memory buffer allocations are also made here. After obtaining user's parameters input, the program enters a large loop where the input files are opened and pictures are coded as described in Chapter 2.

- ***printparams_dvdenc.c:*** When called, this file prints out all the initialized parameter settings of the program *dvdenc*. Some of these parameters are source file name, picture resolution, chrominance format, first frame number to be coded, total number of frames to be coded, which rate control type, GOP structure, VBV buffer size, frame rate, and other options which the user has selected.

- ***printusage_dvdenc.c:*** This file is called whenever incorrect input format is given. It prints out on-screen the encoder's usage and its available options, a brief note about their functions and also their usage syntax. For example, if a user types the following at the prompt:

```
% dvdenc
```

then he will get the following message:

```
MPEG-2 Encoder

Usage:  dvdenc codnam orgnam [options]

Arguments (no defaults):
     codnam = base name of result coded bitstream
     orgnam = base name of original sequence for
input

Options:

...
```

A rather long list of available options is then printed. An example of an option is:

```
-bufsiz <i> = VBV buffer size in units of 16384
(default: 112)
```

- ***parscomlin_dvdenc.c:*** This function is always called at the beginning of each coding to read in user's input parameters. That is, the command line is "parsed," and the newly chosen parameters are initialized as global variables. This function also allows for a continuous command line. A typical input command might look like the following:

```
% dvdenc resulting_bitstream_filename
input_filename -bufsiz 36 -bitrat 6000000
```

where dvdenc is the command to execute the encoder. The first mandatory argument is the file name for the resulting compressed bitstream. Also mandatory, the second argument gives the file name for the input video sequence. The following two options, headed by a "-" and their names, indicate the user's choice of VBV buffer size of 589824 bits (36x16384 bits) and the constant bit rate of 6 Mbits/sec. All these information will be initialized and *printparams_dvdenc* will print them instead of other default values.

- ***enc_code.c:*** This file contains several major functions such as CodePicture and

CodeFrame. These functions are the masterminds for the encoder; they control the formation of the bitstream structure. Some primary operations include buffer control statistics gathering, controlling for DCT and quantization processes, and encoding of each MB.

- ***enc_mec.c:*** All computations needed for the motion estimation/compensation process are contained in this file. Some of these include the macroblock matching process, ensuring that the search range stays within the picture frame, and calculating the prediction error and motion vectors. All types of prediction that MPEG2 employs are included in this file. Another important function provided is `SelectMBMode()` where the type (Intra or Inter) of each MB to be coded is determined. Recall that a MB should be coded Intra instead of Inter if the mean square prediction error exceeds the mean square pel value.

- ***enc_rc.c:*** Rate control is the primary operation that this file provides. Buffer fullness is calculated and kept track; the functions in this file ensure that the number of bits generated are within bound of the VBV buffer size. This rate control process is done by first setting a target number of bits to code the next picture and adjust the quantizer scale to match such target. If more bits need to be generated to keep the VBV buffer from underflowing, the quantizer should use smaller step size to achieve finer quantization. Accordingly, if less bits are required to keep the buffer from overflowing, coarser quantization is performed. From the user's command line, the encoder knows ahead of time the GOP of the video bitstream so the targeting bits to be generated are optimal.

The rate control algorithm used for the MPEG2 simulating software (and also most of the MPEG2 encoder) was implemented by Elliot Linzer at IBM T.J. Watson

Research Center. A more thorough documentation on this rate control process can be found in [22]. Yet, other rate/buffer control algorithms exist such as the ones in [23] and [24], but they mainly revolve around adjusting and fine-tuning the quantization process. Normally, the quantization matrix is multiplied by a quantization scale. This quantization scale is increased (decreased) to reduce (increase) the number of bits produced.

- ***dvdmotest.c:*** This file contains the main function which is executed to compute only motion vectors. In other words, when this program is executed, the video sequence is not coded but rather the motion vectors that could have been used for the coding are generated. Using these precomputed motion vectors, running the MPEG2 encoder *dvdenc* will take about half the time compared to not using any. Recall that the motion estimation/compensation is the most computationally intensive operation throughout the coding process. These precomputed motion vectors are then stored away and can be used for future codings wherever the motion estimation/compensation process is not altered.

3.3.2 Implementation of Intra Slices and Columns

The implementation of the low delay coding algorithm is simplified thanks to the MPEG2 simulating software encoder provided. Not only can this encoder perform standard IPB coding at constant bit rate, it also supports many other options. Among these options are IP coding, controlling a limited VBV buffer size, performing scene change detection, and many others. However, low delay coding using intra slice or intra column updating and constrained motion estimation did not exist. The author was lucky to not have to implement an entire MPEG2 encoder. Not only that such work will be extremely challenging, it is an extremely large task and could not have been completed in a few months.

Declarations and passing of key variables, high level calls of various computational tasks, and the main processing loops are studied before a plan of implementation is devised. Because all the detailed computational functions have already been implemented, there is no need to implement functions such as how to code a MB intra or inter. That also means that not every single line of code in the encoder must be understood. Rather, it is a programming practice to treat many computational functions as "black boxes" and as long as the output of a function is understood, there is no need to understand exactly how it was done. Also using the method of abstraction, it is better to implement functions which will control those computational tasks. Listed below are the major files described above along with the key changes made to implement the low delay coding algorithm. These changes vary from entirely new functions to a few important lines of codes, but only a simplified version of these changes are included here. More details about these modifications can be found in Appendix A.

- ***dvdenc.c:*** All new global variables (and any default values) are declared in this main processing file. Also, a new file is allocated to record the encoder's buffer fullness level for analytical purposes.

- ***printusage_dvdenc.c:*** Four new options are printed. They are:

  -lowdelay <i>: When specified, this option tells the encoder to code using intra slice as the refreshing tool. "<i>" reminds the user to enter the desirable size for the intra slice; for example, replacing "<i>" with "2" signifies the encoder that coding using intra slice updating of size 2 rows of MBs is selected.

  -lowdelay2 <i>: This option allows the encoder to select intra column updating instead of intra slice. Similar to intra slice, "<i>" should be replaced with an integer

indicating the desirable size of the intra column. For example, when "3" is entered, 3 columns of MBs will be used as the size of the intra column.

-skpopt <i>: Not every frame must have an intra slice or intra column. This option allows the encoder to not code an intra slice or column for each frame. For example, if "1" is entered, then an intra slice (or column) will be inserted every other frame since 1 frame is skipped. The user must ensure that the number of frames being skipped from intra coding and the resulting refresh cycle may not violate MPEG2's refreshing requirement of 132 frames.

-constrained_me <i>: This option, which may only be used in conjunction with intra slice/column updating, signals the encoder to perform constrained motion estimation. An integer must be entered indicating the range of the constrained search range in multiples of 16 pixels. For instance, entering "2" will force the encoder to perform motion estimation with limited search range of 32 pixels from the target MB. Obviously, the range may be less than 32 pixels to ensure that it does not extend too far into the recently refreshed region, or more than 32 to reduce prediction errors.

- ***parscomlin_dvdenc.c:*** New codes are added to this file to allow the encoder to recognize the new options when they are selected. When the new options are selected, the new global variables indicating low delay coding are set to the specified values.

- ***printparams_dvdenc.c:*** After initialization and recording all input from the command line, the file prints out additional parameters such as which, if any, type of low delay coding is selected (intra slice or column), whether or not any frames will be skipped, and whether constrained motion estimation and any constrained search range are chosen.

- ***enc_code.c:*** New codes for high level control of coding an intra slice or column are added in this file, under a function called `CodePicture()`. This function makes many calls to other lower level functions and carry out the encoding of each macroblock. As discussed before, a call to the function `SelectMBMode()` (which is implemented in the file enc_mec.c) is part of the encoding processing loop of each MB. The new codes take advantage of such function calls and force the processing loop to check if an intra slice or column is chosen and if so, the current MB type should be set to be `INTRA` and the motion type set to be `FRAME`. The codes check for such conditions by calling either lower level function named `MyIntraSliceFunc()` or `MyIntraColumnFunc()`. These two functions return a flag indicating if that MB should be forced to be coded intra (as part of an intra slice or column). If the function `MyIntraSliceFunc()` (or `MyIntraColumnFunc()`) does not signal for the current MB to be coded intra, then the default function `SelectMBMode()` is called instead to determine the MB type. To keep up with abstraction, no hardwiring was done in the code to force the required Forward Predicted MB type. Below is an extremely simplified key portion of the new codes (for the intra slice case) added to the function `CodePicture()` in the file enc_code.c:

```
if (lowdelay) {
    if (MyIntraSliceFunc()) {
        SetMotionTypeToFRAME();
        SetMBModeToINTRA();
        PrintLocationOfIntraMB();
    }
    else
        SelectMBMode();
}
```

Similarly, the code for the intra column case is:

```
if (lowdelay2) {
    if (MyIntraColFunc()) {
        SetMotionTypeToFRAME();
```

```
            SetMBModeToINTRA();
            PrintLocationOfIntraMB();
        }
        else
            SelectMBMode();
    }
```

- ***enc_mec.c:*** Lower level functions controlling of intra coding a MB such as `MyIntraSliceFunc()`, `MyIntraColFunc`, and SelectMBMode() are included here. Operations to determine what the refreshing pattern should be, what size of the intra region, and if any frame is skipped, are all included in the two functions `MyIntraSliceFunc()` and `MyIntraColFunc()`. Again, a simplified portion of `MyIntraSliceFunc` is given below. `MyIntraColFunc` is implemented using similar steps. The functions' parameters and their actual syntax are irrelevant and not shown here but they are included in Appendix A.

```
if (row_to_be_intra >= current_row) &&
   (row_to_be_intra < current_row +
    intra_slice_size)
   returnvalue = TRUE;
else
   returnvalue = FALSE;

if (current_row = end_of_frame)
   row_to_be_intra += intra_slice_size;

if (row_to_be_intra = end_of_frame)
   Reset_row_to_be_intra_to_zero();

return (returnvalue);
```

In other words, when the row currently being processed fits inside the region to be coded intra, then the return value for the function is TRUE, else FALSE. Once the row currently being processed has reached the final row of the picture frame, the region to be coded intra on the next frame needs to move to a region below the previous intra coded row (or the region to the right of the previous intra coded column for

the intra column case). Finally, when the region to be coded intra has reached the end of the picture, it is reset and started all over again.

Not every coded picture must contain an intra slice or column. Changes were also made in `MyIntraSliceFunc` and `MyIntraColFunc` to accommodate pictures where no intra slice or column is inserted. For instance, an intra slice of size 1 (one row of intra MBs) can be inserted in every other picture without losing sequential order of the intra slices. Such skipping simulates intra updating of size equals to half a row of MBs.

If while coding a picture and the remaining rows (columns) of MBs do not fill one unit of the specified size of intra slice (column), then all those remaining rows (columns) will be refreshed during that picture. When the next frame is coded, the specified size of the intra slice (column) will be used and will start at the top most row (or left most column) of MBs.

- ***enc_rc.c:*** The number of bits generated for a coded picture is calculated in this file. Additional changes have been made to this file so that the number of generated bits are recorded onto a data file. This data is useful to illustrate the behavior of the buffer fullness level to be discussed in Chapter 4.

- ***dvdmotest.c:*** Although no specific changes were made to this file, it needs to be ran with the new low delay's picture structure setup (i.e., IPPPPPP...) so that motion vectors are computed and stored and can be used for future low delay codings where this structure is not altered.

In order for the coding to work correctly, other options must also be selected. For example, the user needs to enter a file name where the data points in the file enc_rc.c can be printed to. He also needs to specify the buffer fullness constraint and the compressed

bit rate; the one-way end-to-end delay of the coding is determined by dividing the buffer fullness constraint by the bit rate. In addition, the GOP structure must be specified by indicating how many I and B frames the coding must have. In this case, there will be only one I frame and zero B frames. Some other options are selected and will be discussed below in an example.

3.3.3 Implementation of Constrained Motion Estimation

Implementation of Constrained Motion Estimation is performed by a new function called `ConstrainSearchRangeInRefreshedRegion()`, which is located in the file enc_mec.c and can be found in Appendix A. This function replaces a previously existed function called `MakeSearchRangeInImage()` where the search range of the motion estimation process is limited to the inside of the picture frame. The new function limits the search range in the vertical direction for the intra slice case and in the horizontal direction for the intra column case. If the MB currently being processed has been refreshed in the current refresh cycle, any motion vector generated for its prediction may not come from an unrefreshed region. No restriction is placed on coding a MB which has not been refreshed in the current refresh cycle.

3.3.4 An Example

In order for the above description to be clearer, an example is given to demonstrate the working order of the low delay encoder. Perhaps a user wants to encode a video sequence called "input_video" using low delay intra slice of size 2 (2 rows of intra MB) and constrained motion estimation of range 16 pixels, he might enter the following:

```
dvdenc output_filename input_video data_filename -
bufsiz 9 -lowdelay 2 -PP 1 -gopsiz 10000 -m 0 -i 0
-bitrat 1500000 -noscdet norep1f -constrained_me 1
```

The compressed bitstream will be stored in file "output_filename" while the data for the buffer fullness will be recorded in file "data_filename." Directly from the input command above, this user wants a bit rate of 1.5 Mbits/sec and a buffer fullness limit of 16384x9 bits (147456 bits), which translate to an end-to-end delay of 147456/1500000 seconds or roughly 98 msecs. The "-PP 1" and the "-gopsiz 10000" options specify the GOP structure of IPPP... where the 10000th picture coded will be the second I frame. The user could have picked any arbitrarily large number (much larger than the total number of frames in the sequence) so that no other I frame will be coded during the sequence. The "-m 0" and the "-i 0" options force the encoder to use errors instead of bit estimates for inter/intra mode determination. "-noscdet" signals the coder to not detect for a scene change and so no I frame will be inserted if there is a scene change; "-norep1f" forces the encoder to code without the repetition of the first field. Most likely, scene changes will not occur in a low delay application such as videoconferencing.

4 Simulation Results

To study the performance of our low delay encoder, we use a 5-second (150 frames) 720x480 pixels CCIR 601 test sequence, named "Susie," which is typical of a videoconferencing scenario where a human's head and shoulders are captured. The video is coded by the MPEG2 software simulator at 29.97 pictures per second and a targeted constant bit rate of 1.5 Mbps. This bit rate represents the amount of data (i.e., 1.5 Mbits) that the encoder's buffer sends to the transmission channel per second. The rate control process within the encoder will control this bit rate by using either MB skipping or bit stuffing. We avoided codings with variable bit rate for the sake of simplicity since this variable mode is usually applicable only to specific network capability. Although the coder is capable of coding at much higher bit rates, the relatively low constant bit rate is chosen because it stresses the performance limit which allows us to make simpler comparative analysis than other setups. In addition, a bit error injector is used to introduce bit errors into the compressed video bitstream. Buffer fullness in units of bits and the peak signal-to-noise ratio (PSNR) are used as analytical tools. The definition of the PSNR of a picture is

$$PSNR = 10.0\log\left(\frac{255^2}{\frac{1}{P}\sum_i\sum_j (f_{ij}-g_{ij})^2}\right) \tag{4.1}$$

where f_{ij} represents the luminance element of the input picture, g_{ij} represents the output luminance element from the decoder of the corresponding picture, and P is the total number of pixels. The denominator inside the first bracket of equation (4.1) is the mean square error calculated between the input picture and the decoded picture. Although PSNR is not a good index for absolute video quality, it gives some insights into relative quality improvements.

Additional simple subjective viewings are conducted to check for quality factors that might not show up through the PSNR data. For documentation on detailed subjective testing techniques, please refer to [13]. The other analytical characteristic is the buffer fullness behavior of various codings. The following subsections describe the data obtained for comparison purposes using buffer fullness behavior and video quality with or without errors introduced. Important interpretation of the data is also given below. Please refer to the subsequent chapter for additional analytical discussion of these data and other experimental options.

4.1 Buffer Fullness Behavior

Recall that the MPEG2 parameter vbv_delay specifies the end-to-end delay time of the encode/decode process, not including any channel and processing delays. It corresponds to the time it takes to remove the compressed data corresponding to a picture of maximum allowable size, i.e. the specified VBV buffer size, from the decoder buffer. In our experiments, we limit that buffer size to be 147456bits (9x16384 bits), which according to bit rate of 1.5 Mbps and equation (3.1), yields a delay of slightly less than 100ms, the desirable upper bound of the delay for videoconferencing [19]. As observed in all the experiments below, the rate control system of the coder prevents any overflow (and underflow) that can cause the preset end-to-end delay to change. Network delays, processing delays, and other delays are not included in our measurements in order to preserve simplicity. Works on those related areas of low delay video coding such as transport structures to support multipoint videoconferencing and choosing the best network configuration can be found in [20] and [19], respectively.

The low delay cases presented in the following experiments are: low delay coding with intra frame updating every 15 frames (IP)[1]; intra slice updating with slice size of one, two,

and three rows of macroblocks per frame (labeled as ld1slice, ld2slice, ld3slice respectively); and intra column with size of one, two, and three columns of macroblocks per frame (ld1col, ld2col, and ld3col respectively). Usage of constrained motion estimation is indicated by an extension of "_cme" along with the constrained search range in integral multiples of 16 pixels. It should be pointed out that ld2slice and ld3col have the same refresh cycle (15 frames) as the intra frame update coding (IP).

Figure 4.1 and Figure 4.2 show the buffer fullness occupancy for the cases mentioned above. It is important to note that Figure 4.1 and Figure 4.2 record only the peak buffer fullness level of each coded picture instead of the actual image data filling and removing as shown in Figure 3.2. We also made sure that the buffer level never underflow throughout the experiments. The buffer fullness occupancy for the $(n+1)^{th}$ picture, B(n+1), is calculated by

$$B(n+1) = B(n) + R(t_{n+1} - t_n) - d_n \quad , \tag{4.2}$$

where B(n) is the buffer fullness for the n^{th} picture, R is the constant bit rate, $(t_{n+1}-t_n)$ is the time it takes to display picture n (or the time the decoder waits after decoding one picture before it begins decoding the next), and d_n is the number of bits picture n occupies. Note that for the IP case, the buffer occupancy level varies widely. Such behavior is expected due to dips occurring whenever an I frame is removed from the buffer, after which the rate control reduces the number of bits generated for each following P picture to reach back the maximum buffer limit. Because an I frame is coded every 15 frames and requires a large amount of bits, the buffer fullness level varies tremendously for the IP algorithm.

