7-1-2011

Investigating low-bitrate, low-complexity H.264 region of interest techniques in error-prone environments

Timothy Sperr

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Investigating Low-Bitrate, Low-Complexity H.264 Region of Interest Techniques in Error-Prone Environments

by

Timothy S. Sperr

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Marcin Łukowiak
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
July, 2011

Approved By:

Dr. Marcin Łukowiak
Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Andres Kwasinski
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Michael Kurdziel
Committee Member – Harris Corporation

Duncan Harris
Committee Member – Harris Corporation
Acknowledgements

I would like to thank my advisor Dr. Marcin Łukowiak for making this endeavor possible, as well as for his support and patience throughout this thesis and its related research efforts. I would also like to thank Dr. Andres Kwasinski, for heading up several research projects that led up to this thesis. To the other members of the committee, Dr. Michael Kurdzieł and Duncan Harris, thanks also go out for their support and interest. Special thanks go out to the Harris Corporation for providing the impetus for this investigation.

This thesis project and its supporting projects, the Video Coding and Region-of-Interest Coding projects, have been partially funded through the Harris Corporation’s University Relations Program in partnership with the Rochester Institute of Technology.
Abstract

The H.264/AVC video coding standard leverages advanced compression methods to provide a significant increase in performance over previous CODECs in terms of picture quality, bitrate, and flexibility. The specification itself provides several profiles and levels that allow customization through the use of various advanced features. In addition to these features, several new video coding techniques have been developed since the standard’s inception. One such technique known as Region of Interest (RoI) coding has been in existence since before H.264’s formalization, and several means of implementing RoI coding in H.264 have been proposed.

Region of Interest coding operates under the assumption that one or more regions of a sequence have higher priority than the rest of the video. One goal of RoI coding is to provide a decrease in bitrate without significant loss of perceptual quality, and this is particularly applicable to low complexity environments, if the proper implementation is used. Furthermore, RoI coding may allow for enhanced error resilience in the selected regions if desired, making RoI suitable for both low-bitrate and error-prone scenarios.

The goal of this thesis project was to examine H.264 Region of Interest coding as it applies to such scenarios. A modified version of the H.264 JM Reference Software was created in which all non-Baseline profile features were removed. Six low-complexity RoI coding techniques, three targeting rate control and three targeting error resilience, were selected for implementation. Error and distortion modeling tools were created to enhance the quality of experimental data. Results were gathered by varying a range of coding parameters including frame size, target bitrate, and macroblock error rates. Methods were then examined based on their rate-distortion curves, ability to achieve target bitrates accurately, and per-region distortions where applicable.
Table of Contents

Acknowledgements ........................................................................................................... i
Abstract ................................................................................................................................. ii
Table of Contents .................................................................................................................... iii
List of Figures ........................................................................................................................... v
List of Tables ........................................................................................................................... vii
Glossary of Terms ..................................................................................................................... viii

Chapter 1  Introduction ........................................................................................................... 1
  1.1. Motivation for Low-Bitrate, Low-Complexity Region of Interest Coding ................. 1
  1.2. H.264 Advanced Video Coding ..................................................................................... 1
  1.3. H.264 Profiles and Levels ............................................................................................... 4
  1.4. Region of Interest Coding in H.264 .............................................................................. 5
  1.5. H.264 JM Reference Software ....................................................................................... 7
  1.6. Project Goals & Assumptions ....................................................................................... 7

Chapter 2  Theory .................................................................................................................. 9
  2.1. H.264 Rate Control Model ............................................................................................ 9
  2.2. H.264 Flexible Macroblock Ordering Techniques ...................................................... 13
  2.3. Video Distortion Modeling ........................................................................................... 15
  2.4. Selected RoI Coding Techniques .................................................................................. 17
    2.4.1 Method 1 – Maximum Bit Transfer ........................................................................... 17
    2.4.2 Method 2 – Content-Based Bit Allocation ................................................................. 18
    2.4.3 Method 3 – Multiple-Priority RoI Coding ................................................................. 20
    2.4.4 Method 4 – Error Resilience via Nonlinear Transform .......................................... 22
    2.4.5 Method 5 – RoI Coding with Flexible Macroblock Ordering ................................... 25
    2.4.6 Method 6 – Explicit Spiral Interleaving ................................................................. 27

