BANDWIDTH-AWARE VIDEO TRANSMISSION WITH ADAPTIVE IMAGE SCALING

By

ARUN S. ABRAHAM

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

UNIVERSITY OF FLORIDA

2003

Dedicated to my Father and Mother.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Dr. Jonathan C. L. Liu, for his guidance and encouragement. Without his confidence in me, I would not have been able to do this thesis. I would like to thank Dr. Douglas D. Dankel II and Dr. Richard E. Newman for serving on my thesis committee. I would also like to thank Dr. Ju Wang of the Distributed Multimedia Group for his critical guidance and contributions towards this research especially regarding the optimal rate-resizing factor section (Chapter 3).

I would like to thank my parents for always encouraging me towards higher studies and for all their love and support. Also, I would like to thank my dear wife for her love and sacrifice while living a student life pursuing higher studies. I would also like to thank my brother, the first one in the family to attain the master's degree, for his love and encouragement. And above all, I would like to thank God for making everything possible for me.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

ABSTRACT x

CHAPTER

1 INTRODUCTION 1

1.1 Novel Aspects 3
1.2 Goals 6
1.3 Overview 8

2 BACKGROUND 9

2.1 MPEG-2 Overview 9
2.2 Rate Distortion 11
2.3 MPEG-2 Rate Control 13

3 APPROXIMATING OPTIMAL RATE - RESIZING FACTOR FUNCTION 18

4 SYSTEM DESIGN 22

4.1 Scaled-PSNR Based Approach 24
4.2 Dynamic Adaptive Image Scaling Design 26

5 EXPERIMENTAL RESULTS 30

5.1 Case 1 30
5.2 CASE 2 33
5.3 Further Analysis 36

6 CONCLUSION 38

6.1 Contributions 38
6.2 Future Work 38

APPENDIX

A PERTINENT MPEG-2 ENCODER SOURCE CODE CHANGES 40

mpeg2enc.c 40
putseq.c 47
putbits.c 56
Sample Encoder Parameter (PAR) File 58

B PERTINENT MPEG-2 DECODER SOURCE CODE CHANGES 59

mpeg2dec.c 59
getbits.c 69

C MATLAB CODE FOR OPTIMAL RATE-RESIZING FACTOR APPROXIMATION 73

D CASE 2 TEST PICTURES 74

LIST OF REFERENCES 76

BIOGRAPHICAL SKETCH 78

LIST OF TABLES

Table page

1. Case 1 Adaptive Image Scaling I-Frame PSNR Data 31

2. Case 2 Adaptive Image Scaling I-Frame PSNR Data 33

3. Frame 45 Testing Of Sequence mei20f.m2v 36

4. Frame 45 Testing Of Sequence bbc3_120.m2v 36

LIST OF FIGURES

Figure page

1. Programming Model 6
2. MPEG Frame References....10
3. Hierarchical Layers of an MPEG Video 10
4. Group of Pictures 11
5. R-D Curve Showing the Convex Hull of the Set of Operating Points. 12
6. Bit Allocation of TM5 Rate Control Bit Rate Is Set to 700kbps.Video Resolution Is 720x240. Image Complexity for I, P, B Frames is set to Xi=974kbits, Xp=365kbits, Xb=256kbits, respectively....14
7. Average Quantization Scale for a GOP, Encoded at 700kbps 15
8. GOP Target Buffer Overflow, Encoded at 700 kbps 15
9. Reducing the Number of MB Before Normal Rate Control and Encoding. 16
10. Effects of Image Scaling 17
11. The Overall Distortion-Resizing Functions When Different Target Bit Rates Are Given. The Video Resolution Is 704×480 (DVD Quality). 21
12. System Design Components....22
13. Bandwidth-Adaptive Quantization....23
14. Scaled-down PSNR Values....24
15. Scaled-down Quantization Values 25
16. Adaptive Image Scaling Scenario 29
17. Case 1 Scaled-PSNR Comparison....31
18. Case 1 Scaled-Quantization Comparison....32

19. Case 1 GOP Target Bits Overflow Comparison 33

20. Case 2 Quantization Comparison 34

21. Comparison of Scaled-down PSNRs.....34

22. Reference Picture for 3rd I-Frame (PSNR = 49.0, S=1,425,311) 74

23. 3rd I-Frame Using Original Encoded Picture (PSNR = 20.3, S=91,627) 75

24. 3rd I-Frame Using Adaptive Image Scaled Picture (PSNR=19.9, S=36,081) 75

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

BANDWIDTH-AWARE VIDEO TRANSMISSION
WITH ADAPTIVE IMAGE SCALING
By

Arun S. Abraham

August 2003

Chair: Jonathan C. L. Liu
Major Department: Computer and Information Science and Engineering

It is essential that MPEG-2/4 video encoding makes the best use of the available bandwidth and generate the compressed bit-stream with highest quality possible. With many types of client machines, it is preferred to have a single format/resolution stored in the storage system and adaptively transmit the video content with proper resolution for the targeted client and network.

Conventional approaches use large quantization scales to adapt to low bit rates. However, the larger quantization scale degrades the image quality severely. We believe that in addition to adaptive quantization for bit allocation, the image resolution should be jointly adjustable. The key design factor should be to reduce the total number of macro-blocks per frame. Our goal is thus to investigate the effects of scaling down the video as an alternative to adapt to bandwidth fluctuations (e.g., 3/4G wireless networks).

We envision that the proposed scheme in addition to the standard TM5 rate control mechanism would be more effective during low-bandwidth situations. From the results,

we see that during low bandwidth situations, scaling down the image does provide comparable image quality while greatly reducing bandwidth requirements (i.e., usually more than 2X). The gain on the video quality can be significantly improved (i.e., up to 2 dB) by re-rendering the image into a smaller resolution with a relatively precise quantization scale (i.e., usually 50% to 100% less than the conventional encoders).

CHAPTER 1
INTRODUCTION

Streaming video is becoming a more common practice recently due to the rapidly growing high-speed networks. To provide high quality video images to the end-user, the underlying network needs to be able to maintain the necessary bandwidth during the playback session. Ideally, QoS-guaranteed communication network (e.g., ATM [12] and IPv6 [3]) should be used for multimedia transportation. However, in the near future, the majority of streaming video applications will still run using best-effort networks (e.g., Internet), where the actual bandwidth is characterized by short-term fluctuations (and the amount can be limited). This observation remains valid when applied to the next generation wireless networks (3G/4G systems).

For example, cellular phone users are rarely guaranteed optimal bandwidth availability. As demonstrated in Wang and Liu [17], the per-channel bandwidth for mobile users is subjected to the cell load and the spreading factor used. Furthermore, the probability of bandwidth changes for the mobile can be higher than 60% during the handoff transition.

Thus, without a proper design, the playback at the end users can be often delayed and video quality suffers different degrees of jitter when transporting compressed video through these networks [1]. This is the common experience when commercial video players (e.g., RealPlayer and Microsoft's Media Player) are used with the current Internet.

Thus, with the use of best-effort networks, it is essential that video encoding makes the best use of the available bandwidth and generate the compressed bit-stream with highest quality as possible. Bandwidth-adaptive video encoding technology, which dynamically updates the video encoding parameters according to the current network situation, is expected to deliver better video quality with limited bandwidth.

However, providing a single video quality for all kinds of client machines becomes a challenging issue. Today's client machines can be conventional PCs, laptop computers, PDA, and Information Appliances (IA). These machines/devices have different sizes of display resolutions and capability. Having different resolutions of the same video content in the servers will waste the efficiency of the storage system since a video title can occupy several gigabytes of disk space.

Thus, it is preferred to have a single format/resolution stored in the storage system and adaptively transmit the video content with proper resolution for the targeted client and network. To perform this kind of adaptation, a video server needs to dynamically adjust the video quality to accommodate the changes in the network conditions (e.g., sudden drop of bandwidth available).

There are various quality factors that can be adjusted to adapt to the sudden drop in bandwidth: increase quantization factor, decrease the color [13], and decrease the video dimensions. The problem in its general format can trace its origin back to the classic rate-distortion theory [2,6].

In general, the rate-distortion theory provides fundamental guidelines to find an optimal coding scheme for an information source, such that the average loss of data

fidelity can be minimized, where the average bits per symbol of the coding scheme is given [2].

For a block transform based video coder (e.g., MPEG-2/4 [19]), the key problem is how to distribute limited bit budget to the image blocks (e.g., 16×16 macroblock in MPEG-2) to achieve the optimal R-D curve. In a practical video encoder, we need to decide the quantization scale and other factors for the proper transformed image blocks.

1.1 Novel Aspects

Some related work on this area can be found in the literature [2,4,6,7,13,15]. However, with the exception of Puri and Aravind [15] and Fox et al. [4], their common focus is on adjusting the quantization parameter of the DCT coefficients based on a particular rate-distortion model. Furthermore, their models are based on using a fixed image resolution, which imposes unnecessary limitations on further exploiting the rate-distortion theory and results in worse video quality when the bit budget is low. Puri and Aravind [15] discuss various general ways an application can adapt to changes in resources but does not mention resolution scaling. Fox et al. [4], on the other hand, provide design principles in application-aware adaptations including resolution scaling as one of the ways of dealing with network variability. However, Fox et al. [4] do not focus on performing a thorough quantitative and theoretical analysis of image scaling. Additionally, no other work that we know of provides experimental results of MPEG-2 encoder using dynamic Adaptive Image Scaling.

When the available data rate is low, a large quantization scale will be used in the traditional rate control methods [19], to maintain the desired bit rate. The larger quantization scale degrades the image quality severely and causes perceptual artifacts

(such as blockness, ring, and mosquito noise), since the detailed image structure cannot survive a coarse quantization.

Another level of bit allocation occurs in the group of picture (GOP) level, where bits are distributed among frames in each GOP. However, due to the imperfect rate control, the P and B frames at the end of a GOP are often over-quantized to prevent the buffer overflow. A large variation of image quality is thus observed, especially when the given bit rate is low.

Therefore, a more precise and effective method for the bit allocation and encoding scheme should be investigated to provide robust video quality when time-network condition is varying. The new method should be effective in improving video quality both objectively and subjectively. It must be able to react quickly to the changes from the network and the encoding complexity should be low to suit the real time encoding requirements.

To this end, a network-probing module as proposed by Kamat et al. [8] and Noble and Satyanarayanan [13] can be used in the video server to monitor the connection status. Additional processing overhead is also required at the video server to perform the refined bit allocation algorithm. Nevertheless, the computational overhead can be affordable since the CPU resources at the video server are largely under-utilized [10]. When live encoding is required, we assume that the video server is equipped with programmable encoding hardware/software that is able to accept different encoding parameters.

The success of the proposed scheme depends on the answer to an immediate question: *how can we use a smaller quantization for each macroblock (i.e., more bits) while maintaining the overall bit rate requirement for each frame, or each GOP?*

The task sounds extremely difficult at first, since increasing bits per macroblock and reducing the bits per frame can be opposite goals. However, with further study along this direction, we have found that it is possible to meet both goals under one condition. The key design factor should be to reduce the total number of macroblocks per frame. Previous study in adaptive rate control usually assumes that this design factor is to be fixed all the time.

But this assumption can be challenged by today's advanced video card technology. Many modern video cards come with advanced image processing capability (e.g., dithering and Z-buffering). These capabilities help to deliver better video quality at the client machines. Thus, though given a smaller resolution (i.e., reduction of the macro-blocks), the final display at the client machines can be compatible with the original quality.

It is based on this idea that we proposed a new video encoding scheme, where not only adaptive quantization is used for bit allocation, but also the image resolution is adjustable to control the bit rate in a higher level. The proposed scheme also addresses the unfair bit allocation issue under small bit rate budgets.

It is observed that, with a small bit rate budget, a larger quantization scale Q is often used, which makes it more erroneous to control the bit allocation. We have found that the actual bits used for the last few frames usually generated a significant buffer overflow. Our proposed scheme did eliminate the unfair bit allocation (thus the corresponding quality degradation).

The choices thus becomes: when the bit budget runs low, we can either down-sample the image to a smaller resolution and use a rather precise quantization parameter,

or directly adapt to a coarse quantization scale with the original image size. Since low-speed networks are more common than high-speed networks these days, scaling-down is perhaps more urgent than scaling-up.

1.2 Goals

Our study is thus to investigate the effects of scaling down the video as an alternative to adapt to bandwidth fluctuations. We believe that gradually decreasing the image size would help alleviate potential losses because of the low bandwidth.

Along this line, one major design goal was to retain the integrity of the baseline functionality of the encoder and decoder even at the loss of performance. We wanted to first understand the behavior of Adaptive Image Scaling and afterwards focus on performance.

As shown in Figure 1, the image scaling adaptation was thus added to the encoder as an outer loop outside the main encoding functionality. Therefore, the image scaling adaptation loop determines which scale to be used based on the current bit rate. This type of programming model is also used in Noble and Satyanarayanan [13] to obtain positive results for making applications adaptive.

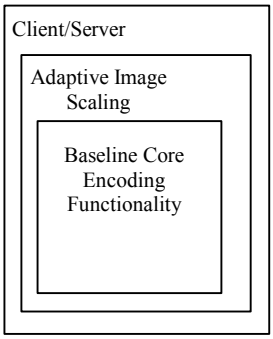

Figure 1. Programming Model

Of course, our study also stresses the importance of providing a user-unaware adaptation as much as possible (i.e., the size variations should not be noticed by the user).

This is accomplished by specifying the preferred display resolution in the encoded video stream, a feature defined by MPEG-2. At the decoder side, the decompressed video images will be re-rendered to the preferred display size, instead of the actual encoded resolution. For the sake of simplicity, the display size at the client remains unchanged for the same video sequence to provide a consistent viewing effect.

Therefore a big challenge is: for a given bit rate, how can we determine the best image scale factor and quantization parameters for encoding the video at the best quality level. In this thesis, our focus is on the determination of the scale factor and frame level bit allocation. For the macroblock level bit allocation, we assume that adaptive quantization is used as proposed by Gonzales and Viscito [5] and ISO/IEC 13818-2 [19].

Theoretically, there exists a mapping between the current bit rate and the optimal image-resizing factor that leads to the minimal distortion. The optimal mapping could be obtained empirically by encoding with all possible resizing factors and comparing the objective video quality. However, such pre-encoding processing lacks practical value due to the tremendous computation involved and is impossible in the case of live video.

We investigated two approaches: the first one assumes a mathematic model obtained from a variation of classic rate-distortion function, and the second method uses an implicit rate control based on PSNR feedback. We used the mathematical model as an approximation for modeling our system. Then the performance results from the PSNR-based approach are demonstrated for supporting the overall design goal.

We have finished the implementation by introducing an image re-sizing module to generate the image data on the fly. The resizing factor is calculated every GOP, based on the current network bandwidth and the image complexity. With the continuous image

resizing, a more accurate rate control can be implemented. Another advantage of the online image re-sampling is that the video server now only stores the original video images.

From our experimental results, we have observed that video quality can be significantly improved (i.e., up to 2 dB) by re-rendering the image into a smaller resolution with a relatively precise quantization scale (i.e., usually 50% to 100% less than the conventional encoders).

Specifically, the experimental results show a promising trend for low-bit-rate video transmission. Using a scaled-down resolution for encoding provides comparable picture quality. Note that the conventional encoders uses a drastically higher quantization scale and utilizes more than double the bandwidth required for approximately the same picture quality. We thus believe the proposed scheme is suitable for the emerging 3G wireless networks, which are targeted for multimedia communications using a limited bandwidth.

1.3 Overview

The remainder of this thesis is organized as follows: Chapters 2 and 3 introduce the theoretical background of rate control theory along with the expected results on the reduction factor. Chapter 4 explains the basic software design of the proposed encoder system. Chapter 5 presents detailed experimental results. Chapter 6 concludes this thesis by discussing the unique contributions and potential future work for this research.

CHAPTER 2
BACKGROUND

This chapter provides essential background information that will help understand this research. The organization and syntax of an MPEG video will be provided. Also, theoretical information of what can be done as the rate deteriorates will be discussed. Finally, this chapter ends by discussing what the baseline encoder does when the bit rate decreases.

2.1 MPEG-2 Overview

MPEG (Moving Picture Experts Group) is a committee that produces standards to be used in the encoding and decoding of audio-visual information (e.g., movies and music). MPEG works with International Standards Organization (ISO) and the International Electro-Technical Commission (IEC). MPEG-1, MPEG-2, and MPEG-4 are widely used standards generated from MPEG. MPEG is different from JPEG in that MPEG is primarily for moving pictures while JPEG is only for still pictures.

Moving pictures are generated by decoding frames of still pictures together, usually at a rate of 30 frames per second. To provide optimum compression and user perceived quality, MPEG capitalizes on the redundancy of video both from subsequent frames (i.e., temporal) and from the neighboring pixels of each frame (i.e., spatial). More specifically, MPEG exploits temporal redundancy by the prediction of motion from one frame to the next, while spatial redundancy is exploited by the use of Discrete Cosine Transform (DCT) for frame compression. To exploit these redundancies, MPEG strongly relies on syntax.

The three main types of frames in MPEG are (listed in priority): P, B, and I. The I-frame is used as the reference frame and has no dependency on any other frames. The I-frame is intra-coded (without dependency on other frames), while the P- and B-frames are inter-coded (depends on other frames). P-frames (i.e., predicted frames) use only forward references while B-frames (i.e., bidirectional frames) use both forward and backward references. Figure 2 depicts the relation between the different frame types.