1. For this mode, the encoder is simply specified to code with a constrained buffer size and without any B pictures.

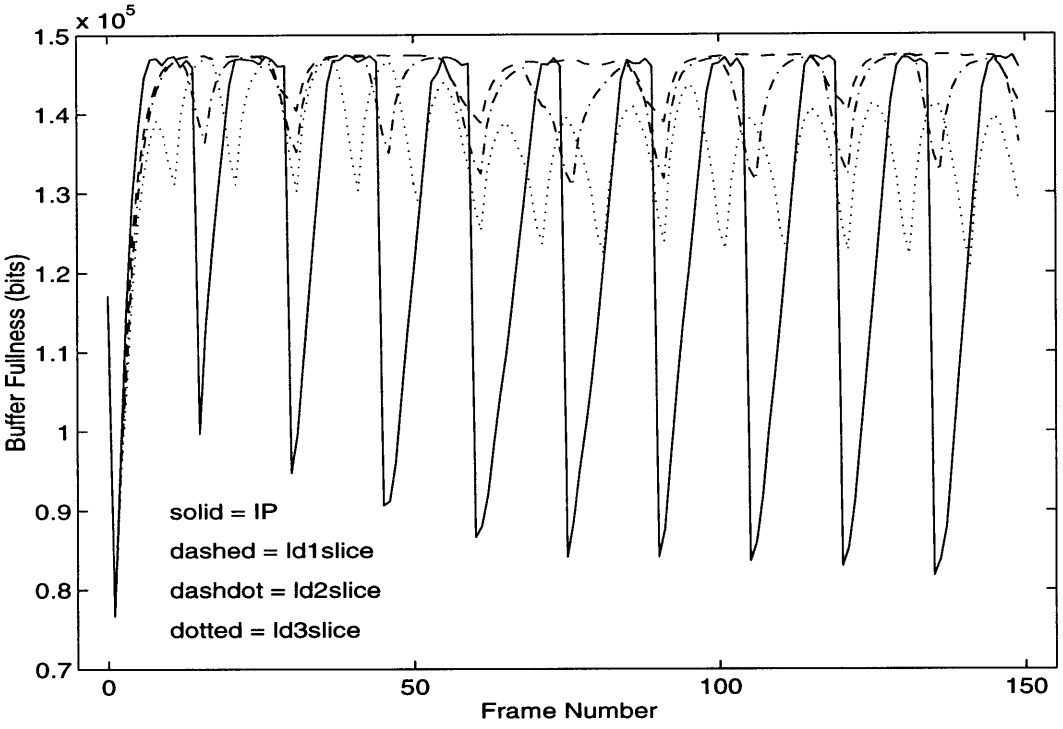


Figure 4.1: Buffer Fullness Level of IP and Intra Slice Codings.

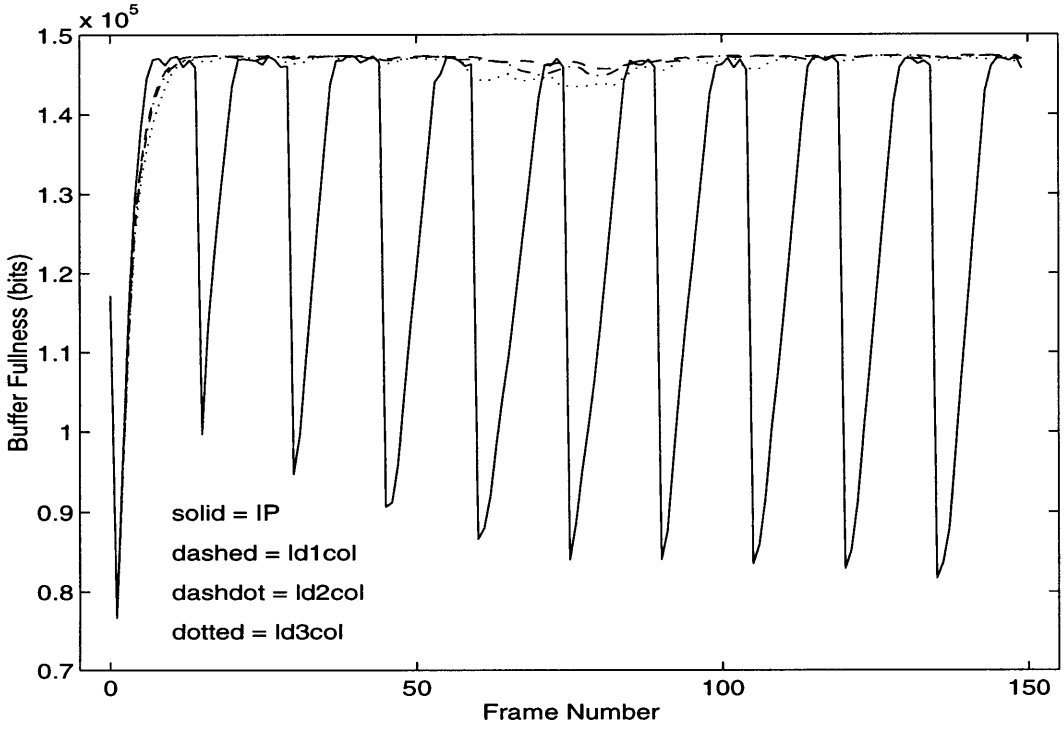


Figure 4.2: Buffer Fullness Level of IP and Intra Column Codings.

For the intra slice cases, their less varying buffer occupancy levels are periodic with the numbers of frames for which an entire picture is refreshed. For instance, ld2slice has a period of 15 frames since each frame has two rows of intra macroblocks and the picture is refreshed entirely after every 15 frames (the video sequence has size of 45x30 macroblocks). Similarly, ld1slice has a period of 30 frames and ld3slice has a period of 10 frames. Such periodic characteristic is a special situation for low motion video sequences and is contributed by the unevenness of how bits are distributed among the intra-macroblocks of a frame. A dip occurs when the intra slices reach the bottom of a frame. The rate control process has more difficulty distributing the remaining bits of a frame to code these intra regions occurring near the bottom of a frame than it does when the intra regions are on top. Recall that these forced intra regions require many more bits to code than inter regions and since coding is done MB-by-MB and in a raster scan order, many consecutive MBs require more unexpected bits near the bottom of a frame, causing the buffer fullness to drop when these frames are removed. After such a dip occurs, the rate control process readjusts the number of bits generated so the buffer fullness level reaches back to the maximum allowable level. Such dipping also does not occur when the intra slices are near the top or middle of a frame because the rate control process still has control over the number of bits to be spent coding the remaining of the picture.

As expected, this periodic behavior does not exist for intra column codings. Only several MBs (for example 3 MBs for ld3col) need to be coded intra per row of MBs, thereby not demanding a very large amount of additional bits for the intra regions. It should be noted that the periodic characteristic found in the intra slice coding only exists when coding sequences with a relatively constant background. Sequences with high motions generate highly varying number of bits and the buffer fullness level is not stable enough to exhibit such behavior.

Limiting the search range to 16 pixels each direction, constrained motion estimation in conjunction with low delay codings does not yield any significant difference in buffer fullness level. This option does not alter the GOP main structure and therefore no major changes occur. The buffer level exhibit the same behavior as if the option were not selected. Of course the level is not identical to that of a coding that does not use constrained motion estimation (using constrained motion estimation does seem to use the buffer slightly more efficient) due to different motion vectors and prediction errors being coded, but the difference is negligible when plotted with the same scaling used for Figure 4.1 and Figure 4.2. Due to such minor difference, its buffer fullness level is not presented here.

Table 4.1 below gives a summary of the average buffer occupancy level of the seven codings. Recall that 1.47456×10^5 bits is the highest possible buffer level at any point. IP coding has the lowest efficiency level resulting from the widely varying buffer level discussed above. The intra slice/column cases exhibit similar average buffer occupancy level, with the exception of a slight drop for the ld3slice. Because 135 MBs are coded

Table 4.1: Average Buffer Fullness Occupancy Level.

Codings	Average Occupancy (bits)
IP	1.2723×10^5
ld1col	1.4545×10^5
ld2col	1.4521×10^5
ld3col	1.4464×10^5
ld1slice	1.4436×10^5
ld2slice	1.4144×10^5
ld3slice	1.3516×10^5

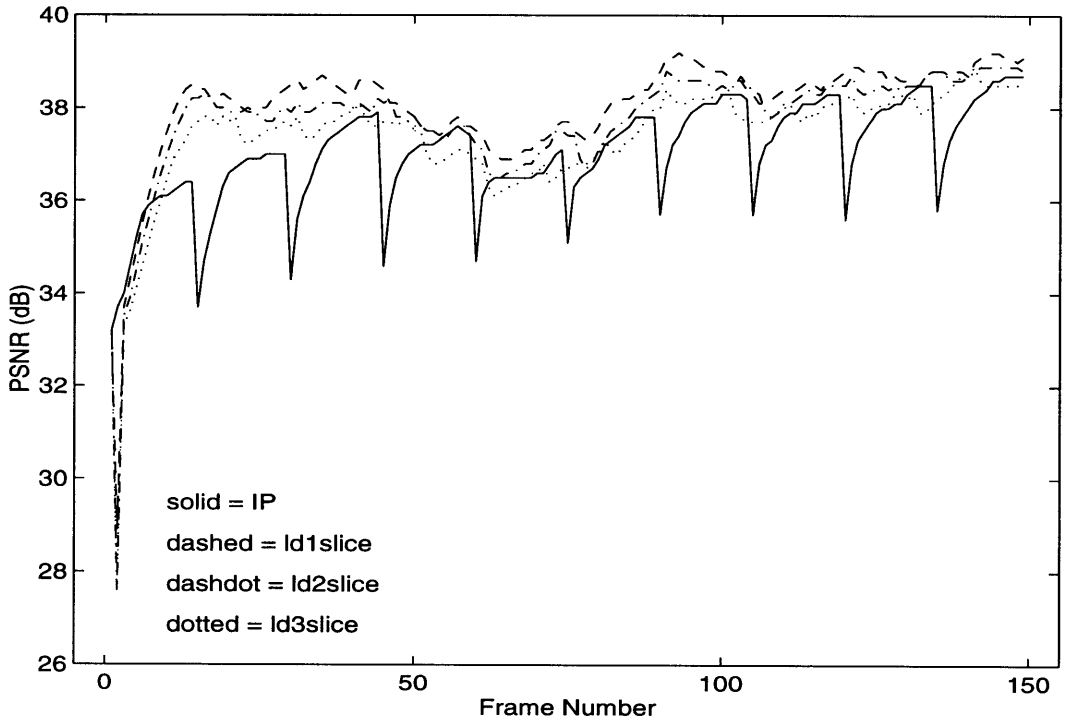


Figure 4.3: PSNRs of Susie Coded at 1.5 Mbps with No Errors.

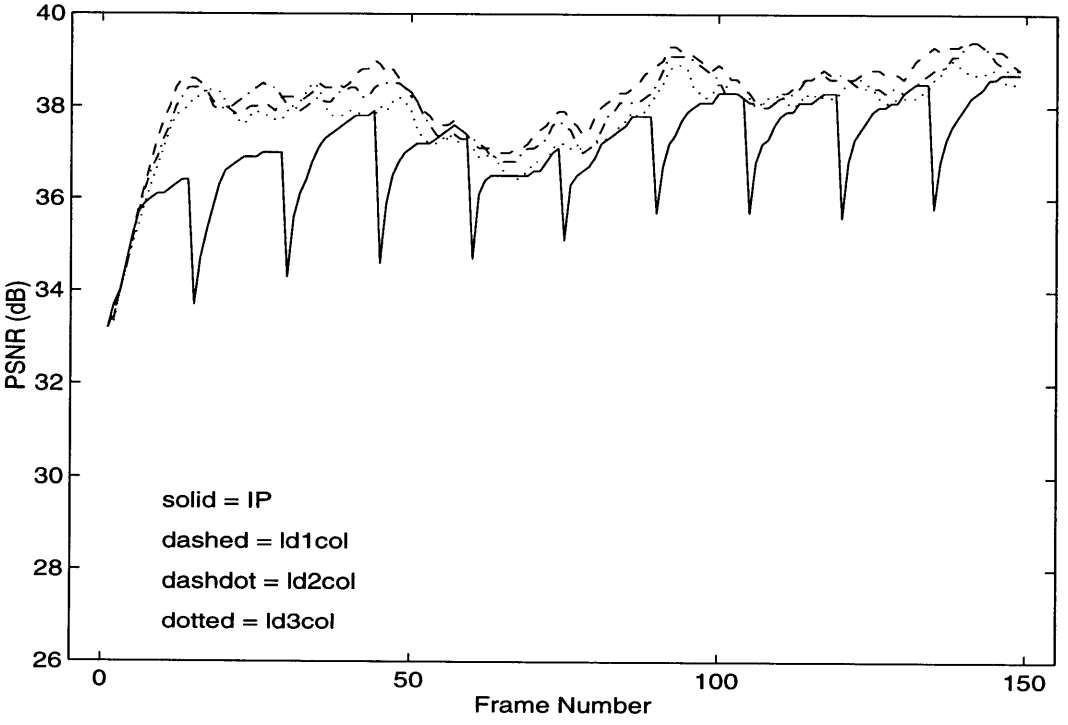


Figure 4.4: PSNRs of Susie Coded at 1.5 Mbps with No Errors.

intra per frame for the ld3slice coding, the periodic dipping of the buffer occupancy level is more pronounced and the average level is lower than the other intra slice/column cases. Obviously, ld1col has the highest efficiency because it has the fewest intra MBs (30) to code per frame compared to other codings.

4.2 Video Quality Without Errors

Because "Susie" is a relatively easy-to-code sequence, it is desirable to have a stable and high buffer fullness level throughout, which indicates that a constant and near maximum number of bits are being used to code each frame, producing uniform video quality. Lower PSNR is observed in Figure 4.3 and Figure 4.4 for the IP case compared to the intra slices/columns cases. This behavior is clearly due to the unsteady buffer fullness level of the IP coding described above. In other words, an I frame requires higher number of bits to achieve good image quality; having very low buffer limit results in significantly lower quality. The PSNRs for the intra slices/columns cases are relatively more stable due to the steady number of bits produced by the lower bit costing continuous series of P frames. Note from Figure 4.1 that the IP coding tries to maximize buffer efficiency and gives the most room for the I pictures by gradually increasing the buffer level after each I frame is removed from the buffer. Simply, fewer bits to be coded per I frame translate to considerable lower quality. This can be seen by both the PSNR plot in Figure 4.3, Figure 4.4, and also the video clips of the 14th frame (an I picture) shown in Figure 4.5. Blocking effects and loss of details in Susie's eyes and hair are apparent in Figure 4.5.b.

It is also important to note that the PSNRs for the 2nd frame of the intra slice cases are below 29 dBs. Recall that the first frame is an I frame so the same PSNR is obtained for all cases. After the first I frame is removed, it is natural for the rate control to code using fewer bits for the first P frame. However, the immediate first row(s) of the picture is

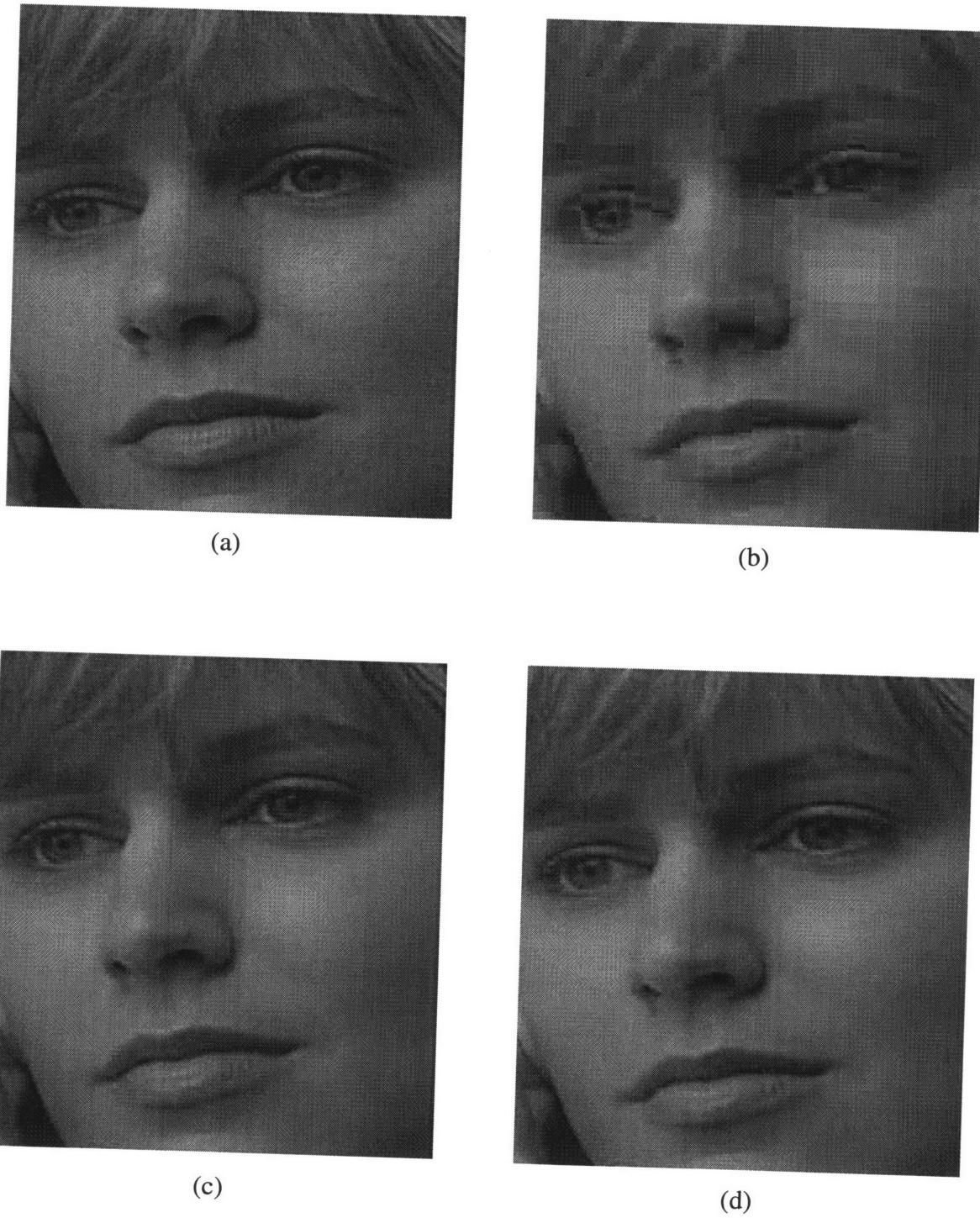


Figure 4.5: Sample video clips of the Susie video sequence and various codings; (a) original input; (b) coded with IP; (c) coded with ld2slice; (d) coded with ld3col.

forced to be intra, requiring more bits which the rate control process does not expect for a typical P frame. This sudden lack of adequate bits results in lower quality for the second frame in a refresh cycle, a behavior that is most obvious for the ld3slice coding since it codes the most intra MBs. The larger the intra region, the more pronounced this behavior is for the second frame.