Chapter 3  Design .................................................................................................................. 29
  3.1. Baseline Software Code Modifications ....................................................................... 29
    3.1.1 High Profile Feature Removal ............................................................................... 29
    3.1.2 Main Profile Feature Removal ............................................................................... 31
    3.1.3 Extended Profile Feature Removal .................................................................... 31
List of Figures

Figure 1.1  H.264 CODEC Block Diagram ................................................................. 2
Figure 1.2  Features of the H.264 Baseline, Extended, Main, and High Profiles [3] ........... 5
Figure 1.3  Typical Region of Interest Incorporating Subject's Face .............................. 6
Figure 2.1  Typical Rate-Quantization Curve .................................................................. 10
Figure 2.2  Rate-Quantization Curve with Varying Source Complexity .......................... 10
Figure 2.3  Layers of Rate Control in H.264 ................................................................ 12
Figure 2.4  Example Foreground/Background FMO Mapping [3] ..................................... 15
Figure 2.5  Example of Nonlinear Transform with Background Loss .............................. 23
Figure 2.6  Advanced FMO Mapping Proposed for Method 3 ........................................ 26
Figure 2.7  Explicit Spiral Interleave (ESI) FMO Mapping [13] ........................................ 27
Figure 3.1  CIF Coastguard Video with Coefficient Errors ............................................. 40
Figure 3.2  CIF Foreman Video with Motion Vector Errors ............................................. 41
Figure 3.3  CIF Foreman Video (Frame 1) with Method 4 Transform Applied .................. 46
Figure 3.4  CIF Foreman Video (Frame 1) with Method 4 Forward and Inverse Transforms Applied .......................................................... 46
Figure 4.1  Method 3 Region Mapping Example ............................................................ 53
Figure 5.1  Coefficient Error Model Accuracy Histograms .............................................. 55
Figure 5.2  Motion Vector Error Model Accuracy Histograms ......................................... 56
Figure 5.3  PSNR Rate-Distortion Curves for Control Group .......................................... 58
Figure 5.4  SSIM Rate-Distortion Curves for Control Group ........................................... 59
Figure 5.5  Control Group Per-Region PSNR ................................................................. 60
Figure 5.6  Control Group Per-Region SSIM ................................................................. 61
Figure 5.7  Control Group Target Bitrate Accuracy ........................................................ 62
Figure 5.8  Rate-Distortion Curves for Method 1 ............................................................. 64
Figure 5.9  Akiyo CIF 512kbps Frame 30 from Control Group (Left) and Method 1 (Right) .... 65
Figure 5.10 Method 1 Per-Region Distortion for CIF "Akiyo" Video ........................................ 66
Figure 5.11 Method 1 Per-Region Distortion for CIF "Foreman" Video ............................... 67
Figure 5.12 Average P-Frame QP Comparison, Method 1 vs. Control ............................. 68
Figure 5.13 Method 1 Target Bitrate Accuracy ............................................................... 69
Figure 5.14 Rate-Distortion Curves for Method 2 ........................................................... 70
Figure 5.15 Method 2 Per-Region Distortion for CIF "Akiyo" Video ........................................ 71
Figure 5.16 Method 2 Per-Region Distortion for CIF "Foreman" Video ............................... 72
Figure 5.17 Method 2 Target Bitrate Accuracy ............................................................... 73
Figure 5.18 Rate-Distortion Curves for Method 3 ........................................................... 74
Figure 5.19 Method 3 Per-Region Distortion for "Akiyo" CIF Video ........................................ 75
Figure 5.20 Method 3 Target Bitrate Accuracy ............................................................... 76
Figure 5.21 Baseline Coefficient Error Response, "Akiyo" CIF ........................................ 78
Figure 5.22 Baseline Coefficient Error Response, "Foreman" CIF ...................................... 79
Figure 5.23 Baseline Motion Error Response, "Akiyo" CIF ............................................... 80
Figure 5.24 Baseline Motion Error Response, "Foreman" CIF .......................................... 81
Figure 5.25 Per-Frame Analysis of Error Insertion .......................................................... 83
Figure 5.26  Rate-Distortion Comparison, Method 4 vs. Control Group .................................................. 85
Figure 5.27  Method 4 Blocking and Background Artifacts (CIF Akiyo, Frame 30) ................................. 86
Figure 5.28  Method 4 Coefficient Error Response, “Akiyo” CIF ................................................................. 87
Figure 5.29  Method 4 Coefficient Error Response, “Foreman” CIF ......................................................... 88
Figure 5.30  Method 4 Coefficient Error Resilience, CIF "Akiyo," Frame 30, 512 kbps .............................. 89
Figure 5.31  Method 4 Coefficient Error Resilience, CIF "Foreman," Frame 60, 512 kbps ....................... 90
Figure 5.32  Method 4 Motion Error Response, "Akiyo" CIF ................................................................. 91
Figure 5.33  Method 4 Motion Error Response, "Foreman" CIF ................................................................. 92
Figure 5.34  Method 4 Motion Error Resilience, CIF “Foreman,” Frame 30, 512 kbps ......................... 93
Figure 5.35  Method 5 Coefficient Error Response, "Akiyo” QCIF ............................................................ 94
Figure 5.36  Method 5 Coefficient Error Response, "Foreman” CIF ......................................................... 95
Figure 5.37  Method 5 Motion Error Response, "Akiyo” QCIF ................................................................. 96
Figure 5.38  Method 5 Motion Error Response, "Foreman” CIF ................................................................. 97
Figure 5.39  Method 6 Coefficient Error Response, "Akiyo” QCIF ............................................................ 98
Figure 5.40  Method 6 Coefficient Error Response, "Foreman” QCIF ....................................................... 99
Figure 5.41  Method 6 Motion Error Response, "Akiyo” QCIF ................................................................. 100
Figure 5.42  Method 6 Motion Error Response, "Foreman” QCIF ............................................................ 101
## List of Tables