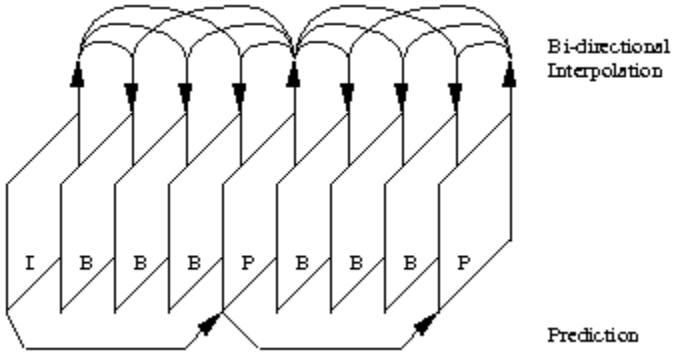

Figure 2. MPEG Frame References [11]

An MPEG video has six hierarchical layers (shown in Figure 3). The highest layer is the Sequence layer, which provides a complete video. The next layer is the Group of

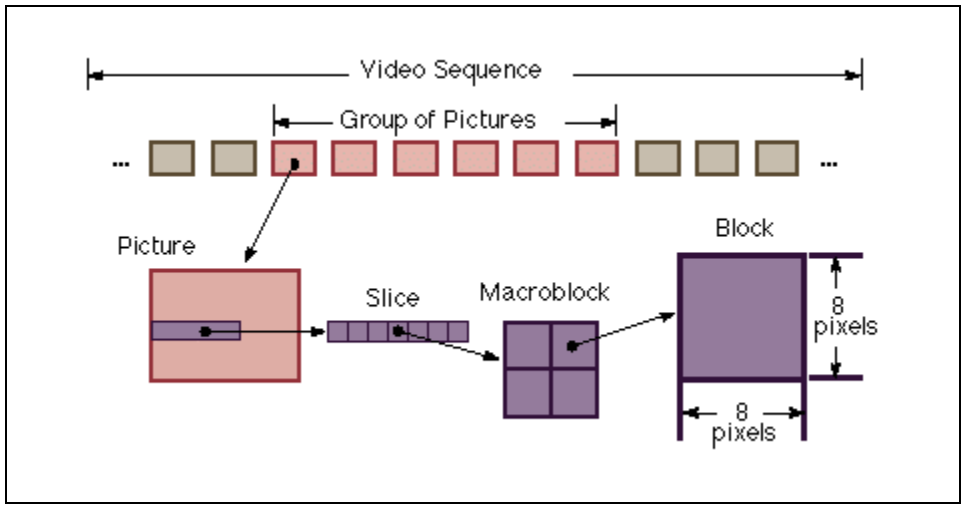

Figure 3. Hierarchical Layers of an MPEG Video [11]

Pictures (GOP) layer (shown in Figure 4) which compromises a full set of the different frame types consisting of only one I-frame and multiple P- and B-frames. The next layer is the Picture layer which is a single frame consisting of multiple slices (the Slice Layer). Each slice can contain multiple macroblocks (the Macroblock Layer), which contain information about motion and transformation. Each macroblock consists of blocks (the Block Layer) which are 8×8 values encoded using DCT.

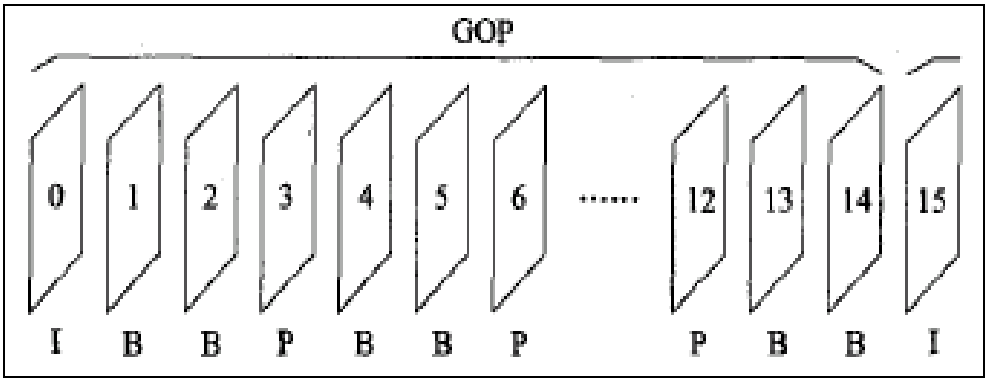

Figure 4. Group of Pictures [2]

2.2 Rate Distortion

Video is encoded using various parameters such as the dimensions of the frame, the color format, and the available bit rate. If the available bit rate is high, then the encoder can use low compression parameters such as a small quantization scale to provide the best quality video at a constant bit rate (CBR). Using a small quantization scale, the frames are encoded precisely and result in larger-sized frames. When the available bit rate varies (VBR), the encoder must decrease the amount of generated bits by using a high quantization scale to meet the lower bit budgets. By adapting to the network variability and providing VBR encoding, the encoder can provide a constant quality video [14]. If the encoder was bandwidth-*un*aware, during low bandwidth situations, it will generate wasteful information [9].

Keeping all the other encoding parameters constant and gradually lowering the bit rate results in the gradual degradation (i.e., distortion) of the video. Minimizing the distortion while meeting the budget implied by the bit rate is the general *rate-distortion problem*. Choosing the optimal encoding parameters that will provide the best possible video quality under the circumstances can minimize distortion. An instance of encoding parameters can be referred to as an operating point [16]. An encoder could try different operating points to see which one provides the best video quality. The convex hull of all the set of operating points provides the best rate distortion performance [16]. The goal of rate-distortion optimization is to find the operating point that will be as close to the curve generated by the convex hull as shown in Figure 5. For a given rate (R1), the distortion will vary based on the different encoding parameters used.

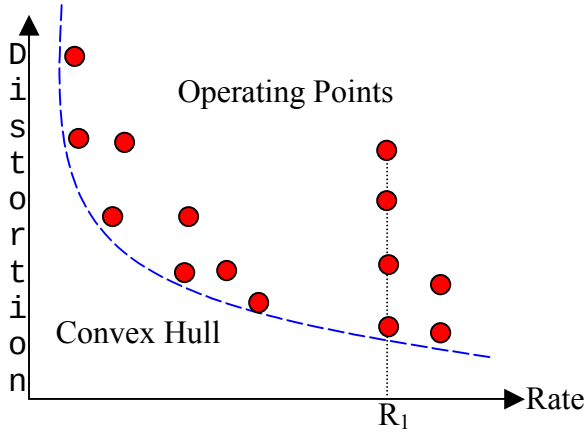

Figure 5. R-D Curve Showing the Convex Hull of the Set of Operating Points [16].

Given the real-time constraints of encoding, optimal operating point cannot be computed on-the-fly, but rather rate control algorithms can be employed to provide the best operating point using general guidelines to meet the target bit budget.

2.3 MPEG-2 Rate Control

The bit allocation in MPEG-2 consists of two steps: (1) target bit estimation for each frame in the GOP and (2) determination of reference quantization scales in the macroblock level for each frame. In the GOP level, bit assignment is based on the frame type. For instance, an I-frame usually has the highest weight and gets the most share, while P-frames have the next priority, and B-frames are assigned the smallest portion of the total bits.

In the frame level, the TM5 rate control algorithm in MPEG-2 controls the output data rate by adaptive quantization. The DCT coefficients, which consist of an 8×8 block, are first quantized by a fixed quantization matrix, and then by an adaptive quantization scale Q. The elements in the fixed quantization matrix are decided according the sensitivity of human visual system to the different AC coefficients and are obtained empirically based on many experiments. The quantization scale Q serves as an overall quantization precision.

This rate control algorithm performs well when the bit rate budget is high; however, it might result in very unfair bit allocation under small bit rate budgets. With a small bit rate budget, a larger quantization scale Q is often used, which makes it more erroneous to control the bit allocation. An often-observed direct result is that the actual bits used for the first few frames are greater than the target bits calculated in the GOP level rate control. This further devastates the bit shortage for the rest of the frames in the same GOP and either causes buffer overflowing (generated bit-stream consumes more bandwidth than allowed) or sudden decrease of the picture quality. Figure 6 demonstrates such performance degradation for an MPEG-2 encoder using a low bit rate.

The performance degradation of the conventional rate control at low bit rates is not so surprising, though, since the accuracy of the rate-distortion theory will only be guaranteed when the bit budget is sufficient and the quantization scale is small. While at low bit rates, the quantization scale is large, and consequently makes the low-order (or even linear) approximation of the rate-distortion function more erroneous.

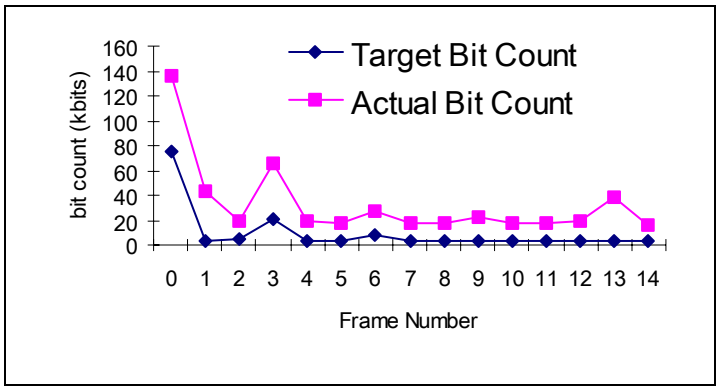

Figure 6. Bit Allocation of TM5 Rate Control Bit Rate Is Set to 700kbps.Video Resolution Is 720x240. Image Complexity for I, P, B Frames is set to Xi=974kbits, Xp=365kbits, Xb=256kbits, respectively.

The average quantization scale and the amount of buffer overflow are shown in Figure 7 and Figure 8 for the above encoding experiments. In Figure 7, it can be seen that the quantization scale increases as the encoding progresses. Towards the end of the GOP, the quantization scalar (Q) used for the B-frames increases to 112, which is the pre-defined maximum value allowed value in the encoder. Any Q value higher than 112 is cut off to maintain a minimum level of viewing quality. This, however, causes severe buffer overflowing for these frames.

The buffer overflow measurement provides a good indication for the reliability and effectiveness of a rate control algorithm. The overflow equals zero when the buffer is not full and equals the accumulated actual bit-count that exceeds the transmission buffer otherwise. Figure 8 shows the measured buffer overflow for the low bit rate encoding

mentioned above. The size of the (transmission) buffer is set to the target bit budget for a group of picture, which is 303333 bits in this test. As the frames in the GOP are encoded, the encoded bits are temporarily stored in the transmission buffer, which will be sent to the network when full.

Also in Figure 8, we observed that the buffer is not overflowed for the first four frames. At the fifth frame (a B-frame), the actual accumulated encoding bits exceed the transmission buffer by 25888 bits. The buffer overflow keeps increasing for the rest of the frames and reaches the highest level of 138411 bits by the end of GOP.

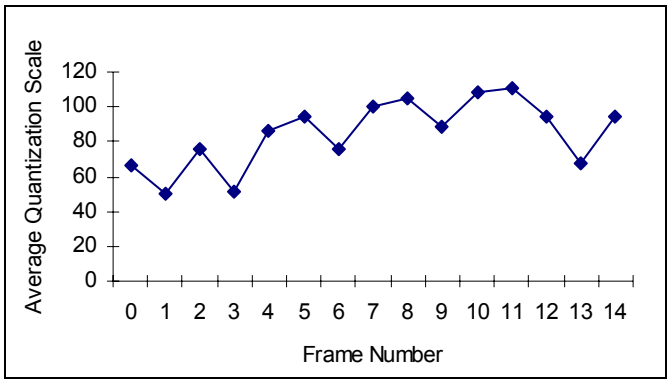

Figure 7. Average Quantization Scale for a GOP, Encoded at 700kbps

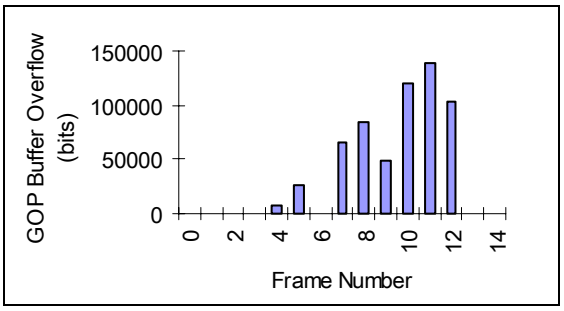

Figure 8. GOP Target Buffer Overflow, Encoded at 700 kbps

We believe that a new way to control the encoding bit rate should be considered besides tuning the quantization scalar. A direct but effective method is to reduce the number of macroblocks (MB) to be encoded. With a smaller number of macroblocks, we

will have a higher bit per symbol value, thus a smaller quantization scalar. Figure 9 describes the simplified new video encoding scheme.

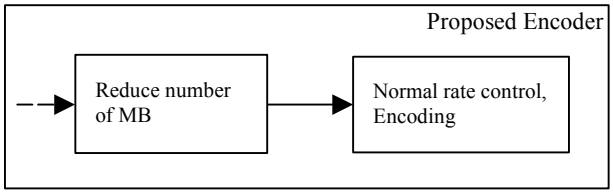

Figure 9. Reducing the Number of MB Before Normal Rate Control and Encoding.

The rationality behind this new scheme is that, by reducing the number of macroblocks, the encoder should be able to reduce the information loss due to heavy quantization which otherwise is inevitable. Nevertheless, we do suffer information loss during the first step when the number of macroblocks are reduced. There is a tradeoff to make via the comparison of information loss. We provide the theoretical analysis of this decision process and the empirical method in the rest of this thesis.

Another aspect of reducing the number of MB is how we should select the original image data in the scaled-down version. For example, we can cut off the image data on the edges, while only preserving the image in the center, or one may sort the MB in the order of importance (e.g., the higher importance is given to MB with high variance) and skip those MB at the end of the list. The full discussion of the strategies on this direction is beyond the scope of this thesis.

In the context of this thesis, we assume that the reduction of the macroblock number is achieved by an image resizing process, which converts the original image to a new image by proportionally scaling down in both dimensions. The resizing process is described by the following equations:

$$I'(u',v') = \frac{1}{|A_{u,v}|} \sum_{p \in A_{u,v}} I(p) \quad \text{where}$$
$$A_{u,v} = \left\{ (x,y) \in N^2 \mid \sqrt{(x-u')^2 + (y-v')^2} < d_0 \right\}$$
$$u' = \sqrt{f}.u, \quad v' = \sqrt{f}.v \quad \text{and} \quad f \in (0, \ 1].$$

As shown in Figure 10, $I'(u',v')$ is the pixel value of the resized image at coordinate (u',v'), $A_{u,v}$ represents the corresponding pixels within a radius of d_0 in the original image centered at (u,v), and f is the resizing factor. The pixel value in the new-scaled image is the average of the pixels in the neighbor area in the original image defined by the resizing factor f and the coordinate (u,v).

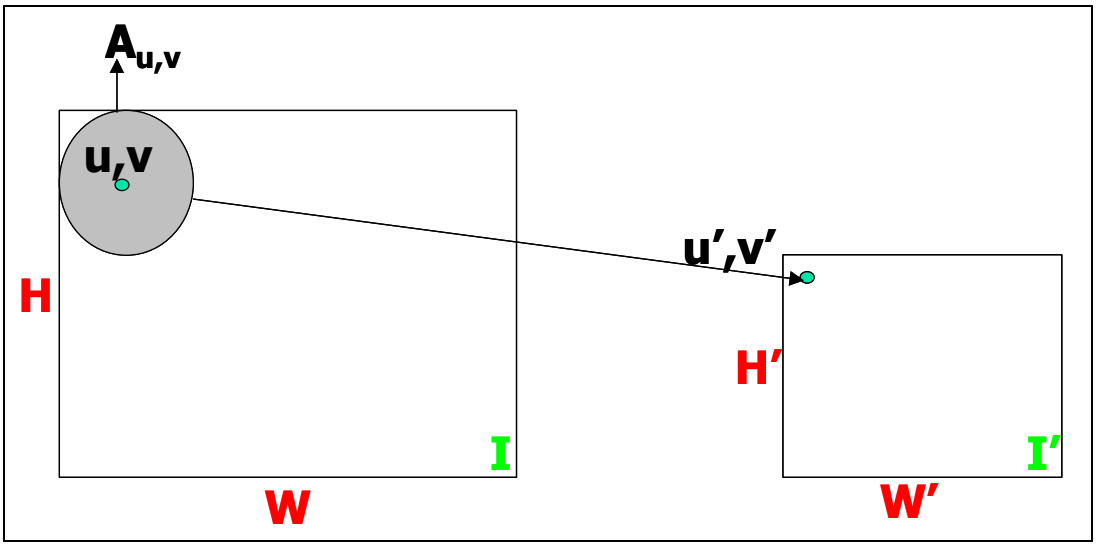

Figure 10. Effects of Image Scaling

The rate distortion theory implies that there is an optimum resizing factor, which will provide the minimum distortion. The next chapter takes an analytical approach into finding the optimal resizing factor.

CHAPTER 3
APPROXIMATING OPTIMAL RATE - RESIZING FACTOR FUNCTION

In this chapter, we provide an analytical model such that the optimal image size can be derived. The distortion measurements in video encoding are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). MSE is the average squared error between the compressed and the original image, whereas PSNR is a measure of the peak error in dB. In MPEG-2 encoding, color component (YUV) in the raw image is represented by an 8-bit unsigned integer number. Thus the peak signal value is always less than or equal to 255. The PSNR and MSE are expressed as follows:

$$PSNR = 10 \log\left(\frac{255^2}{\sqrt{MSE}}\right), \text{ and}$$

$$MSE = \frac{1}{W.H}\sum_{y=1}^{H}\sum_{x=1}^{W}\left(I(x,y) - \hat{I}(x,y)\right)^2.$$

The width and height of the image are represented by W and H, respectively; $I(x,y)$ represents the original image and $\hat{I}(x,y)$, the reconstructed image after encoding-decoding. However, PSNR and MSE distortion measurements are very difficult to be used in an analytical model. In the following discussion, we use absolute difference as the measure of distortion.