Within this error-free environment, the use of constrained motion estimation slightly decreases image quality. This is expected in our design due to the constrained search range imposed upon the motion vectors generated during the motion estimation process. Such limit results in less accurate predicted region for some target MBs and therefore higher prediction errors are generated. After passing through lossy and relatively coarse quantization due to the small bit budget, these high prediction errors are ultimately responsible for lower image quality. Because of the small quality difference, it is not easy to notice the degradation when viewing the sequence in real-time. This result is illustrated in Figure 4.6, where performance between IP, ld2slice, ld3col, and ld2slice_cme2 is included. As mentioned before, IP, ld2slice, and ld3col have the same refresh cycle and should be compared with each other.

For the intra slices/columns cases, we find that the quality performance of intra column refreshing is comparable to that of intra slice refreshing. Because ld3col has the same number of intra macroblocks to code per frame than does ld2slice, its video quality is equivalent to that of ld2slice. One can argue that since ld3col exhibits a more stable buffer fullness level, it's video quality should be slightly less flickering than ld2slice, but this difference is not noticeable when viewing. In general, larger intra regions result in lower quality in a noise-free environment due to the buffer fullness constraint. In terms of buffer fullness behavior, intra column refreshing is more stable than both intra slice and intra frame refreshing at the chosen delay.

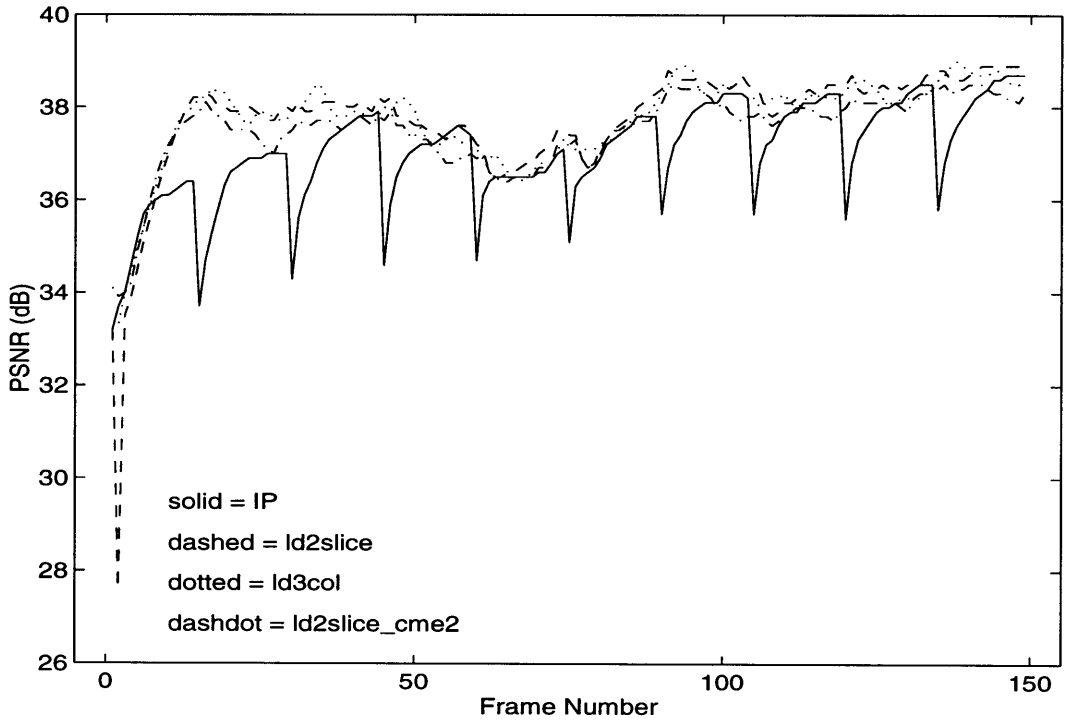


Figure 4.6: PSNRs of Codings with Same Refresh Cycle of Susie at 1.5 Mbps.

4.3 Transmission Error Robustness

4.3.1 The Error Injector

For the simulations with transmission errors, a random bit error injector is implemented using the two-state Markov-chain shown in Figure 4.7. It is assumed that the initial state is not an error. This program can be specified to simulate either single random bit errors or bursty bit errors. Setting the error burst to be 384 bits simulates an ATM cell loss where each cell contains a 48-byte (or 384 bits) payload. The cell loss rate can also be specified.

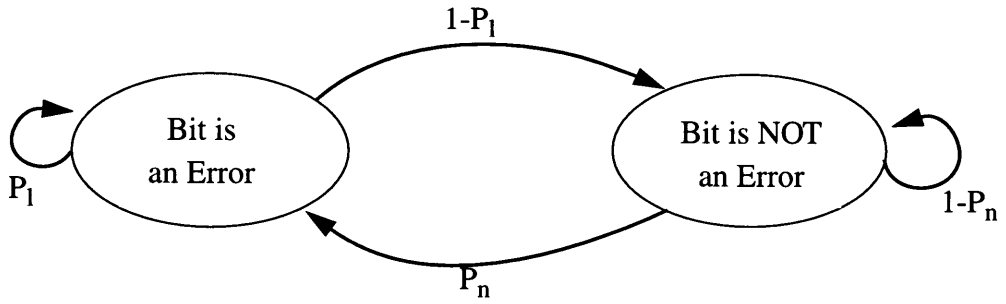


Figure 4.7: Bit Error Rate Modeled as a Two-state Markov-chain.

Two key probabilities determine the error rate in the program: the probability that the current bit is an error given that the previous bit is not (P_n), and the probability that the current bit is erroneous given that the previous bit is (P_l). If the probability that a bit is an error is P, then it is equals to the *sum of* the probability that the bit is an error given the previous bit was not an error multiplied by the probability that the previous bit is an error (P_l * P), *and* the probability that the bit is an error given the previous bit was not an error multiplied by the probability that the previous bit was not an error (P_n * (1-P)). Therefore,

$$P = P \cdot P_l + (1 - P) \cdot P_n \quad , \tag{4.1}$$

which can be simplified to

$$P = \frac{P_n}{(1 - P_l + P_n)} \quad . \tag{4.2}$$

A burst of bit errors occurs when a bit is an error given that one or more previous bits is also an error. A mean burst length B is defined as the mean burst of consecutive bit errors. The probability of a burst length of 1 is the same as the probability that the next bit is not an error ($1-P_l$). Similarly, the probability of a burst length of 2 is equal to the probability that the next bit is an error and the one after that is not an error (P_l * ($1-P_l$)), etc. The mean burst length is then equal to:

$$B = (1 - P_l) + 2P_l(1 - P_l) + 3P_l^2(1 - P_l) + \ldots \quad , \tag{4.3}$$

which when summed up equals:

$$B = \frac{1}{(1 - P_l)} \quad . \tag{4.4}$$

Rearranging equations (4.2) and (4.4) and substituting yield:

$$P_n = \frac{P}{B(1 - P)} \quad , \tag{4.5}$$

$$P_l = 1 - \frac{1}{B} \, . \tag{4.6}$$

Equations (4.5) and (4.6) are used for the error injector program where the user may specify the BER (P) and the mean burst length (B). A standard C-library pseudorandom number generator determines if a current bit is an error by comparing the random probability generated with one of the two corresponding probabilities determined above using the user's input. If it is an error, the bit is then flipped, else it is unchanged. The entire program for this error injector is included in Appendix B.

4.3.2 Experimental Results

In the following experiments, single random bit errors (burst length of 1) with BER of 5×10^{-6} are used, which is much higher than a typical channel suited to MPEG2 application [35]. Encoded bitstreams are injected with bit errors and then decoded. We did not consider introducing errors particularly to the headers of the bitstream because they occupy a very small fraction of the bitstream. For work on error concealment for picture header loss, please refer to [29]. An example of how a MB is affected by a bit error is illustrated in Figure 4.8, which is frame 29th of the ld2slice coding. The error is located on Susie's lower lip.

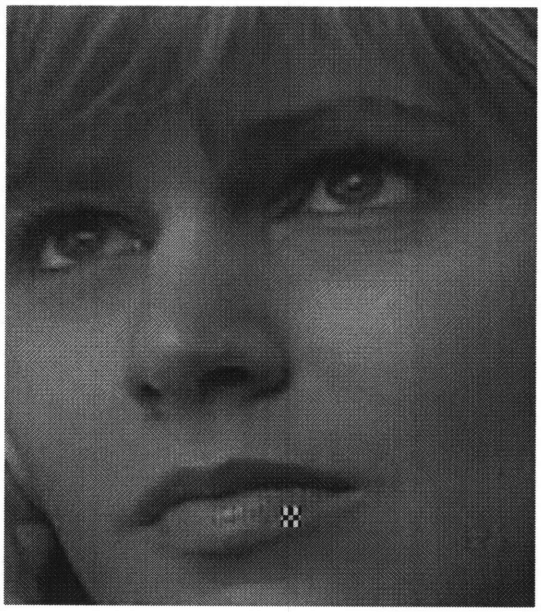


Figure 4.8: An Example of How a Bit Error Can Affect Picture Quality.

The other results are shown in Figure 4.9 and Figure 4.10 where the PSNRs of the error injected videos are subtracted from the error-free sequences, yielding the relative quality degradation. The reader should keep in mind that these are only relative degradation of quality from the error-free sequences. Therefore, less degradation does not necessarily constitute better overall image quality, but rather better error reduction performance. A coding can have less degradation from errors but still has lower overall quality than another sequence.

From these figures, no definite conclusion can be made about which coding is better in terms of relative error robustness--IP coding, intra slice or intra column coding. Each method has its own error removing scheme and each has its advantages and disadvantages. The result depends merely on the chance of where the error is located. If an error is located just before an intra slice (or intra column) passes by or on a frame just before an intra frame, it is quickly removed. What we can conclude from Figure 4.9 and Figure 4.10

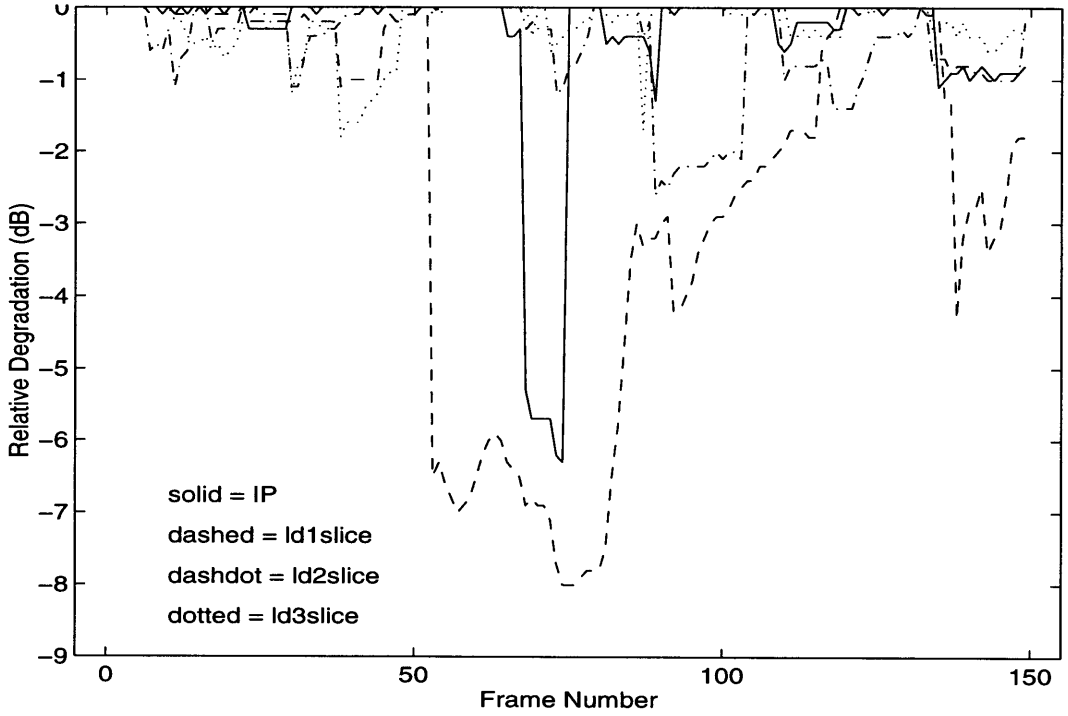


Figure 4.9: Relative Degradation (dB) in Susie with Errors.

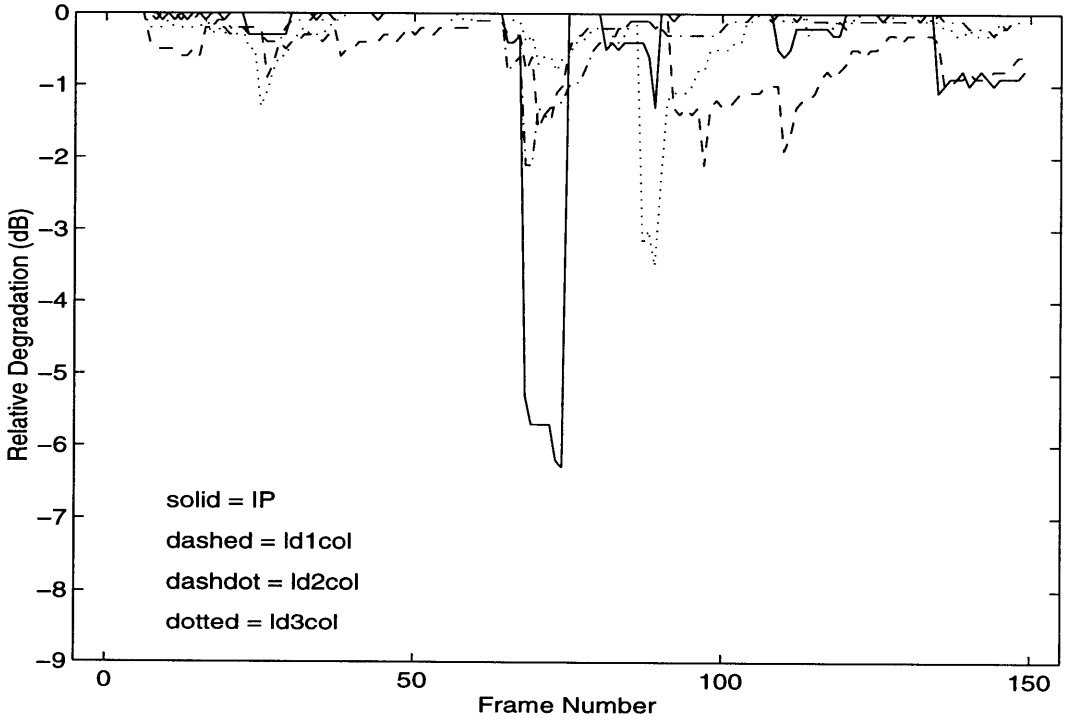


Figure 4.10: Relative Degradation (dB) in Susie with Errors.

is that intra slice/column coding typically has comparable transmission error resilience as IP coding. It is also noticeable that larger intra regions tend to reduce errors quicker. Therefore, larger intra slice or column size should be coded in noisier random bit error environments in order for errors to be cleared more quickly. As Figure 4.9 and Figure 4.10 indicate, ld2col and ld2slice perform relatively better than ld1col and ld1slice respectively in this noisy environment simulation. Errors propagate through more frames for the cases of ld1slice and ld1col since these code half as many intra macroblocks per frame and therefore their refreshing rates are half as slow. Surprisingly, ld3col does not improve its relative degradation as well as ld2col because some intra regions actually contain errors. IP coding quickly removes errors when they are located just before an error-free I frame. In Figure 4.9, a serious error occurs on its 7th P frame of the 5th group of picture (GOP) and propagates until the next error-free I frame (frame 75). A comparative performance summary of various coding methods is provided by Table 4.2. In this table, the average $PSNR_{average}$ is determined by:

Table 4.2: Average PSNR (dB) for Several Coding Methods of Susie at 1.5 Mbps.

Codings	No Errors	With Errors	Relative Degradation
IP	37.0611	36.6134	-0.4477
ld1col	38.1691	37.6054	-0.5638
ld2col	37.9926	37.7826	-0.2101
ld3col	37.7503	37.4396	-0.3107
ld1slice	38.0497	35.6732	-2.3765
ld2slice	37.7631	37.2054	-0.5577
ld2slice_cme2	37.5034	37.0866	-0.4168
ld3slice	37.3940	37.1315	-0.2624

$$PSNR_{average} = \frac{1}{N} \sum_{n=1}^{N} PSNR_n , \tag{4.7}$$

where N is the total number of video frames in the sequence and $PSNR_n$ is the PSNR of the nth frame as defined in equation (4.1).

In our experiments, constrained motion estimation does not prove to be helpful at all times in terms of reducing error propagation. This feature sometimes works as intended by eliminating errors interleaving through refreshing slices or columns, but at other times the region containing errors is not within the constrained search range and is free to propagate until that region is coded intra. It is still possible that the error occurs in recently refreshed region, thus propagating through the sequence. Overall, error reduction performance is comparable to the low delay cases without constrained motion estimation. This can be concluded from Figure 4.11, where PSNRs degradation for the four key error-

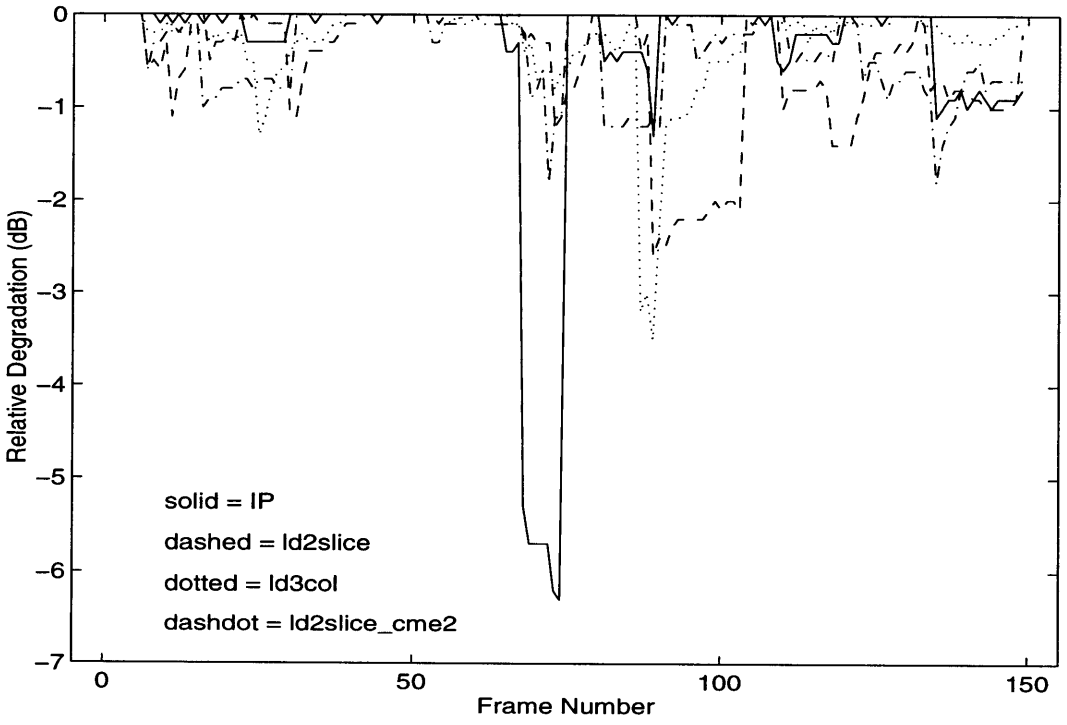


Figure 4.11: Relative Degradation (dB) in Susie with Errors.

injected cases are observed. Note how using constrained motion estimation with intra slices (ld2slice_cme2; dashdot) reduces error propagation that ld2slice encounters from frame 90 to 105. However, not selecting this option seems to be better at other times (e.g., frames 125-140). It is important to note that although bit errors are injected at the same bits for each encoded bitstream file, they do not necessarily occur on the same visual locations of the video pictures because each coding yields a slightly different number of bits for its encoded file. This makes it more difficult to pinpoint the same error on two coded sequences.