<table>
<thead>
<tr>
<th>Table 2-1</th>
<th>H.264 Macroblock-to-Slice-Group Mappings</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3-1</td>
<td>Supported H.264 Levels</td>
<td>33</td>
</tr>
<tr>
<td>Table 3-2</td>
<td>Baseline Modification Results</td>
<td>35</td>
</tr>
<tr>
<td>Table 3-3</td>
<td>Error Insertion Pseudocode</td>
<td>38</td>
</tr>
<tr>
<td>Table 4-1</td>
<td>Control Group Encoder Parameters</td>
<td>51</td>
</tr>
</tbody>
</table>
## Glossary of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>An H.264 profile that includes low-complexity coding features as well as error-resilience tools</td>
</tr>
<tr>
<td>B-Frame</td>
<td>A video frame that is decoded using Inter prediction from one or more reference frames</td>
</tr>
<tr>
<td>BU</td>
<td>Basic Unit, the smallest piece of video on which H.264 rate control may operate</td>
</tr>
<tr>
<td>CABAC</td>
<td>Context-Adaptive Binary Arithmetic Coding, a high-complexity H.264 entropy coding technique</td>
</tr>
<tr>
<td>CAVLC</td>
<td>Context-Adaptive Variable Length Coding, an H.264 entropy coding technique</td>
</tr>
<tr>
<td>CIF</td>
<td>Common Intermediate Format, a video resolution of 352x288 pixels</td>
</tr>
<tr>
<td>CODEC</td>
<td>Encoder/Decoder pair</td>
</tr>
<tr>
<td>Coefficient</td>
<td>An element of the output matrix of the Discrete Cosine Transform</td>
</tr>
<tr>
<td>CPB</td>
<td>Coded Picture Block, a syntax element describing commonly occurring patterns in transform coefficients</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform, a matrix transform common to modern video CODECs</td>
</tr>
<tr>
<td>DPB</td>
<td>Decoded Picture Buffer, a storage location for reconstructed frames</td>
</tr>
<tr>
<td>DPCM</td>
<td>Differential Pulse-Code Modulation, a form of signal encoder that uses traditional Pulse-Code Modulation techniques coupled with prediction</td>
</tr>
<tr>
<td>Extended</td>
<td>An H.264 profile that is an extension of the Baseline Profile</td>
</tr>
<tr>
<td>FMO</td>
<td>Flexible Macroblock Ordering, an H.264 Baseline Profile error-resilience technique</td>
</tr>
<tr>
<td>GOP</td>
<td>Group of Pictures, a set of one or more frames in display order</td>
</tr>
<tr>
<td>High</td>
<td>An H.264 profile that includes high-complexity extensions for high definition video coding; an extension of the Main Profile</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>I-Frame</strong></td>
<td>A video frame that is decoded using only Intra prediction</td>
</tr>
<tr>
<td><strong>Inter Prediction</strong></td>
<td>A prediction method using available samples from within previous reference frames</td>
</tr>
<tr>
<td><strong>Intra Prediction</strong></td>
<td>A prediction method using available samples only from within the same frame</td>
</tr>
<tr>
<td><strong>IPCM</strong></td>
<td>A lossless macroblock coding mode in which pixel data is transmitted directly</td>
</tr>
<tr>
<td><strong>JM</strong></td>
<td>Joint Model, an open-source H.264 implementation provided by the JVT</td>
</tr>
<tr>
<td><strong>JVT</strong></td>
<td>Joint Video Team, the standards group responsible for H.264, consisting of members of both MPEG and VCEG</td>
</tr>
<tr>
<td><strong>Level</strong></td>
<td>A set of performance requirements for decoding a video, used to specify encoder and decoder compliance</td>
</tr>
<tr>
<td><strong>Macroblock</strong></td>
<td>A 16x16 block of pixels within a video frame</td>
</tr>
<tr>
<td><strong>MAD</strong></td>