We assume that the pixels of the video source represent a zero-mean i.i.d. (independent and identical distribution) with the signal variance of each pixel as σ_X^2. Results in Hang and Chen [6] indicate that the bits per symbol (b) versus the distortion caused by the normal encoding process (D_E) can be approximated by

$$b(D_E) = \frac{1}{\alpha \log_2 e} \log_2 (\varepsilon^2 \frac{\sigma_X^2}{D_E}). \quad (1)$$

In (1), α is a constant 1.386 and ε^2 is source dependent (usually 1.2 for Laplacian distribution). Rearranging (1), we have

$$D_E(b) = \varepsilon^2 e^{-\alpha b} \sigma_X^2. \quad (2)$$

In the case of MPEG-2 encoding, our focus is on the I-frame, which is completely intra-coded. For P and B frames, the rate distortion model is not applicable due to the coding gain from motion estimation. Now if we let B be the available bit count for a GOP, r be the percentage of bits assigned to the I-frame, and f be the image scale factor, then we have the following relation:

$$rB = fWHb. \quad (3)$$

Equation (3) reflects a simple fact, that the total bits to encode an I-frame (rB) equals the number of pixels (which is fWH after the resizing effect) times the average bits per symbol, b. Substitute (3) into (2), we have

$$D_E(b) = \varepsilon^2 e^{-\alpha(\frac{rB}{fWH})} \sigma_X^2. \quad (4)$$

From (4), we can see that the scale factor (f) is inversely proportional to the available bit count (B).

Equation (4) only represents the distortion between the pre-encoded image and the reconstructed image. To describe the complete distortion of the resizing-then-encoding process, we should quantatize the information loss during the resizing process D_R. With the assumption of i.i.d. distribution for the pixels, we define the image information (complexity) via the cumulative variance of all image pixels. The image complexity before and after resizing is $H(I) = HW\sigma_x$ and $H(I') = fHW\sigma_x$ respectively.

The loss of the information from the resizing process could be expressed by

$$D_R = H(I) - H(I') = (1-f)HW\sigma_x .$$

Now let us define the total distortion D_T as the summation of distortion caused by normal encoding (D_E), and the distortion caused by resizing (D_R), we have

$$D_T = D_R + D_E = \varepsilon^2 e^{-\alpha(\frac{rB}{fWH})}\sigma_X^2 + (1-f)HW\sigma_x . \quad (5)$$

The optimal resizing factor must correspond to the smallest total distortion (D_T). By taking the first-order derivative of D_T represented by (5) and equaling it to zero, we have

$$\frac{\partial D_T}{\partial f} = \varepsilon^2\sigma_X^2 \frac{\alpha B}{WH} f^{-2} e^{-\alpha(\frac{rB}{fWH})} - HW\sigma_x = 0 . \quad (6)$$

The solution of (6) corresponds to the peaks (or valleys) of local minimum/maximum distortion. To find the optimal resizing factor, the local peaks/valleys are substituted into (5) as well as the end point $f = 1$.

Figure 11 shows the relationship between the distortion and resizing factor based on (5). The image size is fixed to 704×480, and the curves for different target bit rates are plotted. When the target bit rate is set to 2400kbps, we observed that the optimal resizing factor is about 0.64. As the target bit rate decreases, the best resizing factor moves to the left (i.e., becomes smaller). For instance, the optimal resizing factor at 1800 kbps bit rate is 0.55. This behavior matches well with our predication. However, the resizing factor corresponding to the low-end bit rate (e.g., 100kbps) shows that the image size should be scaled-down by more than 30 times, which is not acceptable in reality. In fact, since the distortion model of (1) & (2) is valid when the bits per pixel are relatively

sufficient, this model only provides a reasonable predication when the target bit rate is adequate.

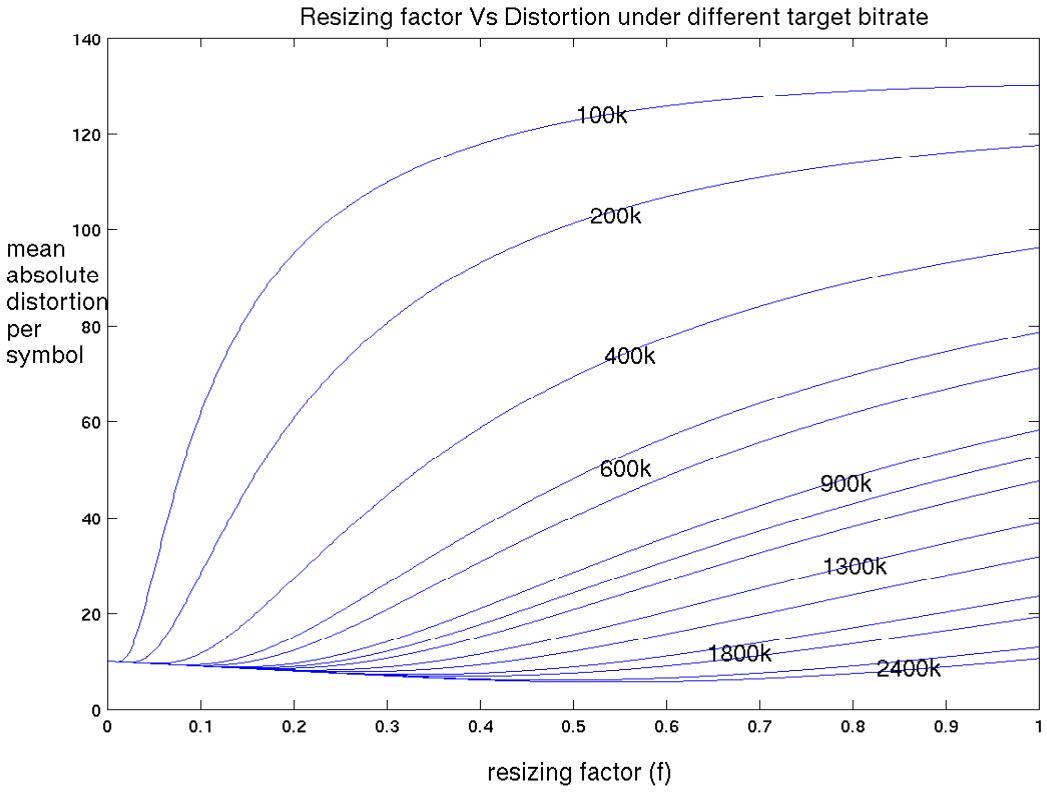

Figure 11. The Overall Distortion-Resizing Functions When Different Target Bit Rates Are Given. The Video Resolution Is 704×480 (DVD Quality).

From the mathematical analysis, we have observed that it is theoretically possible to find the optimal resizing factor given the bit rate and the image resolution. Nevertheless, the accuracy of the above analysis highly depends on the statistical assumptions of each pixel (i.e., that the variance is the same for all pixels). In reality, the i.i.d. assumptions are not always applicable. Therefore, we sought the PSNR based approach to investigate the effects of image scaling. The following chapters describe the experimental design and results of this investigation.

CHAPTER 4
SYSTEM DESIGN

This chapter provides the design rationale for this thesis. Results from initial testing are presented to provide reasoning for the choices made in the system design. Experimental assumptions, parameters, and scenarios will also be discussed.

To test Adaptive Image Scaling, software only versions of the MPEG-2 encoder and decoder were used. Figure 12 shows the System Design Components. The baseline for the encoder/decoder codec used in this study was from the MPEG Software Simulation Group (MSSG) shareware site [21]. The baseline encoder and decoder are standalone, and were modified to allow streaming video as shown in Figure 12. The modified encoder sends video streams over the network to the decoder. The individual frames that are to be encoded are stored on disk in PPM format. The frames are encoded and decoded on the fly over the network. The server and the client are connected using TCP/IP.

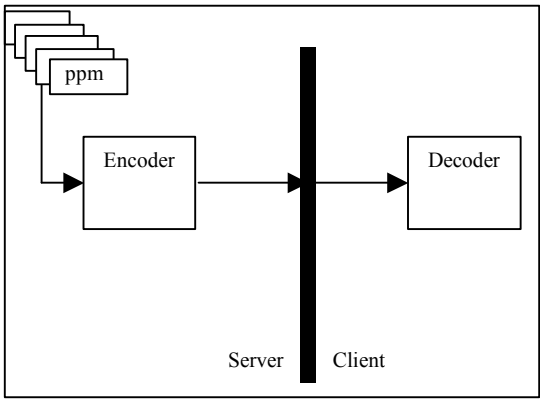

Figure 12. System Design Components

One of the parameters that can be passed to the encoder is the bit rate. The baseline encoder does adapt the quantization parameter based on the bit rate. As discussed earlier, the quantization parameter is bandwidth-adapted per Figure 13 (i.e., as the bit rate decreases, the quantization parameter increases). For high bit rates, the quantization parameter is set to the minimum value of 1, and for low bit rates, the quantization parameter is gradually increased until the maximum value of 112 is used. The bit count vs. quantization is nonlinear [2].

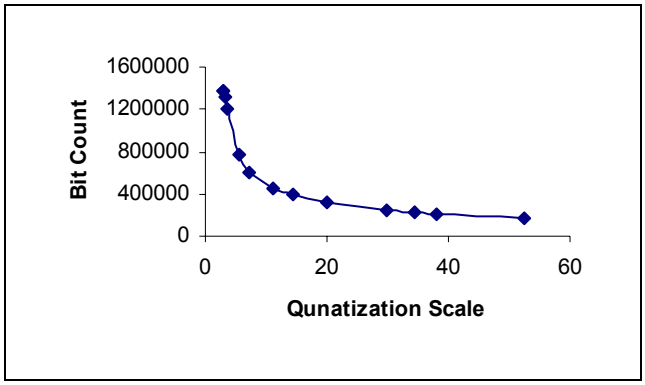

Figure 13. Bandwidth-Adaptive Quantization

When the encoder detects a drop in the bit rate, the reference frame will be scaled-down. As explained in the earlier chapters, sending a scaled-down version of the frame will require smaller amounts of bits to be transferred over the network, thus lowering the bandwidth requirement. The trade off for scaling down is the loss of picture quality. But the loss of quality that is experienced from scaling down should be better than without.

When the baseline decoder receives the scaled-down image, it automatically adjusts to the new scale. However, as mentioned earlier, from the user-unaware perspective, the user should only view the image at a fixed resolution (i.e., that of the reference picture). This can be achieved by modifying the decoder to only display the scaled image at the

reference size. When scaling up an image, negligible losses in the PSNR value were observed (average loss of 0.02).

4.1 Scaled-PSNR Based Approach

Since measuring video quality can be subjective [18], we decided to primarily use the objective metric of PSNR. The baseline encoder provides frame-by-frame PSNR information along with other statistical information that was used in our study. Reasons for pursuing a Scaled-PSNR based approach were gathered from initial experimental results which show that smaller sized resolutions produce as good or better PSNR as that of the original resolution for low bit rates. Figure 14 and Figure 15 show the PSNR and quantization values, respectively, for Frame 45 (4th I-Frame) obtained from multiple runs of the baseline encoder using different combinations of bit rates and resolutions. For initial testing, three different resolutions of the *mei20f.m2v* [20] sequence were used: 704×240 (*f=1.0*), 352×240 (*f=0.5*), and 352×120 (*f=0.25*). The three sets of resolutions were created and stored on disks prior to running the encoder. The bit rates that were tested are as follows: 10kbs, 100kbs, 300kbs, 500kbs, 1mbs, 1.5mbs, 1.7mbs, 2mbs, 3mbs, 4mbs, 5mbs, 7mbs, 9mbs, 15mbs, 20mbs, 40mbs, and 60mbs.

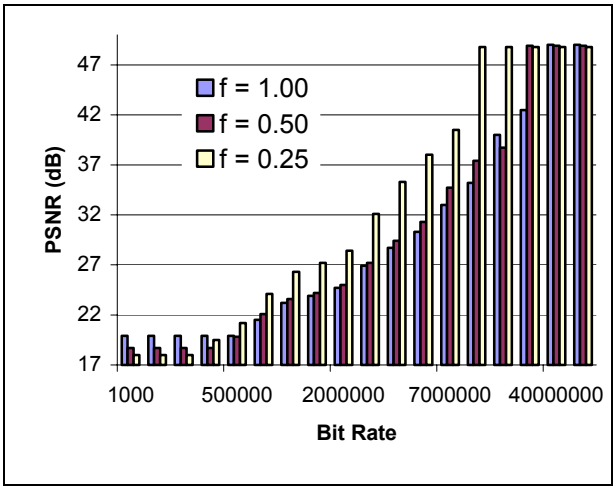

Figure 14. Scaled-down PSNR Values

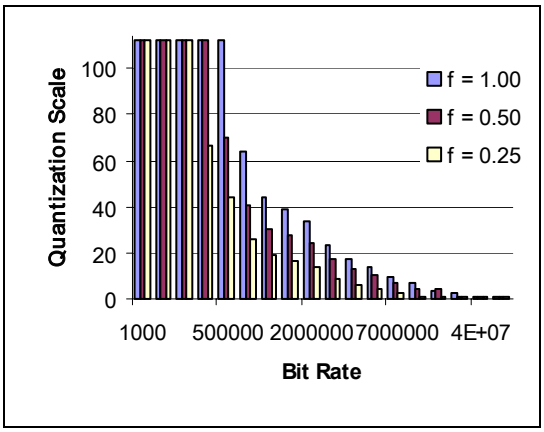

Figure 15. Scaled-down Quantization Values

It can be seen that as the bit rate increases for the different resolutions, the PSNR values saturate at a certain value when the quantization scale is set to the minimum value (1). For the *f=0.25* scale, the saturation point was seen at the bit rate of 9mbs with a value of 48.8dB (*f=0.5*, at 20mbs with a value of 48.9dB; *f=1.0*, at 40mbs with a value of 49.0dB). It can be observed that when the bit rate was more than adequate, the original resolution had better PSNR values. This shows that, when the bit rate is adequate, image scaling is not required.

However, as the bit rate decreases from the saturation point, we see that there is a huge variance between the quantization values used for the different scales. The biggest difference was seen at 500kbs where the *f=1.0* scale is using the maximum allowed quantization (112) while the *f=0.5* scale is using roughly half the amount, and the *f=0.25* scale is still maintaining a low quantization scale. We can see that the *f=0.25* scale increases its quantization to half the maximum value only at bit rates of 300kbs or lower, while the *f=1.0* scale begins to use the maximum quantization at bit rates less than 1mbs. At low bandwidths, the *f=1.0* scale does lose information from the use of higher quantization scales.

From the initial testing, it can be seen that the PSNR value does vary based on the scale at different bit rates. As a result of the initial testing, we have more quantitative evidence that for low bit rates, transmission of smaller amounts of data allows less information loss due to quantization. The initial results show that the scaled-PSNR approach is only useful in low-bandwidth situations (here, less than 1mbs) where the loss of information from scaling down would be better off than the effect of using significantly larger quantization on the original image. Additionally, when there is enough bandwidth, it is better to use the original resolution since a comparably smaller quantization scale is used. Thus, the scaled-PSNR based approach was chosen to test Adaptive Image Scaling.

4.2 Dynamic Adaptive Image Scaling Design

In the dynamic Adaptive Image Scaling scheme, the bit rate is monitored and any changes are notified to the encoder. When the encoder detects a change in the bit rate, it must decide whether scaling the image would produce a better picture for the end user. The start of a GOP is used to make adjustments to the bandwidth for simplicity and to gather initial results. Adding finer granularity (e.g., per frame monitoring) can be added at later stages of this research. When the encoder detects a significant change in the bandwidth, the current image size is scaled-down dynamically in an effort to cope with the limited resources. For the current research purposes, to test different scales, the scale factor was specified as a parameter to the encoder. For simplicity, if a new scale is to be used, the encoder closes the current sequence and restarts a new sequence using the scaled-down image.

Another design decision was to dynamically scale (i.e., render) the image as the encoder executes. The execution time for scaling the image was negligible especially

considering the under-utilization of the server CPU [10] as discussed earlier. This decision reduces storage requirements and allows support for live video.

To focus on the effects of image scaling on the encoding process the issue of real-time detection of bandwidth changes and reacting to those changes will not be addressed in this thesis. The contributions in this area [8,13] can be integrated in future works on Adaptive Image Scaling. For experimental purposes, for every GOP, the initial bit rate is reduced by a bandwidth reduction rate. The bandwidth reduction rate was used to simulate dynamic network bandwidth degradation. For example, if the initial bandwidth was specified to be 700kbs and the bandwidth reduction rate was 50%, the first GOP will be encoded using 700kbs, the second GOP will be encoded using 350kbs, the third GOP will use 175kbs, etc.

In the current design, when encountering a lower bit rate, if a smaller image does not yield a better PSNR value, the current image scale is used. In other words, no further scaling is performed. This was done in the interest of addressing the timing issues of encoding. As discussed earlier, it is not realistic to expect an encoder to try multiple image scale factors before proceeding. Thus the encoder must make a quick decision based on its current scale and its next lower size. Note that the actual scaled size must be slightly adjusted to ensure that dimensions meet the other encoder constraints such as macroblock divisibility.