5 Further Discussion of Simulations

This chapter focuses on additional possible experimental options the user may select to obtain optimal video quality. It also provides discussions on general video quality, error robustness through various strategies, and some miscellaneous topics.

5.1 Buffer Fullness Occupancy

In an attempt to achieve less flickering video quality for IP codings, it is possible to constrain the buffer fullness level to be 80% of the maximum buffer size before the coding of an I picture. Because the buffer occupancy level may not exceed the maximum buffer size, it is inappropriate to allow the buffer fullness to reach near this maximum size before an I picture is coded. The I picture requires many more bits than a P picture and thus needs extra room. Keeping the buffer fullness level to 80% of maximum size before an I picture allows for this extra 20% space and in turn yields less varying picture quality. Obviously, this 80% of the buffer fullness also prevents underflowing. Although the buffer fullness level for the P pictures might not be near the buffer size limit, it does not necessarily constitute lower quality. Having a stable buffer fulless level indicates that the bits being generated per picture are steadily removed. Problems only occur when an I picture is coded and if the buffer level is too low, then the risk of underflowing increases. Similarly, a very high buffer level may result in overflowing.

5.2 Video Quality

In general, the higher the bit rate used for a coding, the better quality is achieved. When a low bit rate is used, quantization is much coarser, resulting in major loss of information especially with the higher frequency DCT coefficients. If the bit rate is too low, the encoder does not have enough bits to code a picture and it will begin to zero out DCT coefficients, causing detrimental performance. Video with high motion also requires more

bits since it is more difficult to code than video with little motion. Therefore, when coding such video, it is harder to keep a low bit rate and small buffer size which are necessary to achieve low latency.

Two very commonly found video artifacts are blocking effects and motion jerkiness. Blocking effects are most often caused by the lack of higher frequency DCT coefficients. Because the MPEG technology is block-based, such effects become apparent very easily. On the other hand, motion jerkiness is often caused by the lack of motion information such as missing or incorrect motion vectors. These two artifacts contribute greatly to the subjective quality of any video sequence.

5.3 Error Resilience

In the previous chapter, we concluded that larger intra regions tend to reduce error propagation more quickly than smaller regions. However, this conclusion is not always true. When a slice has an error, the decoder normally rejects the slice and looks for a new slice header. If a large intra slice has an error and is thrown away, serious image quality degradation will result and propagate at least through the remaining refresh cycle. It has been well-known that smaller slices yield better error robustness but may cause higher overhead cost [25]. Codings with adaptive slice size yield better efficiency, but are only applicable for cell or packet-based transmission. Overall, the size of the intra region directly depends on the error rate of the transmission channel. For a relatively error-free environment, smaller intra region offers the best quality since fewer bits are spent on intra MBs. The saved bits are put to use towards a finer quantization, yielding better quality. From our experimental data, it seems best to use 2 rows or 3 columns on intra MBs per picture at the chosen BER.

It is often thought that intra slice coding is better than intra column for handling bursty errors. If several consecutive MBs of the same row are erroneous, they can be eliminated much quicker using an intra slice than intra column. Also, most video sequences often exhibit much more horizontal motion than vertical. In this case, intra slice is also better than intra column; there are fewer chances for error to propagate through the intra region, saving from using constrained motion estimation which can potentially result in very high prediction errors.

It is critical to point out that when lower BERs are chosen ($<1x10^{-6}$) for all error injected codings, relative degradation becomes negligible and overall quality performance approaches what we found in an error-free environment. For instance, when the BER is chosen to be $1x10^{-9}$, which is a more typical error rate [11], quality degradation is virtually undetectable for the "Susie" sequence! Figure 5.1 shows the relative degradation resulting from $5x10^{-8}$ BER. These small degradations go unnoticed when viewing real-time video. As pointed out above, when a channel contains very few errors, codings with very small slices (or skip pictures to obtain longer refresh cycle) should be chosen to maximize efficiency. Nevertheless, one must be careful not to violate MPEG2's maximum MB "refresh cycle" of 132 frames.

Various codings using constrained motion estimation were also performed. As before, its usage does not seem to be very beneficial so these results are not included here. We suspect that it might be most advantageous to code with constrained motion estimation under environments with extremely high error rate (e.g., 10^{-2} or 10^{-3}), where an additional error concealment technique should be employed instead of simply using intra slice/columns as an error propagation reduction technique.

Several other options can be selected to improve image quality and error resilience of the low delay codings. The options discussed below are much more advanced than simply

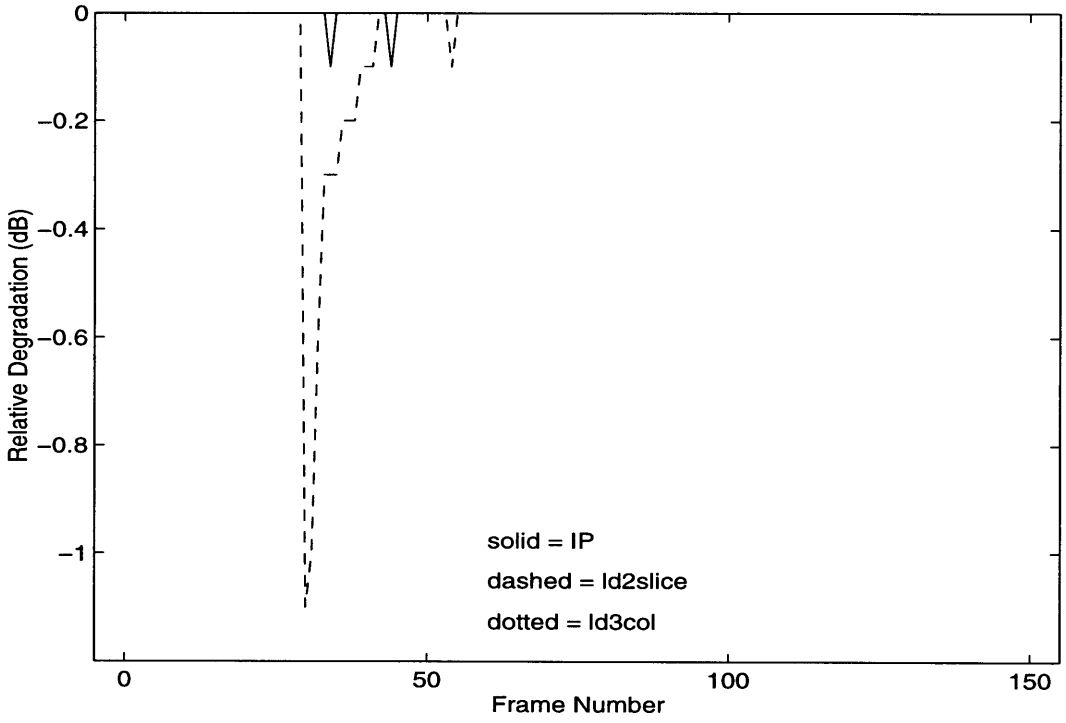


Figure 5.1: Relative Degradation (dB) in Susie at $5x10^{-8}$ BER.

replacing lost data with data from the previous picture. Among them are usage of temporal predictive concealment such as error concealment motion vectors, usage of spatial predictive concealment, or any scalability options discussed before in Chapter 2. For a more in-depth discussion of popular error recovery techniques, please refer to [32] and Appendix D.13 of [4].

- **concealment motion vectors:** Normally, when a macroblock is lost, its motion vector can be estimated from neighboring macroblocks' vectors by, say, averaging them or apply a specially designed filter on them. However, this method is not successful when the neighboring macroblocks are coded intra; intra macroblocks normally do not contain any motion vectors. When "concealment_motion_vectors" is selected, the encoder includes motion vectors for intra macroblocks. It is important that these accompanying motion vectors are transmitted separately (e.g., through different data

packets) so that they are recoverable in the event that the image macroblock is lost.

- **spatial predictive concealment:** Generation of lost macroblocks can be done by interpolation from neighboring macroblocks. This techniques is particularly effective for video with fast motion where only the DC coefficient and lowest AC ones are valuable. It is unlikely that the higher AC coefficients can be interpolated successfully in the first place.

- **scalability options:** Depending on the transmission channel's configuration and capability, scalability options may be applicable for the main purpose of error concealment. Critical data (base layer) are sent via a reliable channel while the enhancement layer can be sent over a more error-prone channel. Even if the enhancement layer is lost, the primary image data can still be decoded from the base layer. Moreover, scalability codings can make better use of network's efficiency level by sending only the base layer when the network is most busy.

Researches have been done to study the performance of MPEG codecs in the presence of various types of errors; their documentations can be found in [33] and [34]. Some examples of other works revolving around specific algorithms for error concealment are special packing of ATM cells [31], usage of Error Correcting Codes (ECC) [28], usage of multi-resolution motion estimation [30], and usage of the Projection Onto the Convex Set (POCS) algorithm [26].

Generally speaking, error recovery is really a decoder's problem. The encoder can only try to minimize for such mishaps by providing additional information so the decoder can use to consume errors. Typically, the decoder has a lot more knowledge about the type of errors that the bitstream has encountered and hence it is more successful with error

recovery. Plenty of work has been done on decoders with special error concealment strategies. Two examples of such a decoding system can be found in [27] and [35].

5.4 Further Discussions

Throughout the project, the input video sequences for these codings are all interlaced. The MPEG2 encoder selects the best prediction mode for the codings and in turn handles interlaced video coding efficiently, which is not possible with MPEG1 technology. MPEG2 also features nonlinear macroblock quantization table, alternate scanning, new intra VLC, adaptive motion compensation, and adaptive transformation in frame or field mode, all of which allows better codings, especially with sequences containing high motions. For a more thorough comparison between MPEG1 and MPEG2 coding standards, one can refer to [18].

Recall that we decided not to code using variable bit rate. This option is especially advantageous when used to maximize network's efficiency. When the network is busy, video should be coded at low bit rate to minimize congestion. When the network is less busy, higher bit rate can be coded to achieve the highest quality possible. We avoided this option because network delay is not accounted into our end-to-end delay and codings with constant bit rate allows for easier analytical comparison.

Besides intra frame, intra slice and intra column updating, another possible implementation of the intra MB refreshing is pseudo-random MB updating. MBs are updated pseudo-randomly with this technique and the entire picture is refreshed after a specified refresh cycle. It is somewhat exciting to have a refreshing algorithm that is much less predictable than intra frame, slice, or column, but in terms of video quality and error resilience, it does not perform better. The same number of MBs still needs to be refreshed for

a certain refresh cycle and errors' behavior is completely random. In fact, using this technique allows for much easier error propagation than intra frame, slice and column.

6 Conclusions

In this thesis project, the general MPEG technology has been described along with low latency performance and channel error robustness of low delay codings using various intra refreshing strategies. The primary features of both MPEG1 and MPEG2 standards are fully covered. Also presented are the attributes of H.261 and H.263 videoconferencing standards and the design and implementation of the low delay intra frame, intra slice, and intra column macroblock refreshing strategies. Low latency, video quality, and error robustness performance are illustrated with experimental simulating software. In addition, further discussions of these experiments and other options are given.

Experimental simulations are performed on a 5-second video sequence named "Susie" that resembles a videoconferencing application, which is expected to be a popular low delay application. From the cases presented, we observed that codings using intra slice/ column macroblocks refreshing clearly outperform IP coding in terms of overall quality in an error-free environment. This result is expected due to the tightly constrained buffer size, not giving the I pictures enough bits to achieve desirable quality. With the chosen BER, it is unclear which coding method is best for relative transmission error reduction; intra slice/column codings have comparable performance to that of IP coding. Nevertheless, intra slice/column codings always achieve higher overall quality compared to IP coding as predicted, especially when lower and more typical BERs are chosen. Intra column updating is favorable with its smooth buffer fullness level, while intra slice updating seems to fit better for removing bursty errors in consecutive macroblocks in the same row. IP coding is advantageous in terms of reducing error propagation with a single error-free I frame, without the hassle of constrained motion estimation. However, it is difficult to conclude from our experiments if constrained motion estimation has helped significantly. The clear result is that the fewer intra macroblocks are inserted, the higher the video quality in

a noise-free transmission environment. With caution, more intra macroblocks should be inserted per frame in a noisy environment for quicker error elimination. If nothing is known about the channel that the low delay application is running on, it will be difficult to decide how many intra macroblocks to be coded per frame. Perhaps it is best to assume that the channel is noisy and proceed accordingly.

The low delay intra frame/slice/column codings presented in this paper are simple in design, yet extremely effective in achieving low delay time compared to standard IPB coding. With the price of higher complexity, future research can concentrate on other algorithms of intra macroblock refreshing to further improve the video quality. For example, an algorithm that refreshes the macroblocks which have been used the most for motion estimation can be applied. A frame dropping feature can be implemented to handle scene changes while maintaining low delay. Yet, another possible macroblock refreshing strategy is to use leaky prediction whenever prediction errors are generated--the predicted frame is added by a (usually) small factor of the original input image, hence updating the image gradually. In addition, further fine-tuning of the rate control algorithm may yield slightly higher image quality. Regarding error propagation reduction and consumption, it is best to leave the job to the decoder since some information about the errors can be obtained there. However, for our experiments, good choices of sizes for the intra regions help reduce random bit error propagation considerably. Because the overall algorithm performs well and since it fully complies to the MPEG2 standard and thus the bitstream it generates is decodable by any standard MPEG2 decoder, the intended goals of this project have been achieved.