<td>Mean Absolute Difference, a distortion metric used at the frame and macroblock levels</td>
</tr>
<tr>
<td><strong>MSE</strong></td>
<td>Mean Squared Error, a distortion metric used at the frame and macroblock levels</td>
</tr>
<tr>
<td><strong>Main</strong></td>
<td>An H.264 profile that includes more computationally intense features than Baseline, but lacks error resilience tools</td>
</tr>
<tr>
<td><strong>MB</strong></td>
<td>Macroblock, a 16x16 pixel piece of a video frame</td>
</tr>
<tr>
<td><strong>Motion Vector</strong></td>
<td>A 2-d vector specifying the relative motion of one macroblock or macroblock partition to its reference, used in Inter prediction</td>
</tr>
<tr>
<td><strong>MPEG</strong></td>
<td>Moving Picture Experts Group, a video compression standards committee under the International Organization for Standardization</td>
</tr>
<tr>
<td><strong>NAL</strong></td>
<td>Network Abstraction Layer, a process by which the encoded H.264 bitstream is translated into byte-aligned packets</td>
</tr>
<tr>
<td><strong>P-Frame</strong></td>
<td>A video frame that is decoded using Inter prediction from a single reference frame</td>
</tr>
<tr>
<td><strong>PPS</strong></td>
<td>Picture Parameter Set, an H.264 syntax element that specifies video coding parameters for one or more frames</td>
</tr>
<tr>
<td><strong>Profile</strong></td>
<td>A specific subset of the H.264 bitstream, used to specify encoder and decoder compliance</td>
</tr>
<tr>
<td><strong>PSNR</strong></td>
<td>Peak Signal-to-Noise ratio, an objective measure of video quality</td>
</tr>
<tr>
<td><strong>QCIF</strong></td>
<td>Quarter Common Intermediate Format, a video resolution of 176x144 pixels</td>
</tr>
<tr>
<td><strong>QP</strong></td>
<td>Quantization Parameter, a value expressing the amount of quality to be removed during the quantization process; varies from 1 to 51 in Baseline Profile H.264</td>
</tr>
<tr>
<td><strong>Rate Control</strong></td>
<td>Video coding algorithms utilized to achieve a target bitrate for decoding or transmission</td>
</tr>
<tr>
<td><strong>RDO</strong></td>
<td>Rate-Distortion Optimization, a computationally intense process used to select a prediction mode by balancing bitrate and distortion</td>
</tr>
<tr>
<td><strong>Residual</strong></td>
<td>The result of subtracting a predicted macroblock from its reference</td>
</tr>
<tr>
<td><strong>RoI</strong></td>
<td>Region of Interest</td>
</tr>
<tr>
<td><strong>Slice</strong></td>
<td>One or more macroblocks; the smallest unit in which compressed pixel information can be sent using NAL coding</td>
</tr>
<tr>
<td><strong>Slice Group</strong></td>
<td>A subset of all of the macroblocks within a frame, used to partition a frame into regions</td>
</tr>
<tr>
<td><strong>SPS</strong></td>
<td>Sequence Parameter Set, an H.264 syntax element that specifies video coding parameters for an entire video sequence</td>
</tr>
<tr>
<td><strong>SSIM</strong></td>
<td>Structural Similarity Index, an objective measure of video quality</td>
</tr>
<tr>
<td><strong>YCbCr</strong></td>
<td>A color space typically used in video coding; analogous to YUV, but implies digital encoding</td>
</tr>
<tr>
<td><strong>YUV</strong></td>
<td>A color space typically used in video coding, with Y representing luminance and U, V representing blue and red chrominance</td>
</tr>
<tr>
<td><strong>VCEG</strong></td>
<td>Video Coding Experts Group, a video compression standards committee under the International Telecommunication Union</td>
</tr>
</tbody>
</table>
Chapter 1  Introduction