Since the resizing factor changes gradually with only a small difference every time, the above procedure could require multiple-frame delays to converge to an optimal image size. To be able to quickly benefit from a sudden recovery of network bandwidth and also to obtain comparison data, the original reference picture resolution is always

compared to the current and next lower level resolutions. The resolution that yields the best PSNR value will be used. The original picture resolution is preferred when there is enough bandwidth.

To see the effects of image scaling we added an override parameter to the encoder to ignore the dynamic rendering results and simply choose the reference picture resolution at every GOP. In the override mode, the scale factor is used to report potential resolutions that the encoder could have chosen instead of the original resolution.

An example of the proposed dynamic Adaptive Image Scaling is shown in Figure 16. In this scenario, the encoder dynamically scales down the original image (704x240) using a factor of $f=0.5$. The scaled image is encoded and sent over the network. When the decoder receives the scaled image, it will read the display size and scales up the image to the original size. It should be noted that the input to the encoder is the original image (i.e., no other resolutions are stored on disk).

The next chapter provides the experimental results from using the dynamic Adaptive Image Scaling with different bit rates and scale factors. The effects of image scaling will be quantitatively examined and the resulting objective and user-perceived quality metrics will be provided.

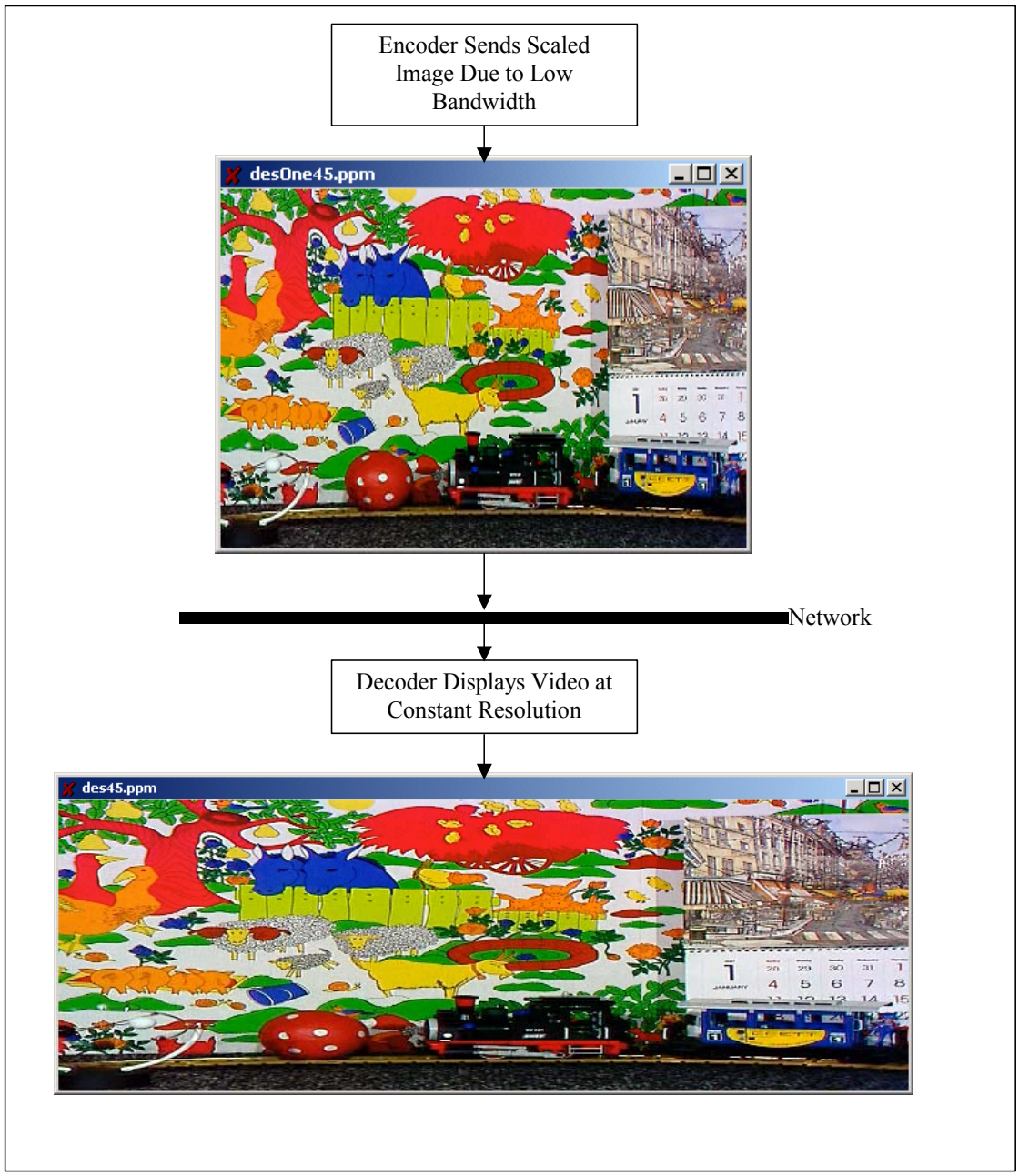

Figure 16. Adaptive Image Scaling Scenario

CHAPTER 5
EXPERIMENTAL RESULTS

Our proposed dynamic rendering encoder was tested with various combinations of the initial bit rate, bandwidth reduction rate, and scale factors. In addition to the *mei20f.m2v* sequence used in the initial testing, the *bbc3_120.m2v* sequence was also tested. Both sequences were downloaded from the MPEG2-Video Test Stream Archive web site [20]. The *bbc3_120.m2v* has a faster rate of motion having dimensions of 704×288. We generalized the data that we gathered into two major cases of behavior: Case 1, when the scaled-down image had a *higher* objective quality and Case 2, when the scaled-down image had a slightly *lower* objective quality.

5.1 Case 1

The dynamic rendering test was run with an initial bit rate of 1.5 mbps, Bandwidth Reduction Rate of 50%, and a Scale Factor of 50%. Using the override flag, the same test was also conducted to see the effects of not using image scaling.

To compare the effects of image scaling on the rest of the frames, the following metrics were compared: PSNR, Quantization Parameter, Target Bits, and Actual Bits. Four GOPs were started using Adaptive Image Scaling. When image scaling was used, the PSNR values from different resolutions at the start of every GOP are listed in Table 1. The best PSNR value is highlighted in the table. Note that in Table 1 the Previous Scale reflects the scale that has being used for the previous GOP (i.e., current scale).

Table 1. Case 1 Adaptive Image Scaling I-Frame PSNR Data

GOP	Bit Rate	Scaled-down Resolution	Previous Scale PSNR (dB)	Original PSNR (dB)	Scaled-down PSNR (dB)
1	1.5mbs	512x192	N/A	24.6	26.0
2	750kbs	352x128	23.0	22.0	24.2
3	375kbs	256x96	21.1	20.7	22.9
4	187.5kbs	176x64	19.4	20.2	21.0

Figure 17 shows frame-by-frame PSNR values of Normal and Adaptive Image Scaled Encoding. The PSNR results show that image scaling produces comparable results. The overall PSNR values for all the frames are relatively close (i.e., our encoder did not lose significant quality though the bit rates have been reduced significantly). All the I-frames (e.g., Frames 0,15,30,45) even have better PSNR values for the scaled-down version of the frames. The maximum PSNR improvement can be in the range of 2 dB.

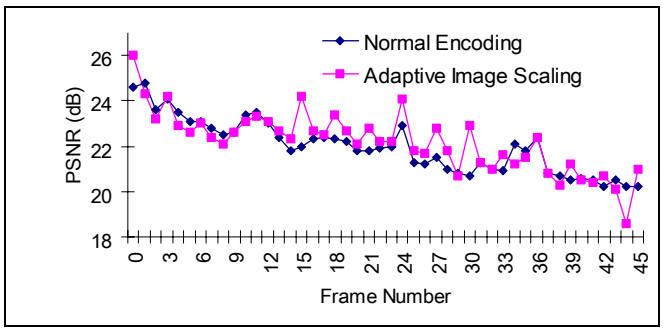

Figure 17. Case 1 Scaled-PSNR Comparison

By analyzing each frame, it can be seen that the initial frames (0-14, associated to the first GOP) have similar PSNR values, while the frames of the second and third GOP (15 - 42) have better PSNR values using Adaptive Image Scaling. However, towards the end of the 3rd GOP (43-45), the PSNR values for Adaptive Image Scaling start to provide lower results (though it is still quite comparable except for the final frame).

As shown in Figure 18, the quantization parameter comparison shows some more reasons for the increase in the original PSNR value. When image scaling is used, a lower

quantization parameter is required. As the encoding of the frames progress with lower bit rates, higher quantization values are used in the normal encoding.

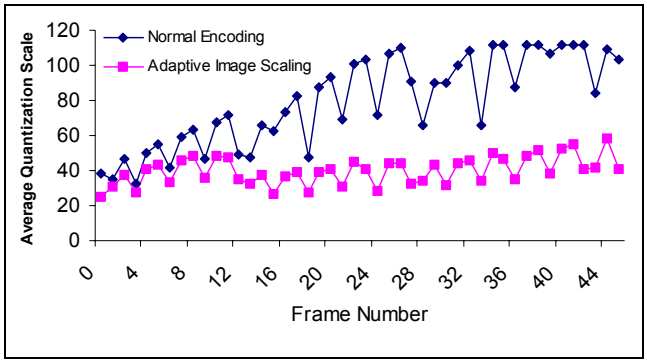

Figure 18. Case 1 Scaled-Quantization Comparison

We observed that at low bandwidths, a scaled-down image using a lower quantization parameter would do better than a non-scaled image using a high quantization parameter. It can be seen that the gap between the quantization parameters required for the image scaling run and the non-image scaling run increases as the encoding encounters lower bit rates.

When image scaling is used, the quantization parameter stays relatively the same with a much smaller variance: 25 - 60. This is not the case for the normal encoding run which has a range of 37 - 112. Lower quantization values produce images with better PSNR. Our proposed scheme generally use 50% to 100% less quantization parameters compared to the normal encoding schemes.

Another metric can be examined with the target bits of each GOP. Figure 19 shows that using normal encoding, the overflow of the GOP target bits per frame gradually increases. It can be seen that as the encoding progresses after a certain point (here, after frame number 21) the buffer overflow in the normal encoding gets worse. Note that at

the start of every GOP, the target bits of the current GOP needs to be reset. We did not observe the similar buffer overflow in our proposed encoder.

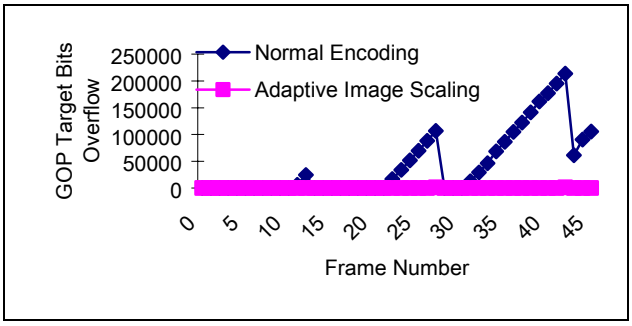

Figure 19. Case 1 GOP Target Bits Overflow Comparison

For Case 1, the scaled-down resolution was always chosen as shown in Table 1. This particular case perhaps represents the best scenario (i.e., that the scaled-down adaptation simply outperformed the normal encoder for all four GOPs). Nevertheless, no encoder can work perfectly for every kind of video content. The following Case 2 represents this possibility.

5.2 CASE 2

It is true that sometimes the Adaptive Image Scaling needs to also consider additional factors. Case 2 was run with an initial bit rate of 700 kbs, Bandwidth Reduction Rate of 50%, and a Scale Factor of 50%. The Case 2 PSNR values from different resolutions at the start of every GOP are listed in Table 2. Using the override flag, a test was conducted to see the effects of dynamic rendering using the mentioned parameters.

Table 2. Case 2 Adaptive Image Scaling I-Frame PSNR Data

GOP	Bit Rate	Scaled-down Resolution	Previous Scale PSNR (dB)	Original PSNR (dB)	Scaled-down PSNR (dB)
1	700kbs	512x192	N/A	22.0	22.6
2	350kbs	352x128	20.8	20.7	20.9
3	175kbs	352x128	19.2	20.3	19.9
4	87.5kbs	256x96	N/A	20.0	19.6

Figure 20 shows the effects of not choosing the scaled-down resolution for the last two GOPS. It can be seen that the Adaptive Image Scaling version had better quantization until the beginning of the 3rd GOP, which is where both runs produce the same quantization values.

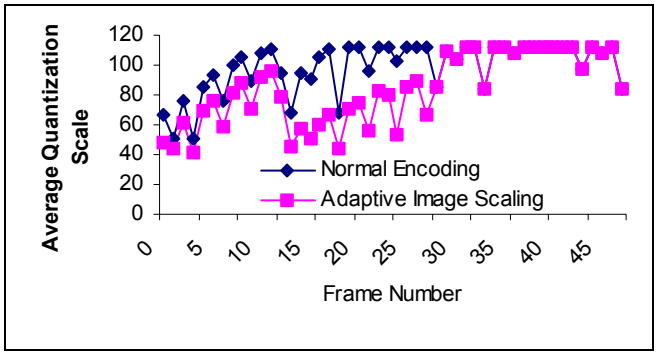

Figure 20. Case 2 Quantization Comparison

Correlation between the image size and the PSNR value can be seen in Figure 21. In Figure 21, for each scale, the bit rate decreases. It can be observed that in higher resolutions, the PSNR of the scaled-down image does relatively same as that of the original resolution. The scaled-down PSNR becomes a better indicator only when the image size decreases below the scale of 0.25. Thus the results suggest that, for this particular video content, the gain is only achieved for 3G wireless data transmission.

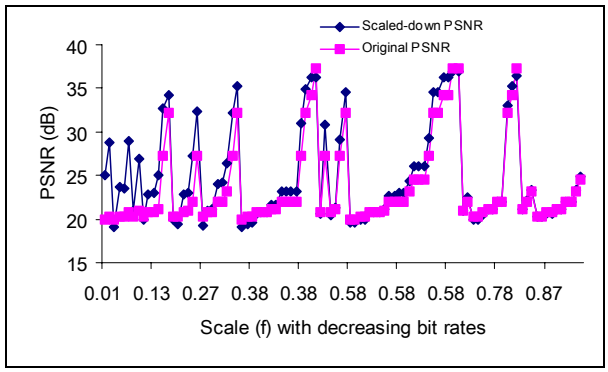

Figure 21. Comparison of Scaled-down PSNRs

For the example of the 3rd I-Frame, Adaptive Image Scaling strictly used the PSNR values to pick the original resolution over the scaled resolution. The differences between the two PSNR values were small: 20.3 vs. 19.9. However, given the 0.4dB loss to the conventional encoders, it is always true that our proposed encoder transmits a smaller amount of video data than the conventional ones.

From the purely PSNR consideration, the following are the deciding parameters used at the start of each GOP. For the first 2 GOPs, the scaled-down resolution was picked. For the 3rd and 4th GOP, the original picture had the better PSNR and thus it was chosen.

Case 2 deserves some further analysis since the decision is no more clear-cut. It is arguable that perhaps the small PSNR variation can be tolerable, while saving the bandwidth is always a design plus for the overall networks. To show that small variations in the scaled-PSNR still provide comparable pictures, we provided the pictures for subjective quality measurements among some human testers.

The pictures were taken from the right side of the video frames (see Appendix D). The reference picture was encoded with bit rate of 60 mbps (i.e., highest quality) and as a result used 1425311 bits with a PSNR of 37.3dB. However, the other two pictures were taken from the 3rd I-Frames of Case 2 above (encoded using a significantly lower bit rate of 175kbs) and as their PSNR values indicate (20.3dB vs. 19.9dB), the visual quality is reported by the testers to be also very comparable. Since at the low bit rates the quality of both images suffers, choosing the scaled-down version does provide comparable results with more than half the bit requirements. Thus, it is better to choose the slightly lower PSNR valued image and reduce the bandwidth requirement.

5.3 Further Analysis

From dynamic image scaling we find that for low bit rates, when the PSNR values of the original and scaled-down frames are close, it is better to encode using the scaled-down resolution. Base on this finding, we further investigated encoding using different scales at low (i.e., 700kbs or lower), constant bit rates. The findings for the fourth I-Frame (45^{th} frame) of both sequences are listed in Table 3 and Table 4. At low bandwidths, we tested the scaled-down PSNR values for the following scales: 0.25, 0.50 and 0.75.

Table 3. Frame 45 Testing Of Sequence mei20f.m2v

Bit Rate	Optimal Scale	Scaled-PSNR Gain	Recommended Scale (Tolerance:0.5)	Scaled-PSNR Gain
700kbs	0.25	1.2	0.25	1.2
300kbs	0.5	-0.1	0.25	-0.5
64kbs	0.75	-0.2	0.5	-0.4

Table 4. Frame 45 Testing Of Sequence bbc3_120.m2v

Bit Rate	Optimal Scale	Scaled-PSNR Gain	Recommended Scale (Tolerance:1.5)	Scaled-PSNR Gain
700kbs	0.75	0.4	0.25	-0.4
300kbs	0.75	0.1	0.5	-1.0
64kbs	0.75	-0.2	0.5	-1.6

The results suggest that scaling down to at least the scale of 0.75, would produce as good or better results than using the original resolution.

In Table 3 and Table 4, the Scaled-PSNR Gain is the gain when compared to the original PSNR. The Recommended Scale should be used to achieve minimum quality requirements using a tolerance for the scaled-PSNR loss. The exact values vary based on the image complexity of the sequence used. We can see that the *bbc3_120.m2v* sequence requires a higher tolerance to use the recommended scales and its optimal scale is higher. Sequence having a higher rate of motion and detail will have higher *rate-distortion*

values [16]. The next chapter draws conclusions regarding the results gathered from this thesis.