Appendix A

Modifications of the MPEG2 Software Encoder

```
/************************************************************************
COMPLETE FILE OF DVDENC.C IS INCLUDED.
/************************************************************************

  Name:       : dvdenc

  Description  : Main program

  Author, date : Tri Tran, July 1, 1996.
  Based on program by Peter Westerink.

************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curses.h>

#include "enc_pre.h"
#include "enc_code.h"
#include "enc_act.h"
#include "enc_sta.h"
#include "enc_dct.h"
#include "picture.h"
#include "enc_huf.h"
#include "enc_rc.h"
#include "enc_zz.h"
#include "enc_quant.h"
#include "code.h"
#include "macros.h"
#include "format_convert.h"
#include "enc_ff.h"
#include "enc_mec.h"

#include "dvdglobvars.h"


#define CODNAMEXT      "m2v"          /* file name extension */


main( int argc, char *argv[] )

{
 int Bpicnr;                    /* B picture number */
 int first_frm_num, last_frm_num;      /* source frame count */
 int ds_fd;                     /* data stream file descr. */
 char filnam[128];                 /* file name */
 Picture dummy_image;                  /* structure for parameters */
 Picture *swap_ptr;                /* pointer for swapping */
 Picture *c_image;                 /* current picture pointer */
 Picture *p_image;                 /* previous picture pointer */
```

```
  Picture image1, image2, b_images[2];  /* pictures */
  int skip_bs;                          /* skip B's flag */
  unsigned long int large_buffer_size=1300000; /* picture buffer */

  void make_hufenc();

/**** Old way of setting defaults: this should eventually go ****/

  SetOrigZero( 0 );
  SetScalePreviousAct( 0 );
  SetGlobalScanPattern  ( ADAPTIVE_SCAN );
  SetGlobalIntraVLCTable( MPEG_1 );
  SetGlobalDCPrecision  ( NINE );
  SetGlobalQScaleType   ( NONLINEAR );
  SetRateControl( 1 );
  SetLuminanceMask( NONE );
  SetActivityMsrScaleNonLinear( 2.0 );
  SetActivityMsrScaleLinear( 0.05 /64.0);
  SetActivityMsrOffset( 0.5 );
  SetActivityMsrCeiling( 2.0 );
  SetRCScale( 1.0 );
  SetActivityTransform( NONLIN );
  SetForceFieldFrameFlag( NEITHER );
  SetQtables( TM );
  SetOffsetFrac( 0.0 );
  SetQmethod( TM_NO_ROUND );
  SetHadFilterType( 3 );
  SetHadFilterPrecision( 2 );
  SetMaskNum( 1 );
  SetQuanitserSubset(0);
  DefaultRFFAlg();
  SetVBR( 0 );

  SetConversionMethod(CHIP1);

  InitAvMvBits();
  InitSequenceStats();
  InitMotionAlgorithm();
  ClearRcAlg();

/**** Parse command line ****/

  parscomlin_dvdenc(argc,argv);

/*==> INITIALIZE PICTURES <==*/

  c_image = &image1;
  p_image = &image2;
  Picture_Initialize( c_image );
  Picture_Initialize( p_image );
  Picture_Initialize( &dummy_image );
  Picture_Initialize( b_images+0 );
  Picture_Initialize( b_images+1 );

/**** For now, copy new parameters into old ones ****/

  strcpy(dummy_image.input_name,orgnam);
  dummy_image.width             = ncols;
  dummy_image.height            = nrows;
```

```
  dummy_image.chroma_format_input       = chroma_format_input;
  dummy_image.chroma_format_coding      = chroma_format_coding;
  dummy_image.n_mpeg                    = npicsGOP;
  dummy_image.m_mpeg                    = m_mpeg;
  dummy_image.concealment_motion_vectors = concealment_motion_vectors;
  dummy_image.picture_structure         = picture_structure;
  dummy_image.temporal_order            = temporal_order;
  dummy_image.non_interlaced_sequence   = non_interlaced_sequence;
  dummy_image.picture_format            = picture_format;
  if (save_recon)
    strcpy(dummy_image.recon_name,recnam);

  first_frm_num = frm0;
  last_frm_num  = frm0 + nfrms - 1;

  fprintf(stderr,"\n============================================\n");

/*==================================*/

/**** Continue ****/

  StudioQtables( &dummy_image );

  if ( !manual_srange_frm )
    DefaultSearchRange_Frm( &dummy_image );

  Picture_CopyAttributes( &dummy_image, c_image );

/*==> COMPRESSION INITIALIZATION <==*/

  make_hufenc();
  allocate("Picture", large_buffer_size);
  Picture_CopyAttributes( c_image, &dummy_image );
  initdct();

/**** Very 1st GOP: code B's immediately after 1st I (open GOP)? ****/

  if ( code_bs_after_first_i ) {
    BackwardsOnly( 1 );       /* predict backwards from I */
    skip_bs = 0;              /* do B's preceding I (disp order) */
  }
  else {                    /* default for start simulations: */
    BackwardsOnly( 0 );       /* predict B's both directions */
    skip_bs = 1;              /* but need other ref pic first */
  }

/**** Further initialization ****/

  InitPreProcess(&dummy_image,first_frm_num,last_frm_num,write_orig,0);
  SetQScaleTypeToGlobal( &dummy_image );

  AllocateIQTableSpace( (422 == chroma_format_coding) ? 4 : 2 );
  SetupIQ( &dummy_image );
  Picture_SetType( b_images+0, B );
  Picture_SetType( b_images+1, B );

/**** Now that all is initialized, print parameter settings ****/

  printparams_dvdenc();
```

```
/**** Open (create) data stream file ****/

 sprintf(filnam,"%s.%s",codnam,CODNAMEXT);

 ds_fd = creat(filnam,0644);
 if (ds_fd == -1) {
  fprintf(stderr,"%s: Error creating %s\n",argv[0],filnam);
  exit(EXIT_FAILURE);
 }


 /*** Open (create/append) buffer fullness file ****/

 if ((to = fopen(bufnam,"a")) == NULL) {
  perror(argv[2]);
  exit(1);
 }


/**** Open files for DVD statistics gathering ****/

 dvdbufconstats_open(codnam);

/*===> MAIN PROCESSING LOOP <===*/

 while ( 1 ) {

  /**** Set-up picture structures for c_image ****/

  /* Put attributes in current picture structure */
  Picture_CopyAttributes( &dummy_image, c_image );
  /* Make p_image the previous picture of c_image */
  Picture_SetPrev( c_image, p_image );

  /**** If GOP is closed, determine if c_image I or P ****/

  if( IsClosedGops() ) {
   /* Set c_image to the correct ref frame type, I or P */
   Picture_SetType( c_image, RefTypeClosed( c_image ) );
   /* Set "skip B-pictures" if c_image turns out to be an I */
   skip_bs = ( c_image->picture_type != P);
  }

  /**** Set-up B picture structures (if not skipped in 1st GOP) ****/

  if ( !skip_bs ) {
   /* Loop over the B-picture count */
   for (Bpicnr = 0; Bpicnr < m_mpeg-1; Bpicnr++) {
    /* Put attributes into this B-picture structure */
    Picture_CopyAttributes( &dummy_image, b_images+Bpicnr );
    /* Make p_image the previous picture of this b_image */
    Picture_SetPrev(  b_images+Bpicnr, p_image );
    /* Make c_image the future picture of this b_image */
    Picture_SetFuture( b_images+Bpicnr, c_image );
   }
  }

  /**** Get the actual picture data for c_image and b_image+i ****/
```

```
    /**** Why isn't this done first, and put in the while () ? ****/

    if ( GetRefFrameAndBFrames( c_image, b_images, skip_bs ) )
      break;    /* if no more data (files), break the main loop */

    /**** If GOP is open, determine if c_image is I or P ****/
    /**** RefTypeOpen() does scene change detection (really) ****/

    if( !IsClosedGops() ) {
      Picture_SetType( c_image, RefTypeOpen( c_image ) );
    }

    /**** Encode the reference picture c_image ****/

    CodeFrame( c_image, code_bs_after_first_i, first_frm_num,
            last_frm_num, enforce_buff );
    dump_buff( ds_fd, codnam );
    if (save_recon == 1) {
      Picture_Save( c_image );
    }

    /**** Encode the B-pictures b_image+i (if not skipped) ****/

    if ( !skip_bs ) {
      for (Bpicnr = 0; Bpicnr < m_mpeg-1; Bpicnr++) {
        CodeFrame( b_images+Bpicnr, code_bs_after_first_i,
                first_frm_num, last_frm_num, enforce_buff );
        dump_buff( ds_fd, codnam);
        if (save_recon == 1) {
          Picture_Save( b_images+Bpicnr );
        }
      }
    }

    /**** Now that we have started, predict B's both directions ****/

    BackwardsOnly( 0 );
    skip_bs = 0;

    /**** Swap c_image and p_image ****/

    swap_ptr = c_image;
    c_image  = p_image;
    p_image  = swap_ptr;

  } /* END while (1) */

/**** Be neat: close files for DVD pass 1 statistics gathering ****/

  dvdbufconstats_close();

/**** Other finishing statements that were always here ****/

  close_data_stream( codnam, ds_fd );
  PrintSequenceStats( );
  PrintInterpStats();
  PicStats2Disk();
  fclose(to);
/**** Exit the program (used to be return ???) ****/
```

```
 exit(EXIT_SUCCESS);
}
```

COMPLETE FILE OF PRINTUSAGE_DVDENC.C IS INCLUDED.

```
/*************************************************************************

 Name:       : printusage_dvdenc

 Description  : Print the usage message for the program "dvdenc". The
           default values will also be printed, when available.

 Remarks      : 1. This function will not return but exit.

 Author, date : Tri Tran, July 2, 1996.
 Based on program by Peter Westerink.

*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "enumdefs.h"

#define BUFSIZUNIT   16384            /* buffer size unit */

extern int bit_rate;                /* bit rate (bits/sec) */
extern int dvd_rate;                /* maximum DVD bit rate */
extern int npicsGOP;                /* #pictures in a GOP */
extern int m_mpeg;                  /* P period */
extern int chroma_format_coding;       /* chroma format in coder */
extern int code_bs_after_first_i;      /* code B's after 1st I */
extern int rate_control_type;          /* type of rate control */
extern char datnam[];               /* pass 1 data for VBR2 */
extern double globdistlev_pass1;       /* global distortion level */
extern int bufsiz;                  /* VBV buffer size Bmax */
extern enum ActMsr activity_measure;    /* RC perceptual factor */



printusage_dvdenc()

{
 enum FFNFlag GetForceFieldFrameFlag();
 char *DCPrecisionString();

/**** Program description and usage ****/

 printf("\n");
 printf("MPEG-2 encoder for CBR or 2-pass VBR for DVD\n");
 printf("\n");
 printf("Usage: dvdenc codnam orgnam bufnam [options]\n");
 printf("\n");
```

```
/**** Mandatory arguments (no defaults) ****/

  printf("Arguments (no defaults):\n");
  printf("  codnam = base name of result coded bitstream\n");
  printf("  orgnam = base name of original sequence for input\n");
  printf("  bufnam = buffer fullness name\n");  /* this is new to collect data */
  printf("\n");

/**** New or updated options, including the low delay coding options****/

  printf("New and updated options:\n");
  printf("-lowdelay <i>  = low delay coding with intraslice size\n");
  printf("  <i> = positive integer numbers of rows of macroblocks\n");
  printf("-lowdelay2 <i> = low delay coding with intracolumn size\n");
  printf("  <i> = positive integer numbers of columns of macroblocks\n");
  printf(" -skpopt <i>        = (for low delay coding only!) numbers of frames being skipped from intraslice/
col coding\n");
  printf(" -constrained_me <i> = (for low delay coding only!) activate constrained motion estimation with
integer size of rows of macroblocks\n");
  printf("-frm0 <i>     = first frame number (default: first available)\n");
  printf("-nfrms <i>    = number of frames (default: all available)\n");
  printf("-bitrat <i>   = average bit rate in bits/sec (default: %d)\n",bit_rate);
  printf("-dvdrat <i>   = maximum DVD rate in bits/sec (default: %d)\n",dvd_rate);
  printf("-ratcon <i>   = type of rate control (default: %d)\n",
      rate_control_type);
  printf("                %d = constant bit rate (CBR)\n",CBR);
  printf("                %d = variable bit rate (VBR) pass 1 (fixed distortion)\n",
      VBR1);
  printf("                %d = variable bit rate (VBR) pass 2\n",VBR2);
  printf("                %d = old method for constant bit rate (CBR)\n",OLDCBR);
  printf("-fixdis <f>   = fixed distortion level for VBR1 (default: %.1f)\n",globdistlev_pass1);
  printf("-p1data <name> = name of first pass data for VBR2 (no extension)\n");
  printf("-savrec <name> = save reconstruction\n");
  printf("-premvs <name> = use pre-computed motion vectors\n");
  printf("-gopsiz <i>   = #pictures in a GOP, n_mpeg (default: %d)\n",
      npicsGOP);
  printf("-bufsiz <i>     = VBV buffer size in units of %d (default: %d)\n",BUFSIZUNIT,bufsiz/BUFSI-
ZUNIT);
  printf("-activm <i>   = activity measure, rate control perceptual factor (default: %d)\n",activity_measure);
  printf("                %d = pseudo variance (old activity measure)\n",OLD);
  printf("                %d = DCT based\n",DCT);
  printf("                %d = Hadamard transform based\n",HADAMARD);
  printf("                %d = none (perceptual factor fixed to 1)\n",ACTMSR_NONE);
  printf("                %d = IBM chip\n",ACTMSR_CHIP);
  printf("-mbcodm <i>   = force a certain macroblock coding method (default: %d)\n",GetForceFieldFrame-
Flag());
  printf("                %d = force ONLY frame-based coding to be used\n",FRM);
  printf("                %d = force ONLY field-based coding to be used\n",FLD);
  printf("                %d = allow both methods (adaptive selection)\n",NEITHER);
  printf("-norep1f      = disable repeat first field detection\n");
  printf("-noscdet      = disable scene change detection\n");
  printf("-nombrc       = disable macroblock level rate control\n");
  printf("-nobufb         = disable enforcment of VBV buffer bounds (disallow stuffing and catastrophic
mode)\n");
  printf("-gop1full <i> = first GOP full size (1) or not (0) (default: %d)\n",code_bs_after_first_i);
  printf("-DCprec <i>   = set DC bit precision (default: %s)\n",
      DCPrecisionString());
  printf("\n");
```

```
/**** Old existing options ****/

  printf("Old unchanged options:\n");
  printf("-CF <i> = coding YUV format (default: %d)\n",
      chroma_format_coding);
  printf("-PP <i> = set P period, is #B's plus 1 (default: %d)\n",
      m_mpeg);
  printf("-i <i>  = method for intra/inter decision (default: 0)\n");
  printf("  0 = use error, not bits\n");
  printf("  1 = use bits with exact bit count for DC\n");
  printf("  2 = use bits with bit count for DC set to 0\n");
  printf("  3 = use bits with bit count for DC average for last I\n");
  printf("-al = use linear activity transform\n");
  printf("-alo <off> = set linear activity transform offset to off\n");
  printf("-als <sca> = set linear activity transform scale to sca\n");
  printf("\n");

#if 0
  printf("-aa <i> = DCT or HAD activity measure mask (default: %d)\n",
      MaskNum);
  printf("  mask 0:\n");
  printactivmask(0);
  printf("  mask 1:\n");
  printactivmask(1);
  printf("-am <max> = maximum perceptual mquant = <max>\n");
  printf("-an use non linear activity transform\n");
  printf("-ans <sc> = set non linear activity transform scale to sc\n");
  printf("-ap <n> = use method n for had filter precision\n");
  printf("-at <n> = use method n for had filter type to nn\n");
  printf("    0-> After field/frame, 1-> field, 2-> min 3->frame\n");
  printf("-CG Use closed GOPs\n" );
  printf("-CM use concealment motion vectors\n" );
  printf("-CW <type> <wx> <wy> = Weighting <type> in coarse search with weights wx and wy\n" );
  printf("-c = catasrophic error testing\n");
  printf("-DF  = Refine frame vector for dual' candidate\n");
  printf("-Dr <type>  = Set dual refines: 1 -> 1 field\n");
  printf("-dd = disable dual motion\n");
  printf("-dr = disable frame motion\n");
  printf("-di = disable field motion\n");
  printf("-dR [g] = disable CBR control and do DVD VBR:\n");
  printf("  g = global distortion level for DVD pass 1\n");
  printf("  if g is omitted we run DVD pass 2\n");
  printf("-EO = erase original after loading\n");
  printf("-E <n> <x> <y> <type> = Introduce error type in frame n at location x,y \n");
  printf("   0->Macroblock type\n");
  printf("   1->f_codes in pic header (use x,y=0)\n");
  printf("-Fb = Do all 4 field refines each way in B frame\n");
  printf("-Fp = Only 2 field refines in P frame\n");
  printf("-FIP <type>: type = 0 -> No field IPs,  1 -> use field IPs\n" );
  printf("-FC <type>: format conversion <type> \n" );
  printf("          t -> Test Model\n" );
  printf("          s -> simple correct phase two-tap\n" );
  printf("          0 -> chip model 0 (discard botom field for interlaced, av for progressive\n" );
  printf("          1 -> chip model 1 av for progressive or interlaced\n" );
  printf("-FR <code> Set frame rate: code: \n");
  printf("1 -> 23.976, 2 -> 24, 3 -> 25, 4 -> 29.97, 5 -> 30, 6 -> 59.94, 7 -> 60\n");
  printf("-if <flag> = if (flag) do intra/inter decision after field/frame decision\n" );
  printf("-HM <h_sub> <v_sub> <h_ref> <v_ref> Hierarchical\n");
  printf("search subsampled <h_sub> X <v_sub>\n");
```

```
  printf("refined <h_ref> X  <v_ref>\n");
  printf("-IE <x> = Set intra error multiplier for intra/inter decision to x\n" );
  printf("-Ie <type> = Use interpolation <type> prediction with scale <scale> in coding\n" );
  printf("-Is = Get stats on interpolation prediction \n" );
  printf("-L<nwe> = Luminance mask: none, weber, elliot \n" );
  printf("-m <i> <f> = bit estimate method, with parameter <param> for mode determination \n");
  printf("-mvb <type> = Use bit estimate method <type> to estimate bits for mvs\n");
  printf("-M <flag> = Set quantisation_scale_code law (flag=0 -> LINEAR, flag=1 -> NONLINEAR)\n");
  printf("-NBa <alpha> = Set parameter alpha for new buffer control equal to <alpha>\n");
  printf("-NBA <A> = Set parameter A for new buffer control equal to <A>\n");
  printf("-NBD <order> <q> = use <order> Taylor expansion of (D/d)^<q> \n");
  printf("-NBE <ex_a> <ex_b> = Force to exact number of bits with params ex_a and ex_b\n");
  printf("-NBK1 <k1> = Set parameter k1 for new buffer control equal to <k1>\n");
  printf("-NBK2 <k2> = Set parameter k2 for new buffer control equal to <k2>\n");
  printf("-NBKM Treat INTRA macroblocks as I pic data as far as K1 goes\n");
  printf("-NBV0 <c> ideal buffer fullness b/4 I pic is c * buffer size\n");
  printf("-NM Use new motion algorithm \n");
  printf("-O <frac> Use frac to calculate offset for intra quantization\n");
  printf("-OZ Zero out dct coefficients based on motion comp from orig pictures");
  printf("-P <flag> = Set progressive (flag=0 -> progressive, flag=1 -> interlaced)\n");
  printf("-PS Field Structured Pictures\n" );
  printf("-q = use test model quantization (with rounding)\n");
  printf("-Q1 = use all one quant matrices\n");
  printf("-Q2 = use all twos quant matrices\n");
  printf("-Q4 = use all fours quant matrices\n");
  printf("-Qc = use separate luma and chrom studio matrices\n");
  printf("-Qj = use JPEG quant matrix\n");
  printf("-Qh = use Heidi's quant matrix\n");
  printf("-Qi = Studio luma/chroma intra matrix\n");
  printf("-Ql = use separate luma and chrom low bit rate studio matrices\n");
  printf("-Qs = use studio q matrices, (same luma/chroma)\n");
  printf("-Qt = use TM inter matrix only\n");
  printf("-Qy = Studio luma/chroma matrices\n");
  printf("-RF <param> <x> = Change repeat first field <param> to <x>\n");
  printf("      0   0->No detection, 1->detection\n");
  printf("      1   0->IIR filter, 1->diffs above threshold\n");
  printf("      2   0->chech after miss, 1->check each possible match\n");
  printf("      3   Horizontal filter weight\n");
  printf("      4   Vertical filter weight\n");
  printf("      5   Difference threshold\n");
  printf("      6   alpha\n");
  printf("      7   beta\n");
  printf("-RR  = Use recon data for reference in coarse ME pass. \n");
  printf("-RH  = Reduce half pel search to integer search window\n");
  printf("-SA scale previous activity by interpixel diff ratio \n");
  printf("-SP <f> = Print macroblock stats for frame <f> ( -1 -> all frames)\n");
  printf("-SQ Use only a sub-set of quantiser scales\n");
  printf("-SR <px> <py> <b1x> <b1y> <b2x> <b2y> = Set frame search range \n");
  printf("-SS <name> Store pic by pic stats to <name> \n");
  printf("-ST studio optimization\n");
  printf("-s <flag> = Set scan pattern (flag=0 -> ZIGZAG, flag=1 -> ALT_SCAN)\n");
  printf("-T <flag> = Set top last (flag=0 -> top first, flag=1 -> top last)\n");
  printf("-V1 <av> <pre_flds> <pos_flds> <pre_complexity>  = One pass variable bit rate\n");
  printf("-V2 <av> <update_speed> <underflow_safty> <min_complexity> <statfile_name>= Second  pass
variable bit rate\n");
  printf("-v <flag> = Set intra VLC    (flag=0 -> MPEG_1, flag=1 -> ALT_VLC)\n");
  printf("-WO <flag> = Write original after 3:2 inversion if <flag> != 0. \n");
#endif
```

```
/**** Exit ****/

 exit(EXIT_FAILURE);
}
```