1.1.  Motivation for Low-Bitrate, Low-Complexity Region of Interest Coding

Since its inception in 2003, the H.264 video coding standard has become one of the most robust Discrete Cosine Transform (DCT) based video compression specifications. Several features have been added as annexes to the standard, including Scalable Video Coding and Multiview (3-D) coding extensions. There has also been much work in the field of H.264 Region of Interest coding; however, no techniques have yet emerged as part of the draft model. As the number of computing resources available to smaller devices continues to increase, so too do efforts to implement low-complexity H.264 CODECs for use in real-time mobile video coding and similar applications.

Though there has been much research in both of these areas, little has been done to combine them. As low-complexity video coding often utilizes wireless transmission channels, which support much smaller bandwidths than wired media and boast greatly varying error rates, Region of Interest coding may be applied to great effect by increasing the error resilience of important pieces of a video. RoI coding may also be used to allocate a larger part of the bandwidth to these regions. Using either technique often requires tradeoffs between computational requirements, video quality, and bandwidth utilization. This thesis project sought to examine those tradeoffs.

1.2.  H.264 Advanced Video Coding

H.264/AVC is a video compression standard developed by the Joint Video Team (JVT), a team consisting of members of both the MPEG and VCEG video compression groups. Initially approved in 2003, the standard uses the DPCM, or Differential Pulse-Code Modulation, coding model. H.264 CODECs also make use of the Discrete Cosine Transform. A block diagram of a basic H.264 CODEC is shown below, in Figure 1.1.
H.264 processes videos frame-by-frame, where each frame is then broken up into macroblocks – discrete 16x16 pixel regions. As H.264 operates in the YCbCr color space, each macroblock contains both luma and chroma pixel values. Figure 1.1 shows the various major elements of an H.264 CODEC, and the processing that each macroblock undergoes during the encoding and decoding processes. The major encoding elements are each described briefly below.

The first stage of encoding in an H.264 CODEC is known as prediction. Before a macroblock can be processed, an encoder must select its prediction model; H.264 contains two primary modes. They are Intra, in which the pixels of a macroblock are constructed by utilizing samples from neighboring blocks within the same frame, and Inter, in which the pixel values are computed using samples from previously coded frames. Each of these modes has several options, leading to a large number of possible values. The end goal of this first phase is to allow the predicted macroblocks to be as close as possible to the corresponding original blocks, using information available to the decoder. After the pixel values are calculated at the first stage, a residual block is computed as the difference between the original macroblock and the predicted block. The closer this residual is to zero, the higher compression attainable for the macroblock. For all cases, prediction is a lossless process.

All residuals are then processed using one or more transforms. H.264 does not technically use the Discrete Cosine Transform; it instead uses an integer approximation to reduce the number of floating-point operations required. This transform allows residual pixel information to be represented in the
frequency domain, where low-frequency (DC) components tend to carry higher importance than higher-frequency (AC) components; actual importance of frequency depends on the properties of the input video. The exact transform used depends on a number of factors, including whether luma or chroma values are being transformed and which prediction mode was used. As with prediction, transformation is a lossless process.

Following transformation, the coefficients are reduced in magnitude during quantization, the only lossy compression process within H.264. Quantization operates via a single parameter – QP – that determines the scaling factor for the coefficients. QP can be provided as a fixed input parameter, or may be calculated via rate control, a process that is described in Section 2.1.

After being quantized, coefficients are given an ordering for later transmission. This reordering places coefficients in order of importance, with the DC component in the top-left of the coefficient matrix first and the highest-frequency AC component in the bottom-right of the matrix last. It is commonly known for the zigzag pattern that it traverses.