CHAPTER 6
CONCLUSION

In this thesis, we presented a study for a novel design and implementation of bandwidth-aware Adaptive Image Scaling for MPEG-2/4 encoding. The strength of the scheme is that this mechanism can reduce the network traffic during congestion situations and still obtain comparable quality. From the experimental results, it was observed that using a small PSNR tolerance provided better results than using a zero-tolerance criteria. This work proposes a complimentary scheme that integrates well with adaptive quantization and other methods of degrading the video's fidelity.

6.1 Contributions

Even though works such as Fox et al. [4] address resolution scaling as an option to adjusting the data fidelity, this thesis has performed a thorough investigation on the effects of MPEG-2 dynamic Adaptive Image Scaling. The most significant contribution from this thesis is that we understand that under low bandwidth requirements, image scaling can produce as good or better quality video with significantly less bits than using the original resolution. Furthermore, this thesis uniquely provides an independent theoretical model which, when tested, backs the experimental results. Both the theoretical and experimental results show that there is an optimal scale factor that will provide the minimum distortion.

6.2 Future Work

All the VBR experiments that were done for this thesis assumed a decreasing bit rate to focus on low bandwidth behavior. This thesis can be extended to handle random

bit rate fluctuations. Though a two-pass approach was used to study the effects of image scaling with that of the reference, an integrated one-pass approach can be implemented by utilizing an optimal algorithm using rules that will determine the appropriate fidelity based on factors such as PSNR, bit rate, and scale.

APPENDIX A
PERTINENT MPEG-2 ENCODER SOURCE CODE CHANGES

(Note: Modified code from MPEG Software Simulation Group [21] is in **boldface**.)

mpeg2enc.c

```
/* mpeg2enc.c, main() and parameter file reading                            */
/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */
/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#define GLOBAL /* used by global.h */
#include "config.h"
#include "global.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

#define MAXSERVERDATASIZE 80
#define MAXHOSTNAME       80
#define MYPORT            5000    // the port users will be connecting to
#define BACKLOG           5     // how many pending connections queue will hold
#define MAXCLIENTS        4
#define MAXTITLES         10
#define MAXTITLELENGTH    15

/* pointer to a signal handler */
```

```
typedef void (*sighandler_t)(int);

int clientcount = 0;


/* private prototypes */
static void readparmfile _ANSI_ARGS_((char *fname));
static void readquantmat _ANSI_ARGS_((void));
static int openConnection _ANSI_ARGS_((void));

// control c signal handler
static void CtrlC()
{
    printf ("<server>: Handling CONTROL-C\n");
    printf ("<server>: Server is going DOWN!\n");
    close(newSockfd);
    close(sockfd);

    fclose(outfile);
    fclose(statfile);
}

// signal handler for fork'ed process termination
static void ChildTerm()
{
  clientcount--;
  printf("<server>: child has died\n");
  printf("<server>: clientcount is %d \n", clientcount);

}



int main(argc,argv)
int argc;
char *argv[];
{
  if (argc!=3)
  {
    printf("\n%s, %s\n",version,author);
    printf("Usage: mpeg2encode in.par out.m2v\n");
    exit(0);
  }

  /* read parameter file */
  readparmfile(argv[1]);

  printf("Reading quantization matrices....\n");
  /* read quantization matrices */
  readquantmat();

  /* open output file */
  if (!(outfile=fopen(argv[2],"wb")))
  {
    sprintf(errortext,"Couldn't create output file %s",argv[2]);
    error(errortext);
  }

  printf("Openning Connection ....\n");
  /* open connection */
  openConnection();
  return 0;
}

static int openConnection()
{
    struct sockaddr_in serverAddr;    // my address information
    struct sockaddr_in clientAddr;    // connector's address information
    struct hostent     *hp;
    int                count, addrlen, nbytes;
    char               myname[MAXHOSTNAME+1];
```

```
char               serverBuffer[MAXSERVERDATASIZE];
int                i, selections;
char               *bitStream;
int                spaceindex, userIndex;

//    clientcount = 0;
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("socket");
    exit(1);
}
memset(&serverAddr, 0, sizeof(struct sockaddr_in));  // clear our address

if (gethostname(myname, MAXHOSTNAME) == -1)  // who are we...
{
    perror("socket");
    exit(1);
}

if ((hp= (struct hostent *)gethostbyname(myname)) == NULL)   // get our address info
{
    perror("gethostbyname");
    exit(-1);  // per tutorial -1
}


serverAddr.sin_family      = AF_INET;
serverAddr.sin_port        = htons(port_number);     // short, network byte order
if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr))== -1)
{
    perror("<server>: bind");
    exit(1);
}

printf("<server>: Server is UP at %s, port %d\n",
       myname, port_number);

// signal hanler for control c --> close the socket!
signal(SIGINT, CtrlC);

if (listen(sockfd, BACKLOG) == -1)
{
    printf("<server>: listen failure %d\n",errno);
    perror("<server>: listen");
    exit(1);
}

// ignore child process termination
//    signal (SIGCHLD, SIG_IGN);
signal (SIGCHLD, ChildTerm);

clientcount = 0;
while(1) // main accept() loop
{

  addrlen = sizeof(serverAddr);

  if (clientcount < MAXCLIENTS)
  {
    printf("<server>: waiting to accept new connection \n");
    if ((newSockfd = accept(sockfd, &serverAddr, &addrlen)) == -1)
    {
      printf("<server>: accept failure %d\n",errno);
      perror("<server>: accept");
       exit(1);
    }
    printf("<server>: got new connection from %s\n",
           inet_ntoa(clientAddr.sin_addr));
    switch(fork())
    {                  /* try to handle connection */
      case -1 :                         /* bad news.  scream and die */
```

```
          perror("<server>: fork");
          close(sockfd);
          close(newSockfd);
          exit(1);
        case 0 :                          /* we're the child, do something */
          close(sockfd);      // close the parent socket fd

          //1. read request for list of titles
          if ((nbytes = read(newSockfd, serverBuffer, MAXSERVERDATASIZE)) < 0)
          {
            perror("<child>: read");
          }
          if (nbytes != 19)
          {
            printf("Error -- could not read request for titles, btyes read is %d\n",
nbytes);
          }
          printf("<child>: Read request for titles, btyes read is %d\n", nbytes);

          printf("<child>: Ready to send bitstream for decoding\n");
          init(horizontal_size, vertical_size);
          putseq();
          break;
        default :                         /* we're the parent so look for */
          close(newSockfd);
          clientcount++;
          printf("<server>: clientcount is %d \n", clientcount);
          continue;
      };
    }
  }
}

void init(int inputH_size, int inputV_size)
{
  int i, size;
  static int block_count_tab[3] = {6,8,12};
  static int firstTime = 1;

  horizontal_size = inputH_size;
  vertical_size   = inputV_size;
  imageScaleTestBufCnt = 0;

  range_checks();
  profile_and_level_checks();


  /* Clip table */
  if (!(Clip=(unsigned char *)malloc(1024)))
    Error("Clip[] malloc failed\n");

  Clip += 384;

  for (i=-384; i<640; i++)
    Clip[i] = (i<0) ? 0 : ((i>255) ? 255 : i);


  initbits();
  init_fdct();
  init_idct();

  /* round picture dimensions to nearest multiple of 16 or 32 */
  mb_width = (horizontal_size+15)/16;
  mb_height = prog_seq ? (vertical_size+15)/16 : 2*((vertical_size+31)/32);
  mb_height2 = fieldpic ? mb_height>>1 : mb_height; /* for field pictures */
  width = 16*mb_width;
  height = 16*mb_height;

  chrom_width = (chroma_format==CHROMA444) ? width : width>>1;
  chrom_height = (chroma_format!=CHROMA420) ? height : height>>1;
```

```
height2 = fieldpic ? height>>1 : height;
width2 = fieldpic ? width<<1 : width;
chrom_width2 = fieldpic ? chrom_width<<1 : chrom_width;

block_count = block_count_tab[chroma_format-1];

/* clip table */
if (!(clp = (unsigned char *)malloc(1024)))
  error("malloc failed\n");
clp+= 384;
for (i=-384; i<640; i++)
  clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

for (i=0; i<3; i++)
{
  size = (i==0) ? width*height : chrom_width*chrom_height;

  if (refHsize == horizontal_size && refVsize == vertical_size)
    refSize[i] = size;
  if (!firstTime)
  {
    free(newrefframe[i]);
    free(oldrefframe[i]);
    free(auxframe[i]);
    free(neworgframe[i]);
    free(oldorgframe[i]);
    free(auxorgframe[i]);
    free(predframe[i]);
    free(tempframe[i]);
    free(temp2frame[i]);
  }
  if (!(newrefframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(oldrefframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(auxframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(neworgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(oldorgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(auxorgframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(predframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(tempframe[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
  if (!(temp2frame[i] = (unsigned char *)malloc(size)))
    error("malloc failed\n");
}

mbinfo = (struct mbinfo *)malloc(mb_width*mb_height2*sizeof(struct mbinfo));

if (!mbinfo)
  error("malloc failed\n");

blocks =
  (short (*)[64])malloc(mb_width*mb_height2*block_count*sizeof(short [64]));

if (!blocks)
  error("malloc failed\n");

// should not recreate a new file on multiple call (just open and append to it!)

/* open statistics output file */
if (firstTime)
{
  if (statname[0]=='-')
    statfile = stdout;
  else if (!(statfile = fopen(statname,"w")))
  {
```

```
      sprintf(errortext,"Couldn't create statistics output file %s",statname);
      error(errortext);
    }
  }
  if (firstTime)
    firstTime = 0;

}

void error(text)
char *text;
{
  fprintf(stderr,text);
  putc('\n',stderr);
  exit(1);
}

static void readparmfile(fname)
char *fname;
{
  int i;
  int h,m,s,f;
  FILE *fd;
  char line[256];
  static double ratetab[8]=
    {24000.0/1001.0,24.0,25.0,30000.0/1001.0,30.0,50.0,60000.0/1001.0,60.0};
  extern int r,Xi,Xb,Xp,d0i,d0p,d0b; /* rate control */
  extern double avg_act; /* rate control */

  if (!(fd = fopen(fname,"r")))
  {
    sprintf(errortext,"Couldn't open parameter file %s",fname);
    error(errortext);
  }

  fgets(id_string,254,fd);
  fgets(line,254,fd); sscanf(line,"%s",tplorg);
  printf("tplorg = %s\n",tplorg);
  fgets(line,254,fd); sscanf(line,"%s",tplref);
  fgets(line,254,fd); sscanf(line,"%s",iqname);
  fgets(line,254,fd); sscanf(line,"%s",niqname);
  fgets(line,254,fd); sscanf(line,"%s",statname);
  fgets(line,254,fd); sscanf(line,"%d",&inputtype);
  fgets(line,254,fd); sscanf(line,"%d",&nframes);
  fgets(line,254,fd); sscanf(line,"%d",&frame0);
  fgets(line,254,fd); sscanf(line,"%d:%d:%d:%d",&h,&m,&s,&f);
  fgets(line,254,fd); sscanf(line,"%d",&N);
  fgets(line,254,fd); sscanf(line,"%d",&M);
  fgets(line,254,fd); sscanf(line,"%d",&mpeg1);
  fgets(line,254,fd); sscanf(line,"%d",&fieldpic);
  fgets(line,254,fd); sscanf(line,"%d",&horizontal_size);
  fgets(line,254,fd); sscanf(line,"%d",&vertical_size);
  refHsize = horizontal_size;
  refVsize = vertical_size;
  fgets(line,254,fd); sscanf(line,"%d",&aspectratio);
  fgets(line,254,fd); sscanf(line,"%d",&frame_rate_code);
  fgets(line,254,fd); sscanf(line,"%lf",&bit_rate);
  fgets(line,254,fd); sscanf(line,"%lf",&bwReductionRate);
  fgets(line,254,fd); sscanf(line,"%lf",&scaleFactor);
  fgets(line,254,fd); sscanf(line,"%lf",&overrideImageScaleResults);
  fgets(line,254,fd); sscanf(line,"%lf",&staticMode);
  fgets(line,254,fd); sscanf(line,"%d",&port_number);
  fgets(line,254,fd); sscanf(line,"%d",&vbv_buffer_size);
  fgets(line,254,fd); sscanf(line,"%d",&low_delay);
  fgets(line,254,fd); sscanf(line,"%d",&constrparms);
  fgets(line,254,fd); sscanf(line,"%d",&profile);
  fgets(line,254,fd); sscanf(line,"%d",&level);
  fgets(line,254,fd); sscanf(line,"%d",&prog_seq);
  fgets(line,254,fd); sscanf(line,"%d",&chroma_format);
  fgets(line,254,fd); sscanf(line,"%d",&video_format);
  fgets(line,254,fd); sscanf(line,"%d",&color_primaries);
```

```
  fgets(line,254,fd); sscanf(line,"%d",&transfer_characteristics);
  fgets(line,254,fd); sscanf(line,"%d",&matrix_coefficients);
  fgets(line,254,fd); sscanf(line,"%d",&display_horizontal_size);
  fgets(line,254,fd); sscanf(line,"%d",&display_vertical_size);
  fgets(line,254,fd); sscanf(line,"%d",&dc_prec);
  fgets(line,254,fd); sscanf(line,"%d",&topfirst);
  fgets(line,254,fd); sscanf(line,"%d %d %d",
    frame_pred_dct_tab,frame_pred_dct_tab+1,frame_pred_dct_tab+2);

  fgets(line,254,fd); sscanf(line,"%d %d %d",
    conceal_tab,conceal_tab+1,conceal_tab+2);

  fgets(line,254,fd); sscanf(line,"%d %d %d",
    qscale_tab,qscale_tab+1,qscale_tab+2);

  fgets(line,254,fd); sscanf(line,"%d %d %d",
    intravlc_tab,intravlc_tab+1,intravlc_tab+2);
  fgets(line,254,fd); sscanf(line,"%d %d %d",
    altscan_tab,altscan_tab+1,altscan_tab+2);
  fgets(line,254,fd); sscanf(line,"%d",&repeatfirst);
  fgets(line,254,fd); sscanf(line,"%d",&prog_frame);
/* intra slice interval refresh period */
  fgets(line,254,fd); sscanf(line,"%d",&P);
  fgets(line,254,fd); sscanf(line,"%d",&r);
  fgets(line,254,fd); sscanf(line,"%lf",&avg_act);
  fgets(line,254,fd); sscanf(line,"%d",&Xi);
  fgets(line,254,fd); sscanf(line,"%d",&Xp);
  fgets(line,254,fd); sscanf(line,"%d",&Xb);
  fgets(line,254,fd); sscanf(line,"%d",&d0i);
  fgets(line,254,fd); sscanf(line,"%d",&d0p);
  fgets(line,254,fd); sscanf(line,"%d",&d0b);

  if (N<1)
    error("N must be positive");
  if (M<1)
    error("M must be positive");
  if (N%M != 0)
    error("N must be an integer multiple of M");

  motion_data = (struct motion_data *)malloc(M*sizeof(struct motion_data));
  if (!motion_data)
    error("malloc failed\n");

  for (i=0; i<M; i++)
  {
    fgets(line,254,fd);
    sscanf(line,"%d %d %d %d",
      &motion_data[i].forw_hor_f_code, &motion_data[i].forw_vert_f_code,
      &motion_data[i].sxf, &motion_data[i].syf);

    if (i!=0)
    {
      fgets(line,254,fd);
      sscanf(line,"%d %d %d %d",
        &motion_data[i].back_hor_f_code, &motion_data[i].back_vert_f_code,
        &motion_data[i].sxb, &motion_data[i].syb);
    }
  }

  fclose(fd);
...
```