COMPLETE FILE OF PARSCOMLIN_DVDENC.C IS INCLUDED.

```
/************************************************************************

 Name:      : parscomlin_dvdenc

 Description  : Parse the command line for the main program "dvdenc".

 Author, date: Tri Tran, July 10, 1996.
 Based on program by Peter Westerink.

************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "enumdefs.h"

#define FUNCNAM     "parscomlin_dvdenc" /* function name */
#define MAXOPTARGC  32              /* maximum #option argum. */
#define BUFSIZUNIT  16384           /* buffer size unit */

/* mandatory arguments */
extern char codnam[];               /* result coded bitstream */
extern char orgnam[];               /* source original sequence */
extern char bufnam[];

/* parameters derived from directory */
extern int ncols, nrows;            /* frame dimensions */
extern int chroma_format_input;     /* chroma format of orginal */
extern int frm0;                    /* first frame */
extern int nfrms;                   /* number of frames */

/* new variables */
extern int lowdelay;                /* brand-new flag for low delay intraslice */
extern int lowdelay2;               /* lowdelay intracol coding */
extern int skpopt;                  /* for lowdelay coding only: skip frame option */
extern int constrain_search_range_size;  /* for lowdelay: size of search range in rows of macroblocks */

extern double frm_rate;             /* frame rate (frames/sec) */
extern int bit_rate;                /* ave. bit rate (bits/sec) */
extern int dvd_rate;                /* maximum DVD bit rate */
extern int rate_control_type;       /* type of rate control */
extern double globdistlev_pass1;    /* global distortion level */
extern int save_recon;              /* save reconstruction ? */
extern char recnam[];               /* reconstruction name */
extern char datnam[];               /* pass 1 data for VBR2 */
extern int useprecompmvs;           /* use precomputed MVs */
```

```
extern char mvsnam[];                /* motion vectors file name */
extern int domacblkbufcon;             /* buf control on MB level? */
extern int bufsiz;                  /* VBV buffer size Bmax */
extern enum ActMsr activity_measure;    /* RC perceptual factor */

/* variables that go into a picture structure (dummy_image) ****/
extern int npicsGOP;                /* I period */
extern int m_mpeg;                  /* P period */
extern int chroma_format_coding;        /* chroma format in coder */
extern int code_bs_after_first_i;       /* code B's after 1st I */
extern int concealment_motion_vectors;  /* use ... */
extern int picture_structure;           /* picture structure */
extern int temporal_order;              /* top or bottom first */
extern int non_interlaced_sequence;     /* progressive source? */
extern int picture_format;              /* picture format */

/* others */
extern int enforce_buff;              /* enforce buffer bounds */
extern int write_orig;                /* useful to save 24Hz */
extern int manual_srange_frm;           /* set search range */



void parscomlin_dvdenc( int argc, char *argv[] )

{
 int optargc;                    /* option argument count */
 char *optargv[MAXOPTARGC];           /* option arguments  */

 void checkoptargc();
 double PicRateDouble();
 void PrintStats();

 /**** If mandatory arguments aren't there, print usage message ****/

 if ((optargc = getnextoption(optargv,argc,argv)) != 4) {
  printusage_dvdenc();
  exit(EXIT_FAILURE);
 }

/**** Get mandatory arguments ****/

 strcpy(codnam,argv[1]);
 strcpy(orgnam,argv[2]);
 strcpy(bufnam,argv[3]);

/**** Guess image format ****/

 if (infovidseq(orgnam,&ncols,&nrows,&frm0,&nfrms,
          &chroma_format_input) == FALSE) {
  fprintf(stderr,"Original sequence %s not found\n",orgnam);
  exit(EXIT_FAILURE);
 }

/**** Get optional arguments ****/

 while ((optargc = getnextoption(optargv,argc,argv)) > 0) {

  /**** New or updated options ****/
```

```
if (strcmp("frm0",optargv[0]) == 0) {        /* new */
  checkoptargc(optargc,optargv,2);
  frm0 = atoi(optargv[1]);
 }

 else if (strcmp("lowdelay",optargv[0]) == 0) {     /* brand-new low delay */
  checkoptargc(optargc,optargv,2);
  lowdelay = atoi(optargv[1]);
 }
 else if (strcmp("lowdelay2",optargv[0]) == 0) {
  checkoptargc(optargc,optargv,2);
  lowdelay2 = atoi(optargv[1]);
 }

 else if (strcmp("skpopt",optargv[0]) == 0) {
  checkoptargc(optargc,optargv,2);
  skpopt = atoi(optargv[1]);
 }

 else if (strcmp("constrained_me",optargv[0]) == 0) {
  checkoptargc(optargc,optargv,2);
  constrain_search_range_size  = atoi(optargv[1]);
 }


 else if (strcmp("nfrms",optargv[0]) == 0) {   /* new */
  checkoptargc(optargc,optargv,2);
  nfrms = atoi(optargv[1]);
 }
 else if (strcmp("savrec",optargv[0]) == 0) {  /* new */
  checkoptargc(optargc,optargv,2);
  save_recon = 1;
  strcpy(recnam,optargv[1]);
 }
 else if (strcmp("ratcon",optargv[0]) == 0) {  /* new */
  checkoptargc(optargc,optargv,2);
  rate_control_type = atoi(optargv[1]);
 }
 else if (strcmp("fixdis",optargv[0]) == 0) {  /* new */
  checkoptargc(optargc,optargv,2);
  globdistlev_pass1 = atof(optargv[1]);
 }
 else if (strcmp("p1data",optargv[0]) == 0) {  /* new */
  checkoptargc(optargc,optargv,2);
  strcpy(datnam,optargv[1]);
 }
 else if (strcmp("bitrat",optargv[0]) == 0) {  /* old -B */
  checkoptargc(optargc,optargv,2);
  bit_rate = atoi(optargv[1]);
 }
 else if (strcmp("dvdrat",optargv[0]) == 0) {  /* new */
  checkoptargc(optargc,optargv,2);
  dvd_rate = atoi(optargv[1]);
 }
 else if (strcmp("gopsiz",optargv[0]) == 0) {  /* old -IP */
  checkoptargc(optargc,optargv,2);
  npicsGOP = atoi(optargv[1]);              /* old n_mpeg */
 }
```

```
else if (strcmp("premvs",optargv[0]) == 0) {  /* old -P2 */
 checkoptargc(optargc,optargv,2);
 useprecompmvs = TRUE;
 strcpy(mvsnam,optargv[1]);
}
else if (strcmp("bufsiz",optargv[0]) == 0) {  /* old -Sb */
 checkoptargc(optargc,optargv,2);
 bufsiz = BUFSIZUNIT * atoi(optargv[1]);
 SetBufSize( atoi(optargv[1]) );          /* future: extern */
}
else if (strcmp("activm",optargv[0]) == 0) {  /* old -a */
 checkoptargc(optargc,optargv,2);
 activity_measure = atoi(optargv[1]);
}
else if (strcmp("mbcodm",optargv[0]) == 0) {  /* old -Rm and -Rl */
 checkoptargc(optargc,optargv,2);
 SetForceFieldFrameFlag( atoi(optargv[1]) );
}
else if (strcmp("nombrc",optargv[0]) == 0) {  /* new */
 checkoptargc(optargc,optargv,1);
 domacblkbufcon = 0;
}
else if (strcmp("norep1f",optargv[0]) == 0) {  /* old -RF 0 0 */
 checkoptargc(optargc,optargv,1);
 ChangeRFFAlg( 0, 0.0 );
}
else if (strcmp("noscdet",optargv[0]) == 0) {  /* old -SC */
 checkoptargc(optargc,optargv,1);
 DisableSceneChangeDetection();
}
else if (strcmp("nobufb",optargv[0]) == 0) {  /* old -db */
 checkoptargc(optargc,optargv,1);
 enforce_buff = 0;
}
else if (strcmp("gop1full",optargv[0]) == 0) {
 checkoptargc(optargc,optargv,2);
 code_bs_after_first_i = atoi(optargv[1]);
}
else if (strcmp("DCprec",optargv[0]) == 0) {      /* old -p */
 checkoptargc(optargc,optargv,2);
 switch (atoi(optargv[1])) {
 case  8:
   SetGlobalDCPrecision( EIGHT );
   break;
 case  9:
   SetGlobalDCPrecision( NINE  );
   break;
 case 10:
   SetGlobalDCPrecision( TEN   );
   break;
 case 11:
   SetGlobalDCPrecision( ELEVEN);
   break;
 default:
   fprintf(stderr,"%s: DC precision must 8, 9, 10, or 11\n",
        FUNCNAM,atoi(optargv[1]));
   exit(EXIT_FAILURE);
 }
}
```

```
/**** Old options with new extern variables ****/

else if (strcmp("CF",optargv[0]) == 0) {
  chroma_format_coding = atoi(optargv[1]);
}
else if (strcmp("CM",optargv[0]) == 0) {
  concealment_motion_vectors = 1;
}
else if (strcmp("PS",optargv[0]) == 0) {
  picture_structure = TOP_FLD;
}
else if (strcmp("T",optargv[0]) == 0) {
  temporal_order = atoi(optargv[1]) ? TOP_LAST : TOP_FIRST;
}
else if (strcmp("WO",optargv[0]) == 0) {
  write_orig = atoi(optargv[1]);
}
else if (strcmp("PP",optargv[0]) == 0) {
  m_mpeg = atoi(optargv[1]);
}
else if (strcmp("P",optargv[0]) == 0) {
  non_interlaced_sequence = !atoi(optargv[1]);
}

/**** Old unchanged options ****/

else if (strcmp("aa",optargv[0]) == 0) {
  SetMaskNum( atoi(optargv[1]) );
}
else if (strcmp("al",optargv[0]) == 0) {
  SetActivityTransform(LIN);
}
else if (strcmp("alo",optargv[0]) == 0) {
  SetActivityMsrOffset( atof(optargv[1]) );
}
else if (strcmp("als",optargv[0]) == 0) {
  SetActivityMsrScaleLinear( atof(optargv[1]) );
}
else if (strcmp("am",optargv[0]) == 0) {
  SetActivityMsrCeiling( atof(optargv[1]) );
}
else if (strcmp("an",optargv[0]) == 0) {
  SetActivityTransform(NONLIN);
}
else if (strcmp("ans",optargv[0]) == 0) {
  SetActivityMsrScaleNonLinear( atof(optargv[1]) );
}
else if (strcmp("ap",optargv[0]) == 0) {
  SetHadFilterPrecision( atoi(optargv[1]) );
}
else if (strcmp("at",optargv[0]) == 0) {
  SetHadFilterType( atoi(optargv[1]) );
}
else if (strcmp("CG",optargv[0]) == 0) {
  UseClosedGops();
}
else if (strcmp("CW",optargv[0]) == 0) {
  SetWeightCoarse( atoi(optargv[1]), atof(optargv[2]),
```

```
            atof(optargv[3]) );
}
else if (strcmp("c",optargv[0]) == 0) {
 TestCatastrophic();
}
else if (strcmp("DF",optargv[0]) == 0) {
 SetDualRefineFrame();
}
else if (strcmp("Dr",optargv[0]) == 0) {
 SetDualRefines( atoi(optargv[1]) );
}
else if (strcmp("dd",optargv[0]) == 0) {
 DontUseMotionMode( DUAL );
}
else if (strcmp("di",optargv[0]) == 0) {
 DontUseMotionMode( FIELD );
}
else if (strcmp("dr",optargv[0]) == 0) {
 DontUseMotionMode( FRAME );
}
else if (strcmp("E",optargv[0]) == 0) {
 IntroduceError( atoi(optargv[1]), atoi(optargv[2]),
            atoi(optargv[3]), atoi(optargv[4]) );
}
else if (strcmp("FC",optargv[0]) == 0) {
 SetConversionMethod( atoi(optargv[1]) );
}
else if (strcmp("FIP",optargv[0]) == 0) {
 Do_IP( atoi(optargv[1]) );
}
else if (strcmp("FR",optargv[0]) == 0) {
 SetPicRate( atoi(optargv[1]) );
 frm_rate = PicRateDouble();
}
else if (strcmp("Fb",optargv[0]) == 0) {
 ChangeFieldAlg(3);
}
else if (strcmp("Fp",optargv[0]) == 0) {
 ChangeFieldAlg(2);
}
else if (strcmp("HM",optargv[0]) == 0) {
 UseHierarchical( atoi(optargv[1]), atoi(optargv[2]),
            atoi(optargv[3]), atoi(optargv[4]) );
}
else if (strcmp("IE",optargv[0]) == 0) {
 SetIntraErrorMultiplier( atof(optargv[1]) );
}
else if (strcmp("IR",optargv[0]) == 0) {
 UseIntraRefresh( atoi(optargv[1]) ,
            ('v' == optargv[2][0]) ? VERTICAL : HORIZONTAL);
 m_mpeg = 1;
}
else if (strcmp("Ie",optargv[0]) == 0) {
 EstimateInterpolation(atoi(optargv[1]),atof(optargv[2]));
}
else if (strcmp("Is",optargv[0]) == 0) {
 DoInterpStats();
}
else if (strcmp("i",optargv[0]) == 0) {
```

```
  UseBitEstimatesForIntraInterDetermination( atoi(optargv[1]) );
}
else if (strcmp("if",optargv[0]) == 0) {
  IntraInterAfterFieldFrame( atoi(optargv[1]) );
}
else if (strcmp("L",optargv[0]) == 0) {
  SetLuminanceMask( atoi(optargv[1]) );
}
else if (strcmp("M",optargv[0]) == 0) {
  SetGlobalQScaleType( atoi(optargv[1]) );
}
else if (strcmp("m",optargv[0]) == 0) {
  UseBitEstimatesForModeDetermination(atoi(optargv[1]),
                                      atof(optargv[2]));
}
else if (strcmp("mvb",optargv[0]) == 0) {
  SetMVBitsCalcMethod( atoi(optargv[1]) );
}
else if (strcmp("NBA",optargv[0]) == 0) {
  SetA( atof(optargv[1]) );
}
else if (strcmp("NBD",optargv[0]) == 0) {
  SetDivide( atoi(optargv[1]), atof(optargv[2]) );
}
else if (strcmp("NBK1",optargv[0]) == 0) {
  SetK1( atof(optargv[1]) );
}
else if (strcmp("NBK2",optargv[0]) == 0) {
  SetK2( atof(optargv[1]) );
}
else if (strcmp("NBKM",optargv[0]) == 0) {
  DoIntraInterG();
}
else if (strcmp("NBR",optargv[0]) == 0) {
  SetR( atof(optargv[1]) );
}
else if (strcmp("NBV0",optargv[0]) == 0) {
  SetV0( atof(optargv[1]) );
}
else if (strcmp("NBa",optargv[0]) == 0) {
  SetAlpha( atof(optargv[1]) );
}
else if (strcmp("O",optargv[0]) == 0) {
  SetOffsetFrac( atof(optargv[1]) );
}
else if (strcmp("OZ",optargv[0]) == 0) {
  SetOrigZero(1);
}
else if (strcmp("Q",optargv[0]) == 0) {
  SetQtables( atoi(optargv[1]) );
}
else if (strcmp("q",optargv[0]) == 0) {
  SetQmethod( TM_ROUND );
}
else if (strcmp("RD",optargv[0]) == 0) {
  SetSearchRef(optargv[1][0],('r'==optargv[2][0]) ? RECON : ORIG);
}
else if (strcmp("RF",optargv[0]) == 0) {
  ChangeRFFAlg( atoi(optargv[1]), atof(optargv[2]) );
```

```
  }
  else if (strcmp("RH",optargv[0]) == 0) {
   ReduceHpSearch2IntRange();
  }
  else if (strcmp("RR",optargv[0]) == 0) {
   SetSearchRef( 'c', RECON );
  }
  else if (strcmp("SA",optargv[0]) == 0) {
   SetScalePreviousAct( 1 );
  }
  else if (strcmp("SP",optargv[0]) == 0) {
   PrintStats( atoi(optargv[1]) );
  }
  else if (strcmp("SQ",optargv[0]) == 0) {
   SetQuanitserSubset( 1 );
  }
  else if (strcmp("SR",optargv[0]) == 0) {
   SetSearchRange_Frm( atoi(optargv[1]), atoi(optargv[2]),
                atoi(optargv[3]), atoi(optargv[4]),
                atoi(optargv[5]), atoi(optargv[6]) );
   manual_srange_frm = 1;
  }
  else if (strcmp("SS",optargv[0]) == 0) {
   SetStorePicStats( 1, optargv[1] );
  }
  else if (strcmp("ST",optargv[0]) == 0) {
   DoStudio();
  }
  else if (strcmp("s",optargv[0]) == 0) {
   SetGlobalScanPattern( atoi(optargv[1]) );
  }
  else if (strcmp("V",optargv[0]) == 0) {
   if (atoi(optargv[1]) == 1) {
    OnePassVBR( atof(optargv[2]), atof(optargv[3]),
            atof(optargv[4]), atof(optargv[5]) );
   }
   else {
    TwoPassVBR( atof(optargv[2]), atof(optargv[3]),
            atof(optargv[4]), atof(optargv[5]), optargv[6] );
   }
  }
  else if (strcmp("v",optargv[0]) == 0) {
   SetGlobalIntraVLCTable( atoi(optargv[1]) );
  }

  else {
   fprintf(stderr,"%s: Unknow option -%s\n",FUNCNAM,optargv[0]);
   exit(EXIT_FAILURE);
  }

 }

/**** Set dependent options ****/

 if (rate_control_type == VBR1) {   /* if VBR pass I */
  enforce_buff  = 0;          /*  don't force buffer bounds */
  domacblkbufcon = 0;          /*  no MB level control */
 }
```

```
/**** Check dependent options ****/

 if (rate_control_type == VBR2 && datnam[0] == '\0') {
  fprintf(stderr,"%s: Option -ratcon %d requires option -p1data\n",
       FUNCNAM,VBR2);
  exit(EXIT_FAILURE);
 }

/**** Return nothing ****/

 return;
}
```

/**

COMPLETE FILE OF PRINTPARAMS_DVDENC.C IS INCLUDED.