The final compression process, known as entropy coding, is a more traditional form of compression. It utilizes variable-length codes to represent commonly-occurring pixel patterns and bitstrings with fewer bits. H.264 uses three different forms of entropy coding: Context-Adaptive Binary Arithmetic Coding (CABAC), Context-Adaptive Variable Length Coding (CAVLC), and Exp-Golomb coding. The choice of coding type depends on which syntax element is being processed, as well as computational requirements – CABAC is more complex, but can provide higher compression ratios, for example.

H.264 includes one additional layer of encoding, known as the Network Abstraction Layer, or NAL. The NAL processes the output of the entropy coder and combines various parts of the H.264 bitstream into byte-aligned NAL units that are more amenable to packet-switched networks. For example, all macroblocks in a single slice are transmitted as a single NAL unit.

The encoding process discussed thus far is known as the forward encoding path. In addition to processing raw video data into a compressed bitstream, an H.264 encoder must also conform to the limits of any decoders processing the bitstream. To maintain synchronization with the reconstructed data that decoders use, an encoder must also reconstruct macroblocks. All processed macroblocks must be rescaled (inverse quantization), and processed via its inverse transforms. Because quantizing macroblocks individually tends to produce blocking artifacts at macroblock boundaries, H.264 also includes an in-loop deblocking filter, which operates on each frame to smooth out these artifacts. All
reconstructed macroblocks are stored in the Decoded Picture Buffer (DPB) until they are no longer needed as references.

For more information on H.264 encoding and decoding, refer to [1] and [2].

1.3. H.264 Profiles and Levels

The H.264 standard draft specifies the bitstream format for all syntax elements contained therein, which includes a wide range of features. As not all of these options are suitable for particular applications, the standard also specifies a set of profiles. Each profile allows a certain set of features from the standard, and in effect only considers those features. The draft also specifies a set of levels, which provide a means of communicating the performance requirements of the decoding process. Each H.264 encoder supports a given set of profiles and levels. If a decoder does not support the profile used to encode a video, it will lack one or more features necessary for decoding; likewise, if it does not support the level used during encoding, the decoder will typically lack the memory capacity or computing resources necessary to process the video.
Figure 1.2 describes the four major profiles of H.264. The Main and High profiles are typically used in high-resolution and high-quality video, and are not generally suitable for real-time video coding. The Baseline and Extended profiles remove several features with higher computational requirements, and add several error resilience tools. This project focused primarily on the Baseline profile, which is particularly well-suited to low-complexity platforms.

1.4. Region of Interest Coding in H.264

In an error-prone environment, improving video quality beyond a certain point is often not possible, even when considering H.264’s breadth of compression features; this is especially true at lower bitrates. In these circumstances, there are still tradeoffs that may be leveraged to ensure that certain quality requirements are met. For example, it is possible to sacrifice quality of the video as a whole to improve
the quality of one or more specific regions. This is the goal of Region of Interest coding. Several techniques are available for RoI coding in H.264, and several of these are applicable specifically to low complexity H.264 implementations, and are ideal for computationally limited platforms.

With H.264, there are two major tradeoffs that may be exploited when considering Region of Interest coding. The first is the tradeoff between quality and bitrate – for a given video sequence, the amount of quantization may be increased to reduce the bitrate of the video at the expense of quality. Region of interest coding may instead reallocate bits from the background to the region(s) of interest, increasing quantization in the background and decreasing it in the foreground. Doing so will increase the quality of the foreground, possibly making the video appear of higher quality to a human observer. For example, human observers have been found to focus on areas of a video that contain faces and hands. If the region of interest includes these areas, as in Figure 1.3, subjective video quality may be improved. Such region of interest techniques must by necessity target the rate controller when implemented in H.264.

The other major tradeoff that may be exploited is that between bitrate and error resilience. H.264 in particular provides several tools (one of which is outlined in Section 2.2) for enhancing the error resilience of a bitstream, by reordering and partitioning data, or by introducing redundant elements. Each of these tools either directly or indirectly introduces overhead into the bitstream when in use. Region of interest techniques may instead choose to enhance the error resilience of the foreground but
not the background, which may be far more effective at preserving RoI quality when a high error rate is present.

This thesis project selected RoI coding techniques from both categories, specifically targeting those that were deemed applicable to both low-complexity and low-bitrate or error-prone environments.