putseq.c

```
/* putseq.c, sequence level routines                                        */

/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */

/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

#include <stdio.h>
#include <string.h>
#include "config.h"
#include "global.h"
#include <Ilib.h>

void outputMessage()
{
  printf(message);
  fprintf(statfile,message);
}

int scaleImage(tinfile, toutfile, twidth, theight, fnum, startIndex)
char *tinfile;
char *toutfile;
int  twidth,   theight, fnum, startIndex;
{
  IGC igc;
  IImage image,nimage;
  char outfile[30];
  IFileFormat input_format = IFORMAT_PPM;
  IFileFormat output_format = IFORMAT_PPM;
  char infile[30];
  FILE *fp;
  IError ret;
  int loop;
  int i;

  for (i=startIndex;i<(fnum+startIndex);i++)
  {
    sprintf(infile,"%s%d.%s",tinfile,i,"ppm");
    sprintf(outfile,"%s%d.%s",toutfile,i,"ppm");
    sprintf(message,"Converting from %s to %s\n",infile,outfile);
    outputMessage();
    if ( ! infile )
      fprintf ( stderr, "No infile specified.  Reading from stdin.\n" );
    if ( ! outfile )
    {
      strcpy(outfile , "out.ppm");
```

```
      fprintf ( stderr, "No outfile specified.  Writing to %s.\n", outfile );
    }

    /* try and determine file types by extension */
    if ( infile )
    {
      ret = IFileType ( infile, &input_format );
      if ( ret )
      {
       fprintf ( stderr, "Input file error: %s\n", IErrorString ( ret ) );
       exit ( 1 );
      }
    }
    if ( outfile )
    {
      ret = IFileType ( outfile, &output_format );
      if ( ret )
      {
       fprintf ( stderr, "Output file error: %s\n", IErrorString ( ret ) );
       fprintf ( stderr, "Using PPM format.\n" );
      }
    }

    if ( infile )
    {
      fp = fopen ( infile, "rb" );
      if ( ! fp )
      {
       perror ( "Error opening input file:" );
       exit ( 1 );
      }
    }
    else
      fp = stdin;

    if ( ( ret = IReadImageFile ( fp, input_format, IOPTION_NONE, &image ) ) )
    {
      fprintf ( stderr, "Error reading image: %s\n", IErrorString ( ret ) );
      exit ( 1 );
    }
    if ( infile )
      fclose ( fp );
    igc=ICreateGC ( );
    nimage=ICreateImage(twidth,theight,IOPTION_NONE);
    ICopyImageScaled ( image,nimage,igc,0,0,IImageWidth(image),
                       IImageHeight(image),0,0,twidth,theight);
    if ( outfile )
    {
      fp = fopen ( outfile, "wb" );
      if ( ! fp )
      {
       perror ( "Cannot open output file: " );
       exit ( 1 );
      }
    }
    else
      fp = stdout;

    IWriteImageFile ( fp, nimage, output_format, IOPTION_INTERLACED );

    if ( outfile )
      fclose ( fp );
  }
  return ( 0 );
}

void putseq()
{
  /* this routine assumes (N % M) == 0 */
  int i, j, k, f, f0, pf0, n, np, nb, sxf, syf, sxb, syb;
  int ipflag;
```

```
  FILE *fd;
  char name[256];
  unsigned char *neworg[3], *newref[3];
  static char ipb[5] = {' ','I','P','B','D'};

  struct snr_data snrVals; // Val1 = snrvals in ref section
  int scaleDir = 0; // -1 = down, 0 = steady, 1 = up
  int mb_widthTemp, mb_heightTemp;
  double prev_bit_rate = bit_rate, currScale = 1.0, testScale = 1.0;
  char refFrameName[256], currFrameName[256];
  char baseRefFrameName[256], baseScaledFrameName[256], tplorgTemp[256],
scaledUpFrameName[256];
  char smallEncStoredFName[256], scaledUpSmallEncStoredFName[256];
  char smallEncStoredFNameBase[256], scaledUpSmallEncStoredFNameBase[256];
  int  currHsize = horizontal_size, currVsize = vertical_size;
  int  prevHsize = horizontal_size, prevVsize = vertical_size;
  int  testHsize = horizontal_size, testVsize = vertical_size;
  int   bestYSnrLevel = 0;
  float bestYSnr = 0.0, level1YSnr = 0.0, level2YSnr = 0.0, level3YSnr = 0.0,
scldNormalYSnr;;
  int   firstTimeInLevel1 = 1;
  char  tempFileName[30];

  strcpy(refFrameName, tplorg);
  strcpy(currFrameName, tplorg);
  //set Testing Flag = false
  imageScaleTesting = 0;
  testLevel = 0;
  initFlag = 1;
  refSnrPass = 0;
  strcpy(tplorgTemp,tplorg);
  strcpy(baseRefFrameName, strtok(tplorgTemp,"%"));  //gets "des" out of "des%d"
  sprintf(message,"baseFrameName = %s\n",baseRefFrameName);
  outputMessage();


  sprintf(message, //put the text in the opposite coloumn
"DDATA,-,Level,-,display#,-,frame,-,Type,-,Dim,-,Area,-,Bit Rate,-,S,-,TargetBits,-
,GOPOverflow,-,Q,-,YSnr,-,Level1Snr,-,Level1S,-,Level1Q,-,Scale\n");
  outputMessage();

  /* loop through all frames in encoding/decoding order */
  for (i=0; i<nframes; i++)
  {
    pf0 = N*((i+(M-1))/N) - (M-1); // used to peek if the current frame is an I frame
    if (pf0<0)
      pf0=0;
    sprintf(message,"pf0 = %d\n", pf0);
    outputMessage();

    //    Assume the following unless overridden in the I frame testing:
    testLevel = MaxTestLevels + 1;  // skip all test levels
    imageScaleTesting = 0;          // do it for real

    bestYSnrLevel = 0;
    bestYSnr = 0;
    level1YSnr = 0;
    level2YSnr = 0;
    level3YSnr = 0;

    if (pf0 == i) //@ every I frame, need to determine to re-scale
    {
      sprintf(message,"DD Start OF GOP f0 = %d-----------------------\n",pf0);
      outputMessage();
      if (i != 0) // execpt for the 1st gop do:
      {
        prev_bit_rate = bit_rate;     // save previously used bitrate
       bit_rate  *= bwReductionRate; // get current bit rate
      }

      if (bit_rate == prev_bit_rate)
```

```
        scaleDir = 0; // steady
      else if (bit_rate > prev_bit_rate)
        scaleDir = 1; // increasing
      else // decreasing
        scaleDir = -1;

      if ((bwReductionRate == 1.0) && (staticMode == 1))
        scaleDir = -1; // assume scale down

      if ((scaleDir != 0) || (i == 0)) // need to test if adaptive scaling needs to be
done
      {                                // if there is a bandwidth change or if initial I-
frame
        if (i != 0)
          putseqend();    // close the current sequence
        initFlag = 1;     // this flag will be set to 0 after initialization
        testLevel = 1;    // start at base test
        imageScaleTesting = 1;
        if (i == 0)
          scaleDir = -1;  // for the first I-frame test to see if scale down is better
than ref.
      }
    }

    // before potential testing, save the current size
    prevHsize = currHsize;
    prevVsize = currVsize;
    testHsize = currHsize;
    testVsize = currVsize;
    refSnrPass = 0; // this flag should be set when ref SNR is to be done
    while (testLevel > 0)
    {
      imageScaleTestBufCnt = 0;
      if ((testLevel == 2) && (prevHsize == refHsize) && (prevVsize == refVsize))
//verify if level 2 is required
        testLevel++;

      sprintf(message,"--->Level = %d, i = %d, ref SNR pass = %d\n", testLevel, i,
refSnrPass);
      outputMessage();

      if (imageScaleTesting == 1)  // if you enter the loop with testing in mind,
initialiation is required
        initFlag = 1;

      if (testLevel == 1)          // level 1 testing - get values for reference
      {
        currHsize = refHsize;
        currVsize = refVsize;
        // set to refernce frame name & size
        strcpy(tplorg, refFrameName);
      }
      else if (testLevel == 2)     // level 2 testing - get values for curr scale
      {
        currHsize = prevHsize;
        currVsize = prevVsize;
        testHsize = currHsize;
        testVsize = currVsize;
      }
      else if (testLevel == 3)     // level 3 testing - scale up/down
      {
        if (scaleDir == 1) // increasing
        {
          testScale = currScale + currScale * scaleFactor;
          if (testScale > 1.0) // floor it to the reference size & in this case no need to
run level 3 testing !!!
            testScale = 1.0;   // POSTPONING since this case is not being dealt w/ in this
THESIS (i.e., assume decreasing)
        }
        else if (scaleDir == -1)  // decreasing
         {
```

```
          testScale = currScale - currScale * scaleFactor;
        }
        // need to adjust ref by the scale
        currHsize = (int) (refHsize * sqrt(testScale));
        currVsize = (int) (refVsize * sqrt(testScale));

        if (currHsize % 2 != 0)
          currHsize += 1;
        if (currVsize % 2 != 0)
          currVsize += 1;

        mb_widthTemp = (currHsize+15)/16;
        mb_heightTemp = prog_seq ? (currVsize+15)/16 : 2*((currVsize+31)/32);
        currHsize = 16*mb_widthTemp;
        currVsize = 16*mb_heightTemp;

        testHsize = currHsize;
        testVsize = currVsize;
        sprintf(message,"testScale = %lf, scaled H = %d, scaled V = %d\n", testScale,
currHsize, currVsize);
        outputMessage();
      }
      else if (testLevel == MaxTestLevels + 1) //the real run
      {
        if (imageScaleTesting == 1) // if we are in testing mode, pick the best SNR level
        {
          if ((bestYSnrLevel == 1) || (overrideImageScaleResults == 1))
          {
            if (overrideImageScaleResults == 1)
            {
              sprintf (message,"****OVERRIDE ImageScaline ON\n");
              outputMessage();
            }
            currHsize = refHsize;
            currVsize = refVsize;
            strcpy(tplorg, refFrameName);
            currScale = 1.0; // since reference is chosen, scale is set to 1
          }
          else if (bestYSnrLevel == 2)
          {
            currHsize = prevHsize;
            currVsize = prevVsize;
            //currScale is unchanged
          }
          else if (bestYSnrLevel == 3)
          {
            currHsize = testHsize;
            currVsize = testVsize;
            currScale = testScale;
          }
          sprintf (message,"****The best Normal Y SNR is found at Level %d Testing with
value of %3.3g\n", bestYSnrLevel, bestYSnr);
          outputMessage();
          sprintf (message,"SUMMARY of Y SNRS: Level 1: %3.3g, Level 2: %3.3g, Level 3:
%3.3g\n", level1YSnr, level2YSnr, level3YSnr);
          outputMessage();

        }
        imageScaleTesting = 0;
      }


      if ((currHsize != refHsize) || (currVsize != refVsize)) //if not ref
      {
        //construct the name from the current scaled size: hxw<refBase>%d
        sprintf(tplorg,"%dx%d%s",currHsize,currVsize,refFrameName);
        strcpy(tplorgTemp,tplorg);
        strcpy(baseScaledFrameName, strtok(tplorgTemp,"%"));
      }

      if (initFlag == 1)
```

```
{
  init(currHsize, currVsize);

  rc_init_seq(); /* initialize rate control */

  /* sequence header, sequence extension and sequence display extension */
  putseqhdr();
  if (!mpeg1)
  {
    putseqext();
    putseqdispext();
  }

  /* optionally output some text data (description, copyright or whatever) */
  if (strlen(id_string) > 1)
    putuserdata(id_string);

  initFlag = 0; // this flag must be set to 1 on a need basis
}
if (!quiet)
{
  fprintf(stderr,"Encoding frame %d ",i);
  fflush(stderr);
}
/* f0: lowest frame number in current GOP
 *
 * first GOP contains N-(M-1) frames,
 * all other GOPs contain N frames
 */
f0 = N*((i+(M-1))/N) - (M-1);

if (f0<0)
  f0=0;

if (i==0 || (i-1)%M==0)
{
  /* I or P frame */
  for (j=0; j<3; j++)
  {
    /* shuffle reference frames */
    neworg[j] = oldorgframe[j];
    newref[j] = oldrefframe[j];
    oldorgframe[j] = neworgframe[j];
    oldrefframe[j] = newrefframe[j];
    neworgframe[j] = neworg[j];
    newrefframe[j] = newref[j];
  }

  /* f: frame number in display order */
  f = (i==0) ? 0 : i+M-1;
  if (f>=nframes)
    f = nframes - 1;

  if (i==f0) /* first displayed frame in GOP is I */
  {
    /* I frame */
    pict_type = I_TYPE;
    forw_hor_f_code = forw_vert_f_code = 15;
    back_hor_f_code = back_vert_f_code = 15;

    /* n: number of frames in current GOP
     *
     * first GOP contains (M-1) less (B) frames
     */
    n = (i==0) ? N-(M-1) : N;

    /* last GOP may contain less frames */
    if (n > nframes-f0)
      n = nframes-f0;

    /* number of P frames */
```

```
    if (i==0)
      np = (n + 2*(M-1))/M - 1; /* first GOP */
    else
      np = (n + (M-1))/M - 1;

    /* number of B frames */
    nb = n - np - 1;

    rc_init_GOP(np,nb);

    putgophdr(f0,i==0); /* set closed_GOP in first GOP only */
  }
  else
  {
    /* P frame */
    pict_type = P_TYPE;
    forw_hor_f_code = motion_data[0].forw_hor_f_code;
    forw_vert_f_code = motion_data[0].forw_vert_f_code;
    back_hor_f_code = back_vert_f_code = 15;
    sxf = motion_data[0].sxf;
    syf = motion_data[0].syf;
  }
}
else
{
  /* B frame */
  for (j=0; j<3; j++)
  {
    neworg[j] = auxorgframe[j];
    newref[j] = auxframe[j];
  }

  /* f: frame number in display order */
  f = i - 1;
  pict_type = B_TYPE;
  n = (i-2)%M + 1; /* first B: n=1, second B: n=2, ... */
  forw_hor_f_code = motion_data[n].forw_hor_f_code;
  forw_vert_f_code = motion_data[n].forw_vert_f_code;
  back_hor_f_code = motion_data[n].back_hor_f_code;
  back_vert_f_code = motion_data[n].back_vert_f_code;
  sxf = motion_data[n].sxf;
  syf = motion_data[n].syf;
  sxb = motion_data[n].sxb;
  syb = motion_data[n].syb;
}

temp_ref = f - f0;
frame_pred_dct = frame_pred_dct_tab[pict_type-1];
q_scale_type = qscale_tab[pict_type-1];
intravlc = intravlc_tab[pict_type-1];
altscan = altscan_tab[pict_type-1];

fprintf(statfile,"\nFrame %d (#%d in display order):\n",i,f);
fprintf(statfile," picture_type=%c\n",ipb[pict_type]);
fprintf(statfile," temporal_reference=%d\n",temp_ref);
fprintf(statfile," frame_pred_frame_dct=%d\n",frame_pred_dct);
fprintf(statfile," q_scale_type=%d\n",q_scale_type);
fprintf(statfile," intra_vlc_format=%d\n",intravlc);
fprintf(statfile," alternate_scan=%d\n",altscan);

if (pict_type!=I_TYPE)
{
  fprintf(statfile," forward search window: %d...%d / %d...%d\n",
          -sxf,sxf,-syf,syf);
  fprintf(statfile," forward vector range: %d...%d.5 / %d...%d.5\n",
          -(4<<forw_hor_f_code),(4<<forw_hor_f_code)-1,
          -(4<<forw_vert_f_code),(4<<forw_vert_f_code)-1);
}

if (pict_type==B_TYPE)
{
```

```
        fprintf(statfile," backward search window: %d...%d / %d...%d\n",
                -sxb,sxb,-syb,syb);
        fprintf(statfile," backward vector range: %d...%d.5 / %d...%d.5\n",
                -(4<<back_hor_f_code),(4<<back_hor_f_code)-1,
                -(4<<back_vert_f_code),(4<<back_vert_f_code)-1);
      }

      sprintf(name,tplorg,f+frame0);
      sprintf(message,"\nReading frame from name = %s\n", name);
      outputMessage();
      sprintf(message,"Horizontal Size = %d, Vertical Size = %d, Bit Rate = %lf\n",
horizontal_size, vertical_size, bit_rate);
      outputMessage();

      if ((currHsize != refHsize) || (currVsize != refVsize)) //if not ref
      {
        scaleImage(baseRefFrameName, baseScaledFrameName, currHsize, currVsize, 1,
f+frame0); // for the next i ?? f+frame0???
      }

      readframe(name,neworg);

      sprintf(tempFileName,"neworgL%d%s\0",testLevel,name);
      store_ppm_tga(tempFileName,neworg,0,horizontal_size,vertical_size,0);

      // FEILD PICTURES ARE NOT SUPPORTED
      pict_struct = FRAME_PICTURE;

      /* do motion_estimation
       *
       * uses source frames (...orgframe) for full pel search
       * and reconstructed frames (...refframe) for half pel search
       */

      motion_estimation(oldorgframe[0],neworgframe[0],
                        oldrefframe[0],newrefframe[0],
                        neworg[0],newref[0],
                        sxf,syf,sxb,syb,mbinfo,0,0);

      predict(oldrefframe,newrefframe,predframe,0,mbinfo);
      dct_type_estimation(predframe[0],neworg[0],mbinfo);
      transform(predframe,neworg,mbinfo,blocks);

      putpict(neworg[0]);

      for (k=0; k<mb_height*mb_width; k++)
      {
        if (mbinfo[k].mb_type & MB_INTRA)
          for (j=0; j<block_count; j++)
            iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                         dc_prec,intra_q,mbinfo[k].mquant);
        else
          for (j=0;j<block_count;j++)
            iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                             inter_q,mbinfo[k].mquant);
      }
      itransform(predframe,newref,mbinfo,blocks);

      sprintf(tempFileName,"newrefL%d%s\0",testLevel,name);
      store_ppm_tga(tempFileName,newref,0,horizontal_size,vertical_size,0);

      snrVals = calcSNR(neworg,newref);

      if (testLevel == 1)
      {
        level1YSnr = snrVals.Ymse;
      }
      else if ((testLevel == 2) || (testLevel == 3)) // then we have: small->encoded
      {
        if (testLevel == 2)
          level2YSnr = snrVals.Ymse;
```

```
        else if (testLevel == 3)
          level3YSnr = snrVals.Ymse;

        //need to store it to ppm
        sprintf(smallEncStoredFName,"smEncSto%s\0",name);
        sprintf(smallEncStoredFNameBase,"smEncSto%s\0",baseScaledFrameName);
        store_ppm_tga(smallEncStoredFName,newref,0,horizontal_size,vertical_size,0);
//this is small->encoded->stored
      }

      sprintf(message,
"DDATA,Level,%d,display#,%d,frame,%d,Type,%c,Dim,%dx%d,Area,%d,Bit
Rate,%lf,S,%d,TargetBits,%d,GOPOverflow,%d,Q,%.1f,YSnr,%3.3g,Level1Snr,%3.3g,Level1S,%d,L
evel1Q,%.1f,Scale,%lf\n",
             testLevel, f, i, ipb[pict_type], horizontal_size, vertical_size,
horizontal_size*vertical_size, bit_rate, currLevelS, TargetBits,
             gopOverflow, currLevelQ, snrVals.Ymse, level1YSnr, level1S, level1Q,
testScale);
      outputMessage();


      if (snrVals.Ymse > bestYSnr)
      {
        bestYSnr = snrVals.Ymse;
        bestYSnrLevel = testLevel;
      }

      stats();

      if (testLevel == MaxTestLevels + 1) // we have just ran it for real
      {
        testLevel = 0; // so falsify loop condition
      }
      else // see if can increment to the next level
      {
          testLevel++;
          refSnrPass = 0;
      }
    }// end of while (testLevel > 0)
    // at this point, any imageScaleTesting should be over (i.e., the flag no be turned
on after this point)

    sprintf(name,tplref,f+frame0);
    writeframe(name,newref);
  }
  putseqend();
}
```