****

```
 Name:      : printparams_dvdenc

 Description  : Print all parameter settings of the program "dvdenc".

 Author, date : Tri Tran, July 12, 1996.
 Based on program by Peter Westerink.

**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "enumdefs.h"

#define MEGA   1000000              /* one milion */

extern char codnam[];               /* result coded bitstream */
extern char orgnam[];               /* source original sequence */
extern int ncols, nrows;            /* frame dimensions */
extern int chroma_format_input;       /* chroma format of orginal */
extern int frm0;                    /* first frame */
extern int nfrms;                   /* number of frames */
extern int lowdelay;
extern int lowdelay2;
extern int skpopt;
extern int constrain_search_range_size;

extern int save_recon;              /* save reconstruction */
extern char recnam[];               /* reconstruction name */
extern int bit_rate;                /* bit rate (bits/sec) */

extern int npicsGOP;                /* #pictures in a GOP */
extern int m_mpeg;                  /* P period */
extern int rate_control_type;         /* type of rate control */
extern int chroma_format_coding;      /* chroma format in coder */
extern int code_bs_after_first_i;     /* code B's after 1st I */
extern int temporal_order;            /* top or bottom first */
extern int non_interlaced_sequence;   /* progressive source? */
```

```
#if 0
static char *str_onoff[] = {"on", "off"};
#endif



void printparams_dvdenc()

{
 double PicRateDouble();

 printf("\n");
 printf(" Source sequence ---------- %s\n",orgnam);
 printf("  Image dimensions (Hxv) ---- %dx%d\n",ncols,nrows);
 printf("  Chroma format ---------- %d\n",chroma_format_input);
 printf(" First frame to code --------- %d\n",frm0);
 printf(" Number of frames to code ---- %d\n",nfrms);
 printf(" Compressed bit stream file -- %s\n",codnam);
 if (save_recon)
  printf(" Reconstruction name --------- %s\n",recnam);
 else
  printf(" Reconstruction name --------- <not saved>\n");
 printf(" Bit rate -------------------- %.1f Mbits/sec\n",
     (double)bit_rate/(double)MEGA);


 if (lowdelay != 0)
      printf("\n*******  LOW DELAY CODING USING INTRASLICE ACTIVATED! (size %d)
******\n\n",lowdelay);

 if (lowdelay2 != 0)
    printf("\n*******  LOW DELAY CODING USING INTRACOLUMN ACTIVATED! (size %d)
******\n\n",lowdelay2);

 if (skpopt != 0)
      printf("\n     @@@@@  Number of frames being skipped from intraslice/col coding: %d
@@@@@\n\n",skpopt);

 if (constrain_search_range_size != 0)
       printf("\n                                Search range size: %d row(s)/column(s) of macrob-
locks\n\n",constrain_search_range_size);


 if (rate_control_type == CBR)
  printf(" Rate control type ---------- CBR\n");
 else if (rate_control_type == VBR1)
  printf(" Rate control type ---------- VBR pass 1\n");
 else if (rate_control_type == VBR2)
  printf(" Rate control type ---------- VBR pass 2\n");
 printf(" Number of pictures in a GOP - %d\n",npicsGOP);
 printf(" B-pictures between refs ----- %d\n",m_mpeg-1);
 printf(" Picture rate ---------------- %.2f pics/sec\n",
     PicRateDouble());
 if (non_interlaced_sequence)
  printf(" Picture format -------------- Progressive\n");
 else
  printf(" Picture format -------------- Interlaced\n");
```

```
 printf("   Coder chroma format ------- %d\n",chroma_format_coding);

#if 0
 if (chroma_format_input == 422 && chroma_format_coding ==420) {
  if (ConversionMethod == TM_420) {
   printf("   422 -> 420 Conversion ---- Test Model\n");
  }
  else if (ConversionMethod == SIMPLE) {
   printf("   422 -> 420 Conversion ---- Correct phase two-tap\n");
  }
  else if (ConversionMethod == CHIP0) {
   printf("   422 -> 420 Conversion ---- Interlaced: remove bottom field\n");
   printf("                              Progressive: correct phase two-tap\n");
  }
  else if (ConversionMethod == CHIP1) {
   printf("   422 -> 420 Conversion ---- Interlaced: 2-tap progressive filter\n");
   printf("                              Progressive: correct phase two-tap\n");
  }
 }
#endif
#if 0
 PrintConversionMethod( c_image);
#endif
 if ( IsMbRefresh() ) {
  PrintIntraMbRefresh();
 }
 else {
  printf("  (N,M) ----------------------- (%d,%d)\n",
       npicsGOP,m_mpeg);
 }

 if ( IsClosedGops() ) {
  printf("  GOP structure ---------------- Closed\n");
  if (code_bs_after_first_i)
   printf("  First GOP -------------------- Full\n");
  else
   printf("  First GOP -------------------- Partial\n");
 }
 else {
  printf("  GOP structure ---------------- Open\n");
  printf("  VBV buffer size -------------- %d\n",
       16384 * GetVbvBufferSize());
 }

 printf("  DC precision ----------------- %s\n",DCPrecisionString());
 printf("  INTRA VLC TABLE -------------- %s\n",IntraVLCTableString());
 printf("  COEF SCAN PATTERN ------------ %s\n",ScanPatternString());
 printf("  QUANTIZATION MATRIX ---------- %s\n",GetQtablesString());
 PrintQMatrices();
 printf("  INTRA Q OFFSET PARAM --------- %s\n",GetOffsetFracString());
 printf("  COEF QUANTIZATION METHOD ---- %s\n",GetQmethodString());
 if (TOP_FIRST == temporal_order)
  printf("  FIELD ORDER ------------------ top first\n");
 else
  printf("  FIELD ORDER ------------------ top last\n");
 printf("  MQUANT LAW ------------------- %s\n",MquantLawString());

 printf("  FIELD CODING DECISION NORM - ");
 PrintFieldCodingDecisionNorm();
```

```
  PrintRFFAlg();
  printf(" BIT EST/MODE DETERM METHOD -\n");
  PrintBitEstForModeDeterminationStatus();

  PrintOrigZero();

  printf("Motion Estimation Algorithm Settings:\n");
  PrintMotionEstimationAlg();

  if (GetRateControl() == 0)
   printf(" NO RATE CONTROL ----------\n");

  printadapquanpars();
  printf(" CODING MODE FORCED --------- %s\n",
       ForceFieldFrameString());
  if ( DoScalePreviousAct( ) )
   printf(" SCALED ACT ---------- Yes\n");
  else
   printf(" SCALED ACT ---------- No\n");

  if (GetEstimateInterpolation() == 1){
   printf(" USE INTERP PREDICTION ------\n");
   PrintEstInterpStatus();
  }
  PrintIntroducedErrors();
  PrintRC();

  printf("\n==================================================\n\n");

/**** Return nothing ****/

  return;
}
```

```
/*********************************************************************
KEY MODIFICATIONS OF ENC_CODE.C ONLY.
*********************************************************************/
    if (rate_control_type == OLDCBR) {  /* Constant Bit Rate Operation */
      /* perceptual quantization scale p(n,k) and put in mquant */
      mquant = activity_mb(pmb8_orig,c_image,mb_mquant,var);
      /* accumulate (here, because mquant is activity) */
      avg_act += mquant;
      /* modify mquant for a linear or non-linear setting */
      /* default is NONLIN (same method as in ref model 5) */
      /* change to LIN with option -al */
      mquant = transform_activity( mquant, averactivest );
      /* clip mquant at a certain maximum (ceiling) */
      /* ceiling is changed with option -am, default=2.0 */
      PerceptualMquantCeiling( &mquant );
      /* multiply mquant with a measure derived from Y DC terms */
      /* type is set with -L option, default=0=NONE ==> no effect */
      LuminanceMask( c_image, pmb8_orig, &mquant );
      /* rc_scale is fixed in main() to 1, no option exists */
```

```
    mquant *= rc_scale;
    /* select motion vector structure and direction */

/* Checking for low delay coding mode:  Tri Tran; July 8, 1996. */
    if ((lowdelay != 0) || (lowdelay2 != 0)) {

      if (lowdelay != 0) {       /* if option is selected */
        if (Myintraslicefunc(imbrow, imbcol, nmbcols, nmbrows, lowdelay, skpopt)) {    /* check for true or
false */
          stats->motion_type=FRAME; /* Set to be frame instead of field */
          stats->mb_mode=INTRA;    /* code mb as intra mb */
          printf(" Intra slice imbcol = %d\n",imbcol);
          printf("   imbrow = %d\n", imbrow);
        }
        else
          SelectMBMode( c_image, pmb8_orig, imbrow, imbcol, var[0] );
      }

      if (lowdelay2 != 0) {
           if (Myintracolfunc(imbrow, imbcol, nmbcols, nmbrows, lowdelay2, skpopt)) {  /* function is
enc_mec.c */
          stats->motion_type=FRAME;
          stats->mb_mode=INTRA;
          printf(" Intra column imbcol = %d\n",imbcol);
          printf("   imbrow = %d\n", imbrow);
        }
        else
          SelectMBMode( c_image, pmb8_orig, imbrow, imbcol, var[0] );
      }
    }

    else            /* if not the low delay case, then code as normal mb */

      SelectMBMode( c_image, pmb8_orig, imbrow, imbcol, var[0] );
      /* use g(n)=mquant, to return g(n)*b(n,k) */
      mb_bufadjscal = BufferAdjust(c_image,bits_so_far,imbrow,imbcol,
                          mquant, y_sae(pmb8_orig,INTRA));
      /* multiply: p(n,k) * (g(n)*b(n,k)) */
      mquant *= mb_bufadjscal;
   }
```

```
/*********************************************************************
```

KEY MODIFICATIONS FOR ENC_MEC.C ONLY.

```
*********************************************************************
```

Name : Myintraslicefunc

Description : Determine an algorithm to insert intra slices into
P-frames for the purpose of low delay coding. There
is no B-frames and only a single I-frame at the beginning.
Currently, the slice size is selectable multiple integer
of numbers of rows of macroblocks.

Remarks : 1. This function returns TRUE or FALSE value so that
the encoder will know to either code the

```
            current macroblock using intra or not.

  Author, date :  Tri Tran, July-8-1996

  ***************************************************************/

#include <curses.h>         /* Just to make sure T=1, F=0 */

static dorownum = -1; /* =0 if 1st frame has intraslice, =-1 for 2nd, etc...*/

Myintraslicefunc(
int imbrow,
int imbcol,
int nmbcols,
int nmbrows,
int lowdelay,
int skpopt)             /* for option selection usage */

{
  int returnvalue;
  /*   static dorownum = -1;    keeps track of which row is being worked on */
  static nr_frm_to_be_skip = 1;  /* number of frames to be skipped */
  static isframeintracoded = 1;  /* keeps track to see if a frame is being intra coded */

  if ((dorownum >= imbrow)  &&  (dorownum < imbrow + lowdelay)  &&  (nr_frm_to_be_skip ==0)) {  /*
critical range */
    returnvalue = 1;
    isframeintracoded = 1;  /* since this mb is intra coded, set flag */
  }
  else {
    returnvalue = 0;  /*  else don't intra code the current mb */
  }

  if ((nr_frm_to_be_skip == 0) && (imbcol==nmbcols-1) && (imbrow==nmbrows-1)) { /* if end of frame
and number of frame to be skipped is 0 */
    nr_frm_to_be_skip=skpopt+1;          /*  a hack by adding 1 */
  }

  if (imbcol==nmbcols-1 && imbrow==nmbrows-1) {  /* when current col and row count has reached end of
a frame */
    if (isframeintracoded) {
      dorownum += lowdelay;  /* only increment dorownum if the frame has been intra coded */
    }
    nr_frm_to_be_skip--;  /* decrement the numbers left to be skipped */
     isframeintracoded = 0;  /* reset flag so dorownum will not be incremented until the next frame is intra
coded */
  }

  if (dorownum >= nmbrows+lowdelay-1) {      /* for example, dorownum >= 30 rows (end of frame)*/
    printf(" Dorownum has reached %d and is being reseted\n",dorownum);
    dorownum = lowdelay-1;   /* start over at top of frame */
  }

  return(returnvalue);/* return the value assigned */

}

/*********************************************************************
```

```
 Name        : Myintracolfunc

 Description : Determine an algorithm to insert intra columns into
               P-frames for the purpose of low delay coding. There
               is no B-frames and only a single I-frame at the beginning.
               Column size is selectable to be multiple integer numbers
               of columns of macroblocks.

 Remarks     : 1. This function returns TRUE or FALSE value so that
                 the encoder will know to either code the
                 current macroblock using intra or not.

               2. Very similar to Myintraslicefunc. See most
                 documentation there instead.

 Author, date : Tri Tran, July-18-1996

 ****************************************************************/

static docolnum = -1;

Myintracolfunc(   /* very similar to Myintraslicefunc, see comments there */
int imbrow,
int imbcol,
int nmbcols,
int nmbrows,
int lowdelay2,
int skpopt)           /* for option selection usage */

{
 int returnvalue;
 /*  static docolnum = -1;  */
 static nr_frm_to_be_skip = 1;
 static isframeintracoded = 1;

 if ((docolnum >= imbcol) && (docolnum < imbcol+lowdelay2) && (nr_frm_to_be_skip ==0)) {
  returnvalue = 1;
  isframeintracoded = 1;
 }
 else {
  returnvalue = 0;
 }

 if ((nr_frm_to_be_skip == 0) && (imbcol == nmbcols-1) && (imbrow == nmbrows-1)) {
  nr_frm_to_be_skip=skpopt+1;
 }

 if (imbcol==nmbcols-1 && imbrow==nmbrows-1) {
  if (isframeintracoded) {
   docolnum += lowdelay2;
  }
  nr_frm_to_be_skip--;
  isframeintracoded = 0;
 }

 if (docolnum >= nmbcols + lowdelay2 - 1) {
  printf(" Docolnum has reached %d and is being resetted\n", docolnum);
  docolnum = lowdelay2 - 1;
 }
```

```
 return(returnvalue);

}

/*************************************************************
 Function ConstrainSearchRangeInRefreshedRegion

 This function is called to limit the search range to within
 the region recently refreshed by intraslices. The size of
 search range is selectable by user. See MPEG2 standard or
 ISCAS 97 paper for a typical scenario.

 Remarks: 1. Within "don't care" regions, there is no
             constraint.
          2. Constraint region is selectable by variable
             "constrain_search_range_size"

 Author, date: Tri Tran, August 1, 1996

 *************************************************************/

static void ConstrainSearchRangeInRefreshedRegion(  /* Constrained Motion Estimation */
   int width,          /* image width at current resolution  */
   int height,         /* image height at current resolution */
   int block_width,      /* width of search block           */
   int block_height,     /* height of search block          */
   int block_num_x,      /* current horizontal block coord number   */
   int block_num_y,      /* current vertical block coord number     */
   int in_minx, int in_miny, int in_maxx, int in_maxy,
   int *out_minx, int *out_miny, int *out_maxx, int *out_maxy)
{
 int dist_x, dist_y;

 dist_x = block_num_x * block_width;
 dist_y = block_num_y * block_height;

 /* For all the cases below, can use a switch, but I don't know if a switch is
    possible for two variables "lowdelay" and "lowdelay2"  */


 switch(constrain_search_range_size)
  {
  case 0:
    *out_minx = MAX( -block_num_x * block_width, in_minx );
    *out_miny = MAX( -block_num_y * block_height, in_miny );
    *out_maxx = MIN( width - (block_num_x + 1) * block_width , in_maxx );
    *out_maxy = MIN( height - (block_num_y + 1) * block_height, in_maxy );
    break;

  default:
    if ((lowdelay == 0) && (lowdelay2 == 0)) {
      *out_minx = MAX( -block_num_x * block_width, in_minx );
      *out_miny = MAX( -block_num_y * block_height, in_miny );
      *out_maxx = MIN( width - (block_num_x + 1) * block_width , in_maxx );
      *out_maxy = MIN( height - (block_num_y + 1) * block_height, in_maxy );
      printf("Constrained ME not to be used without low delay coding....Ignoring option...\n");
    }
```

```
    if ((lowdelay != 0) && (lowdelay2 != 0)) {
      if ((dist_y >= dorownum*16) || (dist_y < 16*(dorownum-constrain_search_range_size)) ||
         (dist_x >= docolnum*16) || (dist_x < 16*(docolnum-constrain_search_range_size))) {
        *out_minx = MAX( (-block_num_x * block_width)            , in_minx );
        *out_miny = MAX( (-block_num_y * block_height)           , in_miny );
        *out_maxx = MIN( (width - (block_num_x + 1) * block_width)  , in_maxx );
        *out_maxy = MIN( (height - (block_num_y + 1) * block_height), in_maxy );
      }
      else {
        *out_minx = MAX ( MAX(-constrain_search_range_size * 16,-dist_x), in_minx);  /* new */
        *out_miny = MAX ( MAX(-constrain_search_range_size * 16,-dist_y), in_miny);   /* new */
        *out_maxx = MIN ( MIN((docolnum*16 - (block_num_x+1) * block_width),
                    (width - (block_num_x + 1) * block_width)), in_maxx);  /* new */
        *out_maxy = MIN ( MIN((dorownum*16 - (block_num_y+1) * block_height),
                    (height - (block_num_y + 1) * block_height)), in_maxy);  /* new */
      }
    }


    if ((lowdelay != 0) && (lowdelay2 == 0)) {  /* for intraslice lowdelay coding */
      if ((dist_y >= dorownum*16) || (dist_y < 16*(dorownum-constrain_search_range_size))) {  /* if in the
don't care region, */
        /* then values just like MakeSearchRangeInImage() */
        *out_minx = MAX( (-block_num_x * block_width)            , in_minx );
        *out_miny = MAX( (-block_num_y * block_height)           , in_miny );
        *out_maxx = MIN( (width - (block_num_x + 1) * block_width)  , in_maxx );
        *out_maxy = MIN( (height - (block_num_y + 1) * block_height), in_maxy );
      }
      else {
        *out_minx = MAX ( (-block_num_x * block_width)               , in_minx);    /* old */
        *out_miny = MAX ( MAX(-constrain_search_range_size * 16,-dist_y), in_miny);    /* new */
        *out_maxx = MIN ( (width - (block_num_x +1) * block_width)     , in_maxx );   /* old */
        *out_maxy = MIN ( MIN((dorownum*16 - (block_num_y+1) * block_height),
                    (height - (block_num_y + 1) * block_height)), in_maxy);  /* new */
/*       printf("dorownum=%d, dist_x=%d, dist_y=%d, inminx=%d, inminy=%d, inmaxx=%d,
             inmaxy=%d\n",dorownum, dist_x, dist_y, in_minx, in_miny, in_maxx, in_maxy);
               printf("outminx=%d,  outminy=%d,  outmaxx=%d,  outmaxy=%d\n",*out_minx,  *out_miny,
*out_maxx, *out_maxy);
*/
      }
    }


    if ((lowdelay == 0) && (lowdelay2 != 0)) {  /* for intracolumn lowdelay coding */
      if ((dist_x >= docolnum*16) || (dist_x < 16*(docolnum-constrain_search_range_size))) {
        *out_minx = MAX( (-block_num_x * block_width)            , in_minx );
        *out_miny = MAX( (-block_num_y * block_height)           , in_miny );
        *out_maxx = MIN( (width - (block_num_x + 1) * block_width)  , in_maxx );
        *out_maxy = MIN( (height - (block_num_y + 1) * block_height), in_maxy );
        }
      else {
        *out_minx = MAX ( MAX(-constrain_search_range_size * 16,-dist_x), in_minx); /* new */
        *out_miny = MAX ( (-block_num_y * block_height)              , in_miny ); /* old */
        *out_maxx = MIN ( MIN((docolnum*16 - (block_num_x+1) * block_width),
                    (width - (block_num_x + 1) * block_width)), in_maxx);  /* new */
        *out_maxy = MIN ( (height - (block_num_y + 1) * block_height)  , in_maxy ); /* old */
/*       printf("docolnum=%d, blwidth=%d, blheight=%d, blnumx=%d, blnumy=%d,
             dist_x=%d, dist_y=%d, inminx=%d, inminy=%d, inmaxx=%d, inmaxy=%d\n",
             docolnum, block_width, block_height, block_num_x, block_num_y, dist_x,
```

```
            dist_y, in_minx, in_miny, in_maxx, in_maxy);
        printf("outminx=%d, outminy=%d, outmaxx=%d, outmaxy=%d\n\n",*out_minx,
            *out_miny, *out_maxx, *out_maxy);
*/
      }
    }
    break;
  }
}
```

```
/*************************************************************************
```