putbits.c

```
/* putbits.c, bit-level output                                              */

/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */

/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

#include <stdio.h>
#include "config.h"
#include "global.h"  // added for accessing imageScaleTesting

#define BUFLENGTH         2048
extern int sockfd,newSockfd;
extern FILE *outfile; /* the only global var we need here */

/* private data */
static unsigned char outbfr;
static int outcnt;
static int bytecnt;
static unsigned char buf[BUFLENGTH];
static int bufCnt = 0, bufcounter = 0;
FILE *fp;
FILE *tfp;


/* initialize buffer, call once before first putbits or alignbits */
void initbits()
{
  outcnt = 8;
  bytecnt = 0;
}

/* write rightmost n (0<=n<=32) bits of val to outfile */
void putbits(val,n)
int val;
int n;
{
  int i;
  unsigned int mask;
  int index = 0, fill = 0;

  imageScaleTestBufCnt += n;
  mask = 1 << (n-1); /* selects first (leftmost) bit */

  for (i=0; i<n; i++)
  {
```

```
    outbfr <<= 1;

    if (val & mask)
      outbfr|= 1;

    mask >>= 1; /* select next bit */
    outcnt--;

    if (outcnt==0) /* 8 bit buffer full */
    {
      /*      printf("writing to %s\n",outfile); */
      putc(outbfr,outfile);
      buf[bufCnt++] = outbfr;
      if (bufCnt == BUFLENGTH)
      {
        if (imageScaleTesting != 1)
          write(newSockfd,buf, bufCnt);
        bufCnt = 0;
      }
      outcnt = 8;
      bytecnt++;
    }
  }
  if (val == 0x1B7L)
  {
    //    fill = BUFLENGTH - bufCnt;
    //    for (index = 0; index < fill; index++)
    //    {
    //      buf[bufCnt++] = 0;
    //    }
    fill = write(newSockfd,buf, bufCnt);
    bufCnt = 0;
    imageScaleTestBufCnt = 0;
  }

}

/* zero bit stuffing to next byte boundary (5.2.3, 6.2.1) */
void alignbits()
{
  if (outcnt!=8)
    putbits(0,outcnt);
}

/* return total number of generated bits */
int bitcount()
{
  return 8*bytecnt + (8-outcnt);
}
```

Sample Encoder Parameter (PAR) File

```
MPEG-2 Test Sequence, 30 frames/sec
des%d    /* name of source files */
reconDes%d       /* name of reconstructed images ("-": don't store) */
-         /* name of intra quant matrix file     ("-": default matrix) */
inter.mat         /* name of non intra quant matrix file ("-": default matrix) */
statNetDyn700a55.out  /* name of statistics file ("-": stdout ) */
2         /* input picture file format: 0=*.Y,*.U,*.V, 1=*.yuv, 2=*.ppm */
48       /* number of frames */
0         /* number of first frame */
00:00:00:00 /* timecode of first frame */
15        /* N (# of frames in GOP) */
3         /* M (I/P frame distance) */
0         /* ISO/IEC 11172-2 stream */
0         /* 0:frame pictures, 1:field pictures */
704       /* horizontal_size -- see header file of ppm inputs*/
240       /* vertical_size   -- see header file of ppm inputs*/
2         /* aspect_ratio_information 1=square pel, 2=4:3, 3=16:9, 4=2.11:1 */
5         /* frame_rate_code 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec. */
700000.0  /* bit_rate (bits/s) total target bitrate budget for 30 frames */
0.5       /* bit_rate/bw reduction rate*/
0.5      /* scale up/down step factor */
0        /* override imagescaling results*/
0         /* staic mode = 1, dynamic mode = 0 */
8000      /* Port Number */
112       /* vbv_buffer_size (in multiples of 16 kbit) */
0         /* low_delay  */
0         /* constrained_parameters_flag */
4         /* Profile ID: Simple = 5, Main = 4, SNR = 3, Spatial = 2, High = 1 */
6         /* Level ID:   Low = 10, Main = 8, High 1440 = 6, High = 4          */
0         /* progressive_sequence */
1         /* chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 */
2         /* video_format: 0=comp., 1=PAL, 2=NTSC, 3=SECAM, 4=MAC, 5=unspec. */
5         /* color_primaries */
5         /* transfer_characteristics */
4         /* matrix_coefficients */
352       /* display_horizontal_size */
120       /* display_vertical_size */
0         /* intra_dc_precision (0: 8 bit, 1: 9 bit, 2: 10 bit, 3: 11 bit */
1         /* top_field_first */
0 0 0     /* frame_pred_frame_dct (I P B) */
0 0 0     /* concealment_motion_vectors (I P B) */
1 1 1     /* q_scale_type  (I P B) */
1 1 1     /* intra_vlc_format (I P B)*/
0 0 0     /* alternate_scan (I P B) */
0         /* repeat_first_field */
0         /* progressive_frame */
0         /* P distance between complete intra slice refresh */
0         /* rate control: r (reaction parameter) */
0         /* rate control: avg_act (initial average activity) */
0         /* rate control: Xi (initial I frame global complexity measure) */
0         /* rate control: Xp (initial P frame global complexity measure) */
0         /* rate control: Xb (initial B frame global complexity measure) */
0         /* rate control: d0i (initial I frame virtual buffer fullness) */
0         /* rate control: d0p (initial P frame virtual buffer fullness) */
0         /* rate control: d0b (initial B frame virtual buffer fullness) */
2 2 11 11 /* P:  forw_hor_f_code forw_vert_f_code search_width/height */
1 1 3  3  /* B1: forw_hor_f_code forw_vert_f_code search_width/height */
1 1 7  7  /* B1: back_hor_f_code back_vert_f_code search_width/height */
1 1 7  7  /* B2: forw_hor_f_code forw_vert_f_code search_width/height */
1 1 3  3  /* B2: back_hor_f_code back_vert_f_code search_width/height */
```

APPENDIX B
PERTINENT MPEG-2 DECODER SOURCE CODE CHANGES

(Note: Modified code from MPEG Software Simulation Group [21] is in **boldface**.)

mpeg2dec.c

```
/* mpeg2dec.c, main(), initialization, option processing                    */

/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */

/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/times.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define GLOBAL
#include "config.h"
#include "global.h"
#define MAXCLIENTDATASIZE 80

/* private prototypes */
static int  video_sequence _ANSI_ARGS_((int *framenum));
static int Decode_Bitstream _ANSI_ARGS_((void));
static int  Headers _ANSI_ARGS_((void));
static void Initialize_Sequence _ANSI_ARGS_((void));
static void Initialize_Decoder _ANSI_ARGS_((void));
static void Deinitialize_Sequence _ANSI_ARGS_((void));
```

```
static void Process_Options _ANSI_ARGS_((int argc, char *argv[]));


#if OLD
static int  Get_Val _ANSI_ARGS_((char *argv[]));
#endif

/* #define DEBUG */

static void Clear_Options();
#ifdef DEBUG
static void Print_Options();
#endif

int main(argc,argv)
int argc;
char *argv[];
{
  int ret, code;

  Clear_Options();

  /* decode command line arguments */
  Process_Options(argc,argv);
  Initialize_My_Buffer();
#ifdef DEBUG
  Print_Options();
#endif

  ld = &base; /* select base layer context */

  /* open MPEG base layer bitstream file(s) */
  /* NOTE: this is either a base layer stream or a spatial enhancement stream */
  /* if ((base.Infile=open(Main_Bitstream_Filename,O_RDONLY|O_BINARY))<0)
  {
    fprintf(stderr,"Base layer input file %s not found\n", Main_Bitstream_Filename);
    exit(1);
    }*/


  if(sockfd != 0)
  {
    printf("Buffer Initialized.\n");
    Initialize_Buffer();

    if(Show_Bits(8)==0x47)
    {
      sprintf(Error_Text,"Decoder currently does not parse transport streams\n");
      Error(Error_Text);
    }

    next_start_code();
    code = Show_Bits(32);

    printf ("code = %d\n",code);
    switch(code)
    {
    case SEQUENCE_HEADER_CODE:
      break;
    case PACK_START_CODE:
      System_Stream_Flag = 1;
    case VIDEO_ELEMENTARY_STREAM:
      System_Stream_Flag = 1;
      break;
    default:
      sprintf(Error_Text,"Unable to recognize stream type\n");
      Error(Error_Text);
      break;
    }

    /*    lseek(base.Infile, 0l, 0);  */
```

```
    myLseek();
    Initialize_Buffer();
  }

  /*  if(base.Infile!=0)           */
  /*  {                            */
  /*    lseek(base.Infile, 0l, 0); */
  /*  }                            */
  myLseek();

  Initialize_Buffer();

  if(Two_Streams)
  {
    ld = &enhan; /* select enhancement layer context */

    /*    if ((enhan.Infile =
open(Enhancement_Layer_Bitstream_Filename,O_RDONLY|O_BINARY))<0)
    {
      sprintf(Error_Text,"enhancment layer bitstream file %s not found\n",
        Enhancement_Layer_Bitstream_Filename);

      Error(Error_Text);
      }*/

    Initialize_Buffer();
    ld = &base;
  }

  Initialize_Decoder();

  ret = Decode_Bitstream();

  close(sockfd);

  /*  if (Two_Streams)
      close(enhan.Infile);*/

  return 0;
}

/* IMPLEMENTAION specific rouintes */
static void Initialize_Decoder()
{
  int i;

  /* Clip table */
  if (!(Clip=(unsigned char *)malloc(1024)))
    Error("Clip[] malloc failed\n");

  Clip += 384;

  for (i=-384; i<640; i++)
    Clip[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

  /* IDCT */
  if (Reference_IDCT_Flag)
    Initialize_Reference_IDCT();
  else
    Initialize_Fast_IDCT();

}

/* mostly IMPLEMENTAION specific rouintes */
static void Initialize_Sequence()
{
  int cc, size;
  static int Table_6_20[3] = {6,8,12};

  /* check scalability mode of enhancement layer */
  if (Two_Streams && (enhan.scalable_mode!=SC_SNR) && (base.scalable_mode!=SC_DP))
```

```
    Error("unsupported scalability mode\n");

  /* force MPEG-1 parameters for proper decoder behavior */
  /* see ISO/IEC 13818-2 section D.9.14 */
  if (!base.MPEG2_Flag)
  {
    progressive_sequence = 1;
    progressive_frame = 1;
    picture_structure = FRAME_PICTURE;
    frame_pred_frame_dct = 1;
    chroma_format = CHROMA420;
    matrix_coefficients = 5;
  }

  /* round to nearest multiple of coded macroblocks */
  /* ISO/IEC 13818-2 section 6.3.3 sequence_header() */
  mb_width = (horizontal_size+15)/16;
  mb_height = (base.MPEG2_Flag && !progressive_sequence) ? 2*((vertical_size+31)/32)
                                     : (vertical_size+15)/16;

  Coded_Picture_Width = 16*mb_width;
  Coded_Picture_Height = 16*mb_height;

  /* ISO/IEC 13818-2 sections 6.1.1.8, 6.1.1.9, and 6.1.1.10 */
  Chroma_Width = (chroma_format==CHROMA444) ? Coded_Picture_Width
                                          : Coded_Picture_Width>>1;
  Chroma_Height = (chroma_format!=CHROMA420) ? Coded_Picture_Height
                                           : Coded_Picture_Height>>1;

  /* derived based on Table 6-20 in ISO/IEC 13818-2 section 6.3.17 */
  block_count = Table_6_20[chroma_format-1];

  for (cc=0; cc<3; cc++)
  {
    if (cc==0)
      size = Coded_Picture_Width*Coded_Picture_Height;
    else
      size = Chroma_Width*Chroma_Height;

    if (!(backward_reference_frame[cc] = (unsigned char *)malloc(size)))
      Error("backward_reference_frame[] malloc failed\n");

    if (!(forward_reference_frame[cc] = (unsigned char *)malloc(size)))
      Error("forward_reference_frame[] malloc failed\n");

    if (!(auxframe[cc] = (unsigned char *)malloc(size)))
      Error("auxframe[] malloc failed\n");

    if(Ersatz_Flag)
      if (!(substitute_frame[cc] = (unsigned char *)malloc(size)))
        Error("substitute_frame[] malloc failed\n");


    if (base.scalable_mode==SC_SPAT)
    {
      /* this assumes lower layer is 4:2:0 */
      if (!(llframe0[cc] = (unsigned char
*)malloc((lower_layer_prediction_horizontal_size*lower_layer_prediction_vertical_size)/(c
c?4:1))))
        Error("llframe0 malloc failed\n");
      if (!(llframe1[cc] = (unsigned char
*)malloc((lower_layer_prediction_horizontal_size*lower_layer_prediction_vertical_size)/(c
c?4:1))))
        Error("llframe1 malloc failed\n");
    }
  }

  /* SCALABILITY: Spatial */
  if (base.scalable_mode==SC_SPAT)
  {
```

```
    if (!(lltmp = (short
*)malloc(lower_layer_prediction_horizontal_size*((lower_layer_prediction_vertical_size*ve
rtical_subsampling_factor_n)/vertical_subsampling_factor_m)*sizeof(short))))
      Error("lltmp malloc failed\n");
  }

#ifdef DISPLAY
  if (Output_Type==T_X11)
  {
    Initialize_Display_Process("");
    Initialize_Dither_Matrix();
  }
#endif /* DISPLAY */

}

void Error(text)
char *text;
{
  fprintf(stderr,text);
  exit(1);
}

/* Trace_Flag output */
void Print_Bits(code,bits,len)
int code,bits,len;
{
  int i;
  for (i=0; i<len; i++)
    printf("%d",(code>>(bits-1-i))&1);
}



/* option processing */
static void Process_Options(argc,argv)
int argc;                  /* argument count  */
char *argv[];              /* argument vector */
{
  int i, LastArg, NextArg;
  struct sockaddr_in serverAddr;
  struct hostent     *hp;
  int                nbytes, endoflist, counter;
  char               *asciiServerAddr;
  char               clientBuffer[MAXCLIENTDATASIZE];
  char               *charptr, userChoice;

  /* at least one argument should be present */
  if (argc != 3)
  {
    printf("\n%s, %s\n",Version,Author);
    printf("Usage:  mpeg2decode <servername> <portnumber>\n\n");
    printf("   or:  mpeg2decode standalone out.m2v\n\n");
    exit(0);
  }

  sprintf(sentFile,"sentFile.m2v");
  sentFilePtr = fopen(sentFile, "w");
  Output_Type = 4;
  Output_Picture_Filename = "";


  if (strcmp(argv[1],"standalone") == 0)
  {
    printf ("Got switch \n");
    InputSrc = -1;
    // open m2v file for reading
    if(!(sockfd=open(argv[2],O_RDONLY|O_BINARY))<0)
    {
      printf("ERROR: unable to open reference filename (%s)\n",argv[2]);
      exit(1);
```

```
    }
    printf ("Opened %s \n",argv[2]);
  }
  else
  {

    InputSrc = 0;
    if ((hp=gethostbyname(argv[1])) == NULL)
    {
        perror("Error getting host IP.");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    serverAddr.sin_family = AF_INET;    // host byte order

    printf("Attempting to connect to server....\n");
    memcpy((char *) &serverAddr.sin_addr, (char *) hp->h_addr, hp->h_length);
    asciiServerAddr = (char *)  inet_ntoa(serverAddr.sin_addr);
    printf("server address: %s\n",asciiServerAddr);
    serverAddr.sin_port = htons(atoi(argv[2]));  // short, network byte order
    if (connect(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
    {
        perror("connect");
        exit(1);
    }
    printf("Connected to server....\n");


    printf ("Enter to request bitstream from server.\n");
    getchar();

    //1. make request for list of titles
    if ((nbytes = write (sockfd, "SEND LIST OF TITLES", 19)) < 0)
    {
      perror("write");
    }
    if (nbytes != 19)
    {
      printf("Error -- could not send request for titles, btyes written is %d\n",
nbytes);
    }
    printf("Send request for bitstream, btyes written is %d\n", nbytes);

    //printf ("Enter to begin reading bitstream from server.\n");
    // getchar();
  }// else InputSrc = socket;
  /* force display process to show frame pictures */
  if((Output_Type==4 || Output_Type==5) && Frame_Store_Flag)
    Display_Progressive_Flag = 1;
  else
    Display_Progressive_Flag = 0;

#ifdef VERIFY
  /* parse the bitstream, do not actually decode it completely */


#if 0
  if(Output_Type==-1)
  {
    Decode_Layer = Verify_Flag;
    printf("FYI: Decoding bitstream elements up to: %s\n",
      Layer_Table[Decode_Layer]);
  }
  else
#endif
    Decode_Layer = ALL_LAYERS;
```

```
#endif /* VERIFY */

  /* no output type specified */
  if(Output_Type==-1)
  {
    Output_Type = 9;
    Output_Picture_Filename = "";
  }


#ifdef DISPLAY
  if (Output_Type==T_X11)
  {
    if(Frame_Store_Flag)
      Display_Progressive_Flag = 1;
    else
      Display_Progressive_Flag = 0;

    Frame_Store_Flag = 1; /* to avoid calling dither() twice */
  }
#endif


}


#ifdef OLD
/*
   this is an old routine used to convert command line arguments
   into integers
*/
static int Get_Val(argv)
char *argv[];
{
  int val;

  if (sscanf(argv[1]+2,"%d",&val)!=1)
    return 0;

  while (isdigit(argv[1][2]))
    argv[1]++;

  return val;
}
#endif



static int Headers()
{
  int ret;

  ld = &base;


  /* return when end of sequence (0) or picture
     header has been parsed (1) */

  ret = Get_Hdr();


  if (Two_Streams)
  {
    ld = &enhan;
    if (Get_Hdr()!=ret && !Quiet_Flag)
      fprintf(stderr,"streams out of sync\n");
    ld = &base;
  }

  return ret;
```

```
}


static int Decode_Bitstream()
{
  int ret;
  int Bitstream_Framenum;

  Bitstream_Framenum = 0;

  for(;;)
  {

#ifdef VERIFY
    Clear_Verify_Headers();
#endif /* VERIFY */

    ret = Headers();

    if(ret==1)
    {
      ret = video_sequence(&Bitstream_Framenum);
    }
    else
      return(ret);
  }

}


static void Deinitialize_Sequence()
{
  int i;

  /* clear flags */
  base.MPEG2_Flag=0;

  for(i=0;i<3;i++)
  {
    free(backward_reference_frame[i]);
    free(forward_reference_frame[i]);
    free(auxframe[i]);

    if (base.scalable_mode==SC_SPAT)
    {
     free(llframe0[i]);
     free(llframe1[i]);
    }
  }

  if (base.scalable_mode==SC_SPAT)
    free(lltmp);

#ifdef DISPLAY
  if (Output_Type==T_X11)
    Terminate_Display_Process();
#endif
}


static int video_sequence(Bitstream_Framenumber)
int *Bitstream_Framenumber;
{
  int Bitstream_Framenum;
  int Sequence_Framenum;
  int Return_Value;

  Bitstream_Framenum = *Bitstream_Framenumber;
  Sequence_Framenum=0;
```

```
  Initialize_Sequence();

  /* decode picture whose header has already been parsed in
     Decode_Bitstream() */


  Decode_Picture(Bitstream_Framenum, Sequence_Framenum);

  /* update picture numbers */
  if (!Second_Field)
  {
    Bitstream_Framenum++;
    Sequence_Framenum++;
  }

  /* loop through the rest of the pictures in the sequence */
  while ((Return_Value=Headers()))
  {
    Decode_Picture(Bitstream_Framenum, Sequence_Framenum);

    if (!Second_Field)
    {
      Bitstream_Framenum++;
      Sequence_Framenum++;
    }
  }

  /* put last frame */
  if (Sequence_Framenum!=0)
  {
    Output_Last_Frame_of_Sequence(Bitstream_Framenum);
  }

  Deinitialize_Sequence();

#ifdef VERIFY
    Clear_Verify_Headers();
#endif /* VERIFY */

  *Bitstream_Framenumber = Bitstream_Framenum;
  return(Return_Value);
}



static void Clear_Options()
{
  Verbose_Flag = 0;
  Output_Type = 0;
  Output_Picture_Filename = " ";
  hiQdither  = 0;
  Output_Type = 0;
  Frame_Store_Flag = 0;
  Spatial_Flag = 0;
  Lower_Layer_Picture_Filename = " ";
  Reference_IDCT_Flag = 0;
  Trace_Flag = 0;
  Quiet_Flag = 0;
  Ersatz_Flag = 0;
  Substitute_Picture_Filename  = " ";
  Two_Streams = 0;
  Enhancement_Layer_Bitstream_Filename = " ";
  Big_Picture_Flag = 0;
  Main_Bitstream_Flag = 0;
  Main_Bitstream_Filename = " ";
  Verify_Flag = 0;
  Stats_Flag  = 0;
  User_Data_Flag = 0;
}
```

```
#ifdef DEBUG
static void Print_Options()
{

  printf("Verbose_Flag                        = %d\n", Verbose_Flag);
  printf("Output_Type                         = %d\n", Output_Type);
  printf("Output_Picture_Filename             = %s\n", Output_Picture_Filename);
  printf("hiQdither                           = %d\n", hiQdither);
  printf("Output_Type                         = %d\n", Output_Type);
  printf("Frame_Store_Flag                    = %d\n", Frame_Store_Flag);
  printf("Spatial_Flag                        = %d\n", Spatial_Flag);
  printf("Lower_Layer_Picture_Filename        = %s\n", Lower_Layer_Picture_Filename);
  printf("Reference_IDCT_Flag                 = %d\n", Reference_IDCT_Flag);
  printf("Trace_Flag                          = %d\n", Trace_Flag);
  printf("Quiet_Flag                          = %d\n", Quiet_Flag);
  printf("Ersatz_Flag                         = %d\n", Ersatz_Flag);
  printf("Substitute_Picture_Filename         = %s\n", Substitute_Picture_Filename);
  printf("Two_Streams                         = %d\n", Two_Streams);
  printf("Enhancement_Layer_Bitstream_Filename = %s\n",
Enhancement_Layer_Bitstream_Filename);
  printf("Big_Picture_Flag                    = %d\n", Big_Picture_Flag);
  printf("Main_Bitstream_Flag                 = %d\n", Main_Bitstream_Flag);
  printf("Main_Bitstream_Filename             = %s\n", Main_Bitstream_Filename);
  printf("Verify_Flag                         = %d\n", Verify_Flag);
  printf("Stats_Flag                          = %d\n", Stats_Flag);
  printf("User_Data_Flag                      = %d\n", User_Data_Flag);

}
#endif
```

getbits.c

```
/* getbits.c, bit level routines                                            */

/*
 * All modifications (mpeg2decode -> mpeg2play) are
 * Copyright (C) 1996, Stefan Eckart. All Rights Reserved.
 */

/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */

/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

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

#include "config.h"
#include "global.h"

/* initialize buffer, call once before first getbits or showbits */
char myBuf[1][DECODE_WINDOW_SIZE];
int bufc;
static int bufcounter = 0;
int blocking_readSocket(int s, char *bptr, int buflen)
{
  int n = 0, actualRead = 0;
  char *myptr = bptr;
  printf("actualRead = %d\n", actualRead);
  while (actualRead != buflen)
  {
    n = read(s, myptr, buflen - actualRead);
    printf("read %d\n", n);
    if (n <= 0)
    {
      fclose(sentFilePtr);
      break;
    }
    fwrite (myptr, 1, n, sentFilePtr);
    myptr += n;
    actualRead += n;
  }
  printf("actualRead = %d\n", actualRead);

  return actualRead;
}
void Initialize_Buffer()
{
```

```
  ld->Incnt = 0;
  //  ld->Rdptr = ld->Rdbfr + 2048;
  ld->Rdptr = ld->Rdbfr + DECODE_WINDOW_SIZE;
  ld->Rdmax = ld->Rdptr;

#ifdef VERIFY
  /*  only the verifier uses this particular bit counter
   *  Bitcnt keeps track of the current parser position with respect
   *  to the video elementary stream being decoded, regardless
   *  of whether or not it is wrapped within a systems layer stream
   */
  ld->Bitcnt = 0;
#endif

  ld->Bfr = 0;
  Flush_Buffer(0); /* fills valid data into bfr */
}

void Initialize_My_Buffer()
{
  int i;
  bufc = 0;
  for(i=0;i<1;i++)
  {
    //    read(sockfd,myBuf[i],2048);
    blocking_readSocket(sockfd,myBuf[i],DECODE_WINDOW_SIZE);
    printf("%d\n",bufcounter++);
  }
}

void myLseek()
{
  bufc = 0;
}

void Fill_Buffer()
{
  int Buffer_Level;
  if (bufc < 1)
    {
      //memcpy(ld->Rdbfr,myBuf[bufc],2048);
      memcpy(ld->Rdbfr,myBuf[bufc],DECODE_WINDOW_SIZE);
      bufc++;
      //Buffer_Level = 2048;
      Buffer_Level = DECODE_WINDOW_SIZE;
    }
  else
  {

    //    Buffer_Level = read(sockfd,ld->Rdbfr,2048);
    Buffer_Level = blocking_readSocket(sockfd,ld->Rdbfr,DECODE_WINDOW_SIZE);
    printf("%d\n",bufcounter++);
  }


  //  Buffer_Level = read(ld->Infile,ld->Rdbfr,2048);
  ld->Rdptr = ld->Rdbfr;

  if (System_Stream_Flag)
    //    ld->Rdmax -= 2048;
    ld->Rdmax -= DECODE_WINDOW_SIZE;


  /* end of the bitstream file */
  //  if (Buffer_Level < 2048)
  if (Buffer_Level < DECODE_WINDOW_SIZE)
  {
    /* just to be safe */
    if (Buffer_Level < 0)
      Buffer_Level = 0;
```

```
    /* pad until the next to the next 32-bit word boundary */
    while (Buffer_Level & 3)
      ld->Rdbfr[Buffer_Level++] = 0;

       /* pad the buffer with sequence end codes */
    //    while (Buffer_Level < 2048)
    while (Buffer_Level < DECODE_WINDOW_SIZE)
    {
      ld->Rdbfr[Buffer_Level++] = SEQUENCE_END_CODE>>24;
      ld->Rdbfr[Buffer_Level++] = SEQUENCE_END_CODE>>16;
      ld->Rdbfr[Buffer_Level++] = SEQUENCE_END_CODE>>8;
      ld->Rdbfr[Buffer_Level++] = SEQUENCE_END_CODE&0xff;
      close(sockfd); // nework (why close fd?)
    }
  }
}


/* MPEG-1 system layer demultiplexer */

int Get_Byte()
{
  //  while(ld->Rdptr >= ld->Rdbfr+2048)
  while(ld->Rdptr >= ld->Rdbfr+DECODE_WINDOW_SIZE)
  {
    //read(ld->Infile,ld->Rdbfr,2048);
    //    read(sockfd,ld->Rdbfr,2048);
    blocking_readSocket(sockfd,ld->Rdbfr,DECODE_WINDOW_SIZE);
    printf("%d\n",bufcounter++);
    //    putchar('.');

    //    ld->Rdptr -= 2048;
    ld->Rdptr -= DECODE_WINDOW_SIZE;
    //    ld->Rdmax -= 2048;
    ld->Rdmax -= DECODE_WINDOW_SIZE;
  }
  return *ld->Rdptr++;
}

/* extract a 16-bit word from the bitstream buffer */
int Get_Word()
{
  int Val;

  Val = Get_Byte();
  return (Val<<8) | Get_Byte();
}


/* return next n bits (right adjusted) without advancing */

unsigned int Show_Bits(N)
int N;
{
  return ld->Bfr >> (32-N);
}


/* return next bit (could be made faster than Get_Bits(1)) */

unsigned int Get_Bits1()
{
  return Get_Bits(1);
}


/* advance by n bits */

void Flush_Buffer(N)
int N;
{
```

```
  int Incnt;

  ld->Bfr <<= N;

  Incnt = ld->Incnt -= N;

  if (Incnt <= 24)
  {
    if (System_Stream_Flag && (ld->Rdptr >= ld->Rdmax-4))
    {
      do
      {
        if (ld->Rdptr >= ld->Rdmax)
          Next_Packet();
        ld->Bfr |= Get_Byte() << (24 - Incnt);
        Incnt += 8;
      }
      while (Incnt <= 24);
    }
    else if (ld->Rdptr < ld->Rdbfr+2044)
    {
      do
      {
        ld->Bfr |= *ld->Rdptr++ << (24 - Incnt);
        Incnt += 8;
      }
      while (Incnt <= 24);
    }
    else
    {
      do
      {
       //        if (ld->Rdptr >= ld->Rdbfr+2048)
        if (ld->Rdptr >= ld->Rdbfr+DECODE_WINDOW_SIZE)
          Fill_Buffer();
        ld->Bfr |= *ld->Rdptr++ << (24 - Incnt);
        Incnt += 8;
      }
      while (Incnt <= 24);
    }
    ld->Incnt = Incnt;
  }

#ifdef VERIFY
  ld->Bitcnt += N;
#endif /* VERIFY */

}


/* return next n bits (right adjusted) */

unsigned int Get_Bits(N)
int N;
{
  unsigned int Val;

  Val = Show_Bits(N);
  Flush_Buffer(N);

  return Val;
}
```

APPENDIX C
MATLAB CODE FOR OPTIMAL RATE-RESIZING FACTOR APPROXIMATION

```
%Author: Ju Wang, June 2003
function d3=hang_d(target_bitrate,W,H,sigma_sqr);
%hang_d(700000,704,480,10)

f=(0.001:0.001:1);
epsilong_sqr=1.2 %dependency on X, 1.2 for Laplasian
alpha=1.36 %log_2^e
b=target_bitrate./(4*f*W*H)
d_f=epsilong_sqr^2*sigma_sqr^2*exp(-alpha.*b)
%figure
%hold
%plot(f,d_f)
d2=(1-f)*sigma_sqr;
d3=d_f+d2;
plot(f,d3)
```

APPENDIX D
CASE 2 TEST PICTURES

Figure 22. Reference Picture for 3rd I-Frame (PSNR = 49.0, S=1,425,311)

Figure 23. 3rd I-Frame Using Original Encoded Picture (PSNR = 20.3, S=91,627)

Figure 24. 3rd I-Frame Using Adaptive Image Scaled Picture (PSNR=19.9, S=36,081)

LIST OF REFERENCES

[1] Mark Claypool and Jonathan Tanner, "The Effects of Jitter on the Perceptual Quality of Video," *Proceedings of the 7th ACM International Conference on Multimedia '99*, vol. 2, Oct. 30 – Nov. 5, 1999, pp.115-118.

[2] W. Ding and B. Liu, "Rate Control of MPEG Video Coding and Recording by Rate-Quantization Modeling," *IEEE Trans. on Circuits and Systems for Video Technology*, vol.6, no.1, Feb. 1996, pp.12-20.

[3] A Durand, "Deploying IPv6," *IEEE Internet Computing*, vol.5, no.1, Feb. 2001, pp.79-81.

[4] Armando Fox, Steven D. Gribble, Eric A. Brewer, Elan Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *In Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, Cambridge, MA, Oct.1996, pp. 160-170.

[5] C.A. Gonzales and E. Viscito, "Motion Video Adaptive Quantization in the Transform Domain," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.1, no.4, Dec. 1991, pp.351-361.

[6] Hsueh-Ming Hang and Jiann-Jone Chen, "Source Model for Transform Video Coder and Its Application—Part I: Fundamental Theory," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.7, no.2, April 1997, pp.287-298.

[7] Hsueh-Ming Hang and Jiann-Jone Chen, "Source Model for Transform Video Coder and Its Application—Part II: Variable Frame Rate Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.7, no.2, April 1997, pp.299-311.

[8] Nashnashi Kamat, J. Wang and J.C Liu, "An Efficient Re-routing Scheme for Voice over IP," to appear at ICME2003, Baltimore, 2003.

[9] Javed I. Khan, Qiong Gu and Raid Zaghal, "Symbiotic Video Streaming by Transport Feedback Based Quality-Rate Selection," Proceedings of the 12th IEEE International Packet Video Workshop 2002, April 2002.

[10] Jonathan C.L. Liu, Jenwei Hsieh, David H.C.Du and Meng-jou Lin, "Performance of a Storage System for Supporting Different Video Types and Qualities," *IEEE Journal on Selected Areas in Communications*, vol.14, no.9, Aug. 1996, pp.1087-1097.

[11] Victor Lo, "A Beginners Guide for MPEG-2 Standard," http://www.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/mk/mpeg2beg/beginnzi.htm, accessed June 2003.

[12] J.M. McManus and K.W. Ross, "Video-on-Demand Over ATM: Constant-Rate Transmission and Transport," *IEEE Journal on Selected Areas in Communications*, vol.14, no.9, Aug. 1996, pp. 1087-1097.

[13] B.D. Noble and M. Satyanarayanan, "Experience with Adaptive Mobile Applications in Odyssey," *Mobile Networks and Applications*, vol. 4, 1999, pp. 245-254.

[14] Antonio Ortega, "Variable Bit Rate Video Coding," Compressed Video over Networks, 2001, pp. 343-382.

[15] A. Puri and R. Aravind, "Motion-Compensated Video Coding with Adaptive Perceptual Quantization," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.1, Dec. 1991, pp. 351-361.

[16] Iain E. G. Richardson, *Video Codec Design*, John Wiley & Sons Ltd, London, 2002.

[17] J. Wang and J. Liu, "Handoff Algorithms in Dynamic Spreading WCDMA System Supporting Multimedia Traffic," *IEEE Journal of Selected Areas on Communication*, to appear in 2003.

[18] Magda El Zarki, "Video Coding and Quality Issues," *CENIC – QoS Workshop*, Jan. 24, 2002.

[19] ISO/IEC 13818-2, Information Technology – Generic Coding of Moving Pictures and Audio Information: Video, 1998.

[20] MPEG Elementary Streams, http://www.mpeg.org/MPEG/video.html#video-test-bitstreams, accessed June 2003.

[21] MPEG Software Simulation Group, MPEG-2 video codec version 1.2, http://www.mpeg.org.tristan/MPEG/MSSG, accessed October 2002.

BIOGRAPHICAL SKETCH

Arun S. Abraham was born in India. He joined the University of Florida in 1990 and received his bachelor's degree in computer and information science and engineering in 1994. Since then he has worked in the software engineering industry. He came back to the University of Florida in 2001 to pursue the master's degree.

His research interests include object oriented methodologies, design patterns, and multimedia.