KEY MODIFICATIONS TO ENC_RC.C ONLY.

```
*************************************************************************/
void AdjustVbvForHeaderBits( Picture * c_image )
{
 FILE *to;
 int hdr_bits = c_image->f_sta.seq_hdr_bits + c_image->f_sta.gop_hdr_bits + 32; /* the 32 is for picture
start code */

 BufferFullnessBits -= hdr_bits;
 BufferFullnessBitsVirtual -= hdr_bits;
#if DEBUG_BUF == 1
 fprintf(stderr,"BUFFUL> subtract %6d header bits ==> %8d\n",
      hdr_bits,BufferFullnessBits);
#endif


 to = fopen(bufnam,"a"); /* new stuff here to record data for fullness */
 fprintf(to,"%8d\n",BufferFullnessBits);
 fclose(to);
}
```

```
/*************************************************************************
```

COMPLETE FILE OF DVDMOTEST.C IS INCLUDED.

```
*************************************************************************/

 Name: dvdmotest

 Description: To compute for motion vectors to be used for future codings (to speed
              up coding speed).

*************************************************************************/

#undef DEBUG_MOTION

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <curses.h>

#include "constants.h"
#include "picture.h"
```

```
#include "format_convert.h"
#include "misc.h"
#include "enc_act.h"
#include "enc_mec.h"

#include "dvdglobvars.h"

/**** This is black ****/

#define INIT_Y    16
#define INIT_CB   128
#define INIT_CR   128



main( int argc, char *argv[] )

{
  Picture *c_image;                 /* current picture pointer */
  Picture *p_image;                 /* previous picture pointer */
  Picture *f_image;                 /* future picture pointer */
  Picture image1, image2, image3;       /* actual picture strucures */
  Picture dummy_image;                  /* structure for parameters */
  int tmp_num;                      /* temp variable (needed) */
  int storsize_y, storsize_c;           /* storage sizes */
  int f, ff, frame_num, first_frm_num, last_frm_num;
  enum PictureStructure ps[2];          /* field/frame structure */

/**** Parse command line ****/

  parscomlin_dvdmotest(argc,argv);
  printparams_dvdmotest();

/**** Initialize picture structures ****/

  c_image = &image1;
  p_image = &image2;
  f_image = &image3;
  Picture_Initialize( &dummy_image );
  Picture_Initialize( c_image );
  Picture_Initialize( p_image );
  Picture_Initialize( f_image );

/**** For now, copy new parameters into old ones ****/

  strcpy(dummy_image.input_name,orgnam);
  dummy_image.width                 = ncols;
  dummy_image.height                = nrows;
  dummy_image.chroma_format_input       = chroma_format_input;
  dummy_image.chroma_format_coding      = chroma_format_coding;
  dummy_image.n_mpeg                  = npicsGOP;
  dummy_image.m_mpeg                   = m_mpeg;
  dummy_image.concealment_motion_vectors = concealment_motion_vectors;
  dummy_image.picture_structure         = picture_structure;
  dummy_image.temporal_order            = temporal_order;
  dummy_image.non_interlaced_sequence    = non_interlaced_sequence;
  dummy_image.picture_format            = picture_format;
  if (save_recon)
```

```
  strcpy(dummy_image.recon_name,recnam);

 first_frm_num = frm0;
 last_frm_num  = frm0 + nfrms - 1;

/**** Set Motion Algorithm default values ****/

 InitMotionAlgorithm();

/**** Continue with what was there ****/

 if ( !manual_srange_frm )
  DefaultSearchRange_Frm( &dummy_image );

 m_mpeg = dummy_image.m_mpeg;

 if (dummy_image.picture_structure == FRM_PIC) {
  ps[0] = FRM_PIC;
  ps[1] = FRM_PIC;
 }
 else {
  if (dummy_image.temporal_order == TOP_FIRST) {
   ps[0] = TOP_FLD;
   ps[1] = BOT_FLD;
  }
  else {
   ps[0] = BOT_FLD;
   ps[1] = TOP_FLD;
  }
 }

/**** Copy parameters from dummy_image to c_image, and frm0 ****/

 Picture_CopyAttributes( &dummy_image, c_image );
 Picture_SetFrameNumber( c_image, first_frm_num );

/**** Print motion estimation parameters ****/

 PrintMotionEstimationAlg();

/**** Load very first I-picture, or blank it ****/

 Picture_SetType( c_image, I );      /* does this matter? */

 if (code_bs_after_first_i == 0) {
  frame_num = frm0;
  Picture_Load( c_image );
 }

 else {
  frame_num = frm0 - 1;
  Picture_Realloc(c_image);
  storsize_y = c_image->width * c_image->height;
  storsize_c = storsize_y / 2;
  memset(Picture_Yptr(c_image),INIT_Y,storsize_y);
  memset(Picture_Cbptr(c_image),INIT_CB,storsize_c);
  memset(Picture_Crptr(c_image),INIT_CR,storsize_c);
 }
```

```
  HierarchicalDecimation( c_image, ORIG );

  if (ps[0] != FRM_PIC) {
   Picture_SetType( c_image, IP );
   c_image->picture_structure = ps[1];
   Picture_MotionEstimate( c_image, NULL );
  }
  p_image = c_image;
  c_image = &image2;

/**** Main processing loop ****/

  frame_num = frame_num + m_mpeg;
  while ( frame_num <= last_frm_num ) {

   /**** Reference picture (I or P) ****/

   Picture_CopyAttributes( &dummy_image, c_image );
   Picture_SetType( c_image, P );      /* always P: not relevant? */
   Picture_SetPrev( c_image, p_image );
   Picture_SetFrameNumber( c_image, frame_num );

   Picture_Load( c_image );
   HierarchicalDecimation( c_image, ORIG );

   c_image->picture_structure = ps[0];
   Picture_MotionEstimate( c_image, NULL );
   if (ps[0] != FRM_PIC) {
    c_image->picture_structure = ps[1];
    Picture_MotionEstimate( c_image, NULL );
   }

   /**** Future reconstruction becomes current picture ****/

   f_image = c_image;

   /**** Loop over the B-pictures between reference pictures ****/

   tmp_num = frame_num;
   for (f = tmp_num-m_mpeg+1; f < tmp_num; f++ ) {

    frame_num = f;

    c_image = &image3;                    /* Could be done */
    Picture_CopyAttributes(&dummy_image,c_image); /* outside loop */

    Picture_SetType( c_image, B );
    Picture_SetFuture( c_image, f_image );
    Picture_SetPrev( c_image, p_image );
    Picture_SetFrameNumber( c_image, frame_num );

    Picture_Load( c_image );
    HierarchicalDecimation( c_image, ORIG );

    c_image->picture_structure = ps[0];
    Picture_MotionEstimate( c_image, NULL );
    if (ps[0] != FRM_PIC) {
     c_image->picture_structure = ps[1];
     Picture_MotionEstimate( c_image, NULL );
```

```
    }

   }

   /**** Current picture becomes past, and past becomes future ****/

   c_image = p_image;
   p_image = f_image;

   /**** Update frame numbering ****/

   if (m_mpeg != 1)
    frame_num++;
   frame_num += m_mpeg;
  }

/**** Exit ****/

 exit(EXIT_SUCCESS);
}
```

Appendix B

The Random Bit Error Injector

```
/*************************************************************
 Name:      injerr

 Description: to introduce error into a bitstream sequence. Need
          mean error rate and burst length. Probability is
          calculated as in Test Model 5. Errors represent noise
          errors where an input bit is toggled.

 Author, date: Tri Tran, August 27, 1996

 *************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define BUFSIZE 1


main(argc, argv)
int argc;
char **argv;
{
 int n=0,integervalue;  /* counter, integervalue for format conversion */
 FILE *from, *to;      /* file handlers for input and output files   */
 long float error_rate;     /* mean error rate                        */
 long float burst_length;   /* burst length                           */
 unsigned char buf;    /* input buffer                          */
 long float drand48(void); /* random number generator                 */
 int PrevBitErr=0, BitErr=0; /* flags */
 long float PL, PN;  /*  Probability of bit error given prev is toggled or not */
 int index;/* for loop counter, also to index bit of MASK              */
 int MASK; /* This is to create the error byte to be XOR'ed with data byte */

 if (argc !=5) {
  printf("\nIntroduce noise errors into a bitstream file\n\n");
  write(2, "Usage: ",7);
  write(2, *argv, strlen(*argv));
  write(2," bitstream_file result_file mean_error_rate burst_length\n\n", 60);
  printf("     bitstream_file :  input encoded video sequence\n");
  printf("     result_file    :  output error injected bitstream file\n");
  printf("     mean_error_rate:  mean error occuring rate (i.e. 0.0001)\n");
  printf("     burst_length   :  how long (integer) is burst length (i.e. 5)\n\n");
  exit(1);
 }
```

```
/**** Obtaining argument parameters ****/

error_rate=atof(argv[3]);    /* mean error rate */
burst_length=atof(argv[4]); /* mean burst length    */

/**** Printing out some chosen parameters ****/

printf("Bitstream file: %s\n",argv[1]);
printf("Resulting file: %s\n",argv[2]);
printf("Error rate = %2.10f\n",error_rate);
printf("Burst length = %2.10f\n",burst_length);
printf("Sizeofbuf = %d\n",sizeof(buf));


/**** Open/Create input bitstream and resulting files ****/

if ((from = fopen(argv[1], "r")) == NULL) {
 printf("Error opening %s\n",argv[1]);
 perror(argv[1]);
 exit(1);
}

if ((to = fopen(argv[2], "a")) == NULL) {
 printf("Error opening/creating %s\n",argv[2]);
 perror (argv[2]);
 exit(1);
}

/*****  Processing Plant  *****/

PL = 1-1/burst_length;                    /* key equation */
PN = error_rate/(burst_length * (1 - error_rate)); /* key equation */
printf("PL is %2.10f\n",PL);
printf("PN is %2.10f\n",PN);

/* PL is probability that the current bit is toggled given that the previous
  bit is toggled.

  PN is probability that the current bit is toggled given that the previous
  bit is not toggled.

  */
PrevBitErr = 0; /* set flag indicating that previous bit is not an error */
while ((n = fread(&buf, 1, sizeof(buf), from)) > 0) {  /* while not bitstream EOF */

 integervalue=buf;  /* format conversion */

 MASK = 0;  /* reset value of MASK */

 for (index=0;index<(8*sizeof(buf));index++) {  /* looping through the bits of MASK */

  switch (PrevBitErr) {   /* equality dependent on if prev bit is toggled or not */
  case 1:
   if (drand48() < PL) {
```

```
        BitErr = 1;
       }
       else {
        BitErr = 0;
       }
       break;
     case 0:
       if (drand48() < PN) {
        BitErr = 1;
       }
       else {
        BitErr = 0;
       }
       break;
     }  /*  END of switch  */

     if (BitErr) {      /* if a bit is determined to be toggled, then add it to MASK */
       MASK += (1<<index);  /* shift and add */
     }

     PrevBitErr = BitErr;  /* new assignment for PrevBitErr */

   }    /*  END of for loop. Finished preparing a MASK number  */

   integervalue ^= MASK;  /* XOR operation to toggle input bits.  This is done once for every buf size at a
time */

   buf=integervalue;       /* convert back to char */

   fwrite(&buf, 1, n, to);  /*  write back buf to result file */

 }  /*  END while loop.  Done  */

 printf("Found end of input file\n");  /* reached eof of bitstream */

 /* closing both input and output files and exit */

 fclose(from);
 fclose(to);
 exit(0);

}
```

References

[1] ISO/IEC JTC1/SC29/WG11, Test Model 3, Doc. N0328, Draft, November 1992.

[2] ISO/IEC JTC1/SC29/WG11, Test Model 5, Doc. N0400, Draft Revision 2, April 1993.

[3] ISO/IEC, "Information Technology--Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5Mbits/s," International Standard 11172-2, May 1992.

[4] ISO/IEC, "Information Technology--Generic Coding of Moving Pictures and Associated Audio Information: Video," International Standard 13818-2, May 1996.

[5] ISO/IEC, "Information Technology--Generic Coding of Moving Pictures and Associated Audio: Systems," International Standard 13818-1, November 1994.

[6] ISO/IEC, "Information Technology--Generic Coding of Moving Pictures and Associated Audio: Audio," International Standard 13818-3, November 1994.

[7] International Telecommunication Union, "Recommendation H.261---Video Codec for Audiovisual Services at p*64 kbit/s," Geneva, August 1990.

[8] International Telecommunication Union, "Video Coding for Low Bitrate Communication," ITU-T Draft H.263, May 1996.

[9] Tino von Roden, "H.261 and MPEG1 -- A Comparison," *IEEE International Phoenix Conference on Computers and Communications*, New Jersey, 1996, pp. 65-71.

[10] Bernd Girod, Eckehard Steinbach, and Niko Faerber. "Comparison of the H.263 and H.261 Video Compression Standards," *Proceedings of SPIE -- The International Society for Optical Engineering*, 1995, pp. 233-251.

[11] Thomas G. Robertazzi, Performance Evaluation of High Speed Switching Fabrics and Networks (ATM, Broadband ISDN, and MAN Technology), IEEE Press, 1993.

[12] Arun N. Netravali and Barry G. Haskell, Digital Pictures: Representation, Compression, and Standards, 2nd Edition, Plenum Press, New York, 1995.

[13] N. S. Jayant and Peter Noll, Digital Coding of Waveforms, Prentice Hall, New York, 1994.

[14] Jae S. Lim, Two-Dimensional Signal and Image Processing, Prentice Hall, New Jersey, 1990.

[15] Alan V. Oppenheim and Ronald W. Schafer, Discrete-Time Signal Processing, Prentice Hall, 1989.

[16] Didier Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, April 1991, pp. 46-58.

[17] Cheng-Tie Chen, "Video Compression: Standards and Applications," *Journal of*

Visual Communication and Image Representation, June 1993, pp. 103-111.

[18] Andria H. Wong and Cheng-Tie Chen, "A Comparison of ISO MPEG1 and MPEG2 Video Coding Standards," *Proceedings of SPIE, Visual Communications and Image Processing '93,* November 1993, pp. 1436-1448.

[19] Mario Baldi and Yoram Ofek, "End-to-end Delay of Videoconferencing Over Packet Switched Networks," *in preparation*, IBM T.J. Watson Research Center, September 1996.

[20] Ting-Chung Chen and Wen-Deh Wang, "Low-Delay Center-Bridged and Distributed Combining Schemes for Multipoint Videoconferencing," *Proceedings of SPIE, The International Society for Optical Engineering*, 1994, pp. 850-861.

[21] Shaw-Min Lei, Ting-Chung Chen, and Ming-Ting Sun, "Video Bridging Based on H.261 Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, August 1994, pp. 425-437.

[22] Elliot Linzer, "A Robust MPEG2 Rate Control Algorithm," IBM T. J. Watson Research Center, January 1996.

[23] Masahisa Kawashima, Cheng-Tie Chen, Fure-Ching Jeng, and Sharad Singhal, "A New Rate Control Strategy for the MPEG Video Coding Algorithm," *Journal of Visual Communication and Image Representation*, September 1993, pp. 254-262.

[24] Cheng-Tie Chen and Andria Wong, "A Self-Governing Rate Buffer Control Strategy for Pseudoconstant Bit Rate Video Coding," *IEEE Transactions on Image Processing,* January 1993, pp. 50-59.

[25] I. E. G. Richardson and M. J. Riley, "Improving the Error Tolerance of MPEG video by varying slice size," *Signal Processing* vol. 46 n 3 October 1995, pp. 369-372.

[26] Max Chien, Huifang Sun, and Wilson Kwok, "Temporal and Spatial POCS Based Error Concealment Algorithm for the MPEG Encoded Video Sequence," *Proceedings of SPIE, The International Society for Optical Engineering*, 1995, pp. 168-174.

[27] Jen-Fwu Shen and Hsueh-Ming Hang, "Compressed Image Error Concealment and Postprocessing for Digital Video Recording," *Proceedings, IEEE Asia-Pacific Conference on Circuits and Systems*, 1994, pp. 636-641.

[28] Hanoch Magal, Reuven Ianconescu, and Peretz Meron, "A Robust Error Resilient Video Compression Algorithm," *IEEE MILCOM 1994*, pp. 247-251.

[29] Huifang Sun and Joel Zdepski, "Error Concealment Strategy for Picture-header Loss in MPEG Compressed Video," *Proceedings of SPIE, The International Society for Optical Engineering*, 1994, pp. 145-152.

[30] Augustine Tsai, Stephen Wiener, and Joseph Wilder, "Error Concealment using Multiresolution Motion Estimation," *Proceedings of SPIE, The International Society for Optical Engineering*, 1995, pp. 321-325.

[31] Paul Salama, Ness B. Shroff, Edward J. Coyle, and Edward J. Delp, "Error Concealment Techniques for Encoded Video Streams," *IEEE International*

Conference on Image Processing, 1995, pp. 9-12.

[32] O. A. Aho and J. Juhola, "Error Resilience Techniques for MPEG-2 Compressed Video Signal," *Proceedings of the International Broadcasting Convention*, 1994, pp. 327-332.

[33] Paul Hodgins and Eisaburo Itakura, "Error Issues of MPEG-2 over ATM," *ATM Forum Technical Committee*, Ottawa, Canada, September 1994.

[34] Ya-Qin Zhang and Xiaobing Lee, "Performance of MPEG Codecs in the Presence of Errors," *Journal of Visual Communication and Image Representation*, December 1994, pp. 379-387.

[35] Cheng-Tie Chen, "Error Detection and Concealment with an Unsupervised MPEG2 Video Decoder," *Journal of Visual Communication and Image Representation*, September 1995, pp. 265-279.