1.5. H.264 JM Reference Software

In addition to drafting a standard for the H.264 bitstream, the JVT has designed and implemented a reference H.264 CODEC in the form of the JM Reference Software. It is one of the most fully-featured H.264 CODECs available, and has the added benefit of using the same terminology in its code as the H.264 standard document.

The JM Reference Software implementation includes all H.264 profiles and levels, and has been updated to include the Scalable Video Coding and Multiview Coding extensions as well. It does, however, perform much more slowly than other implementations [4].

This project utilized the JM Reference Software as the foundation for its implementation and testing. The code was modified by stripping out all non-Baseline features, and further outfitted with error modeling and region of interest coding features. These code modifications are described in Chapter 3.

1.6. Project Goals & Assumptions

As previously stated, this project’s motive was to investigate low-complexity Region of Interest coding techniques that are applicable to low-bitrate and error-prone scenarios. The term “low bitrate” is used to mean any encoded video bitstream that requires approximately 256 kbps or less to transmit at some specific framerate. “Error-prone” is used to describe a transmission channel that is likely to lose packets of data. These two definitions form the basis of several assumptions for the project, namely the RoI coding methods selected, their implementation, and the parameters chosen for the experiments performed.

Another important assumption was that the desired RoI techniques targeted implementations on low-complexity platforms; that is, platforms with relatively limited computational resources. This assumption was the primary criterion used to select each of the six coding techniques. As profiling of H.264 encoders and decoders [5] demonstrates motion estimation and compensation to be a major bottleneck of
CODEC performance, methods targeting these functions were eliminated from the investigation. Furthermore, methods utilizing non-Baseline Profile features, such as weighted prediction, bi-prediction, and subpixel interpolation, were also excluded.

The investigation sought to examine how each RoI coding technique performed in terms of video quality, ability to achieve target bitrate, and error resilience to both coefficient and motion vector errors. Section 4.2 provides detailed information about the parameters that were varied and outputs that were gathered during experimentation. Chapter 5 provides a thorough analysis of the results obtained.
Chapter 2  Theory

2.1.  H.264 Rate Control Model

Rate control in video coding is designed to address two issues – limitations on transmission speeds, and decoder buffering. If transmission channels cannot handle the volume of video data at a sufficient rate, then significant buffering will be required when decoding. Similarly, it is desirable to avoid the decoder’s buffer becoming too full or from emptying, causing the decoding process to stall waiting for data. If all of these requirements are met, then near-real-time video coding becomes possible, a common goal in low complexity scenarios. The rate control model for the JM Reference Software is specified primarily in the document JVT-G012-r1 [6]. Note that the H.264 standard places no constraints on rate control, and simply provides guidance to aid in implementation. The following model is the default rate control model included in the JM.

The fundamental goal of rate control then becomes meeting the target bitrate. Knowing this bitrate, which may be constant or time-varying, and the framerate of the video being processed, an encoder may generate bit targets for various parts of the video sequence. Rate control in H.264 operates under several assumptions, the first of which are the properties of video rate-distortion curves.

For a given encoder and video frame, there exists a rate-quantization curve that describes the relationship between compression rate and bitrate. Because compression rate in H.264 is largely determined by the quantization parameter used, QP is often used in its place. Figure 2.1 shows the shape of a typical rate-quantization curve. Note that this curve assumes all other values, such as prediction mode, constant.
Over the course of a video sequence, the complexity of frames may vary drastically. Complexity can be defined either as variation across a single frame or variation across multiple frames (i.e. fast motion). As source complexity varies, the rate-quantization curve will shift, as shown in Figure 2.2.

The task of a rate controller is to select the appropriate point on the rate-quantization curve to ensure that a target bitrate is met. This task becomes more difficult as the variance of the source complexity increases. The relationship between QP and bitrate has been studied in some detail and can be estimated through a quadratic model, as in (2-1).
In (2-1), Bits represents the number of bits for the entity being coded (typically a frame), MAD is the Mean Absolute Difference (used to measure distortion) between two entities, and $C_1$ and $C_2$ are coefficients updated for each entity. In H.264, the bit target can apply only for residual bits; a certain number of bits will always be required by header elements.

To ensure that the decoder’s buffer requirements are met, the rate controller employs a fluid flow traffic model as well. This model assumes a given number of bits will be removed from the buffer for each frame that is processed, and uses both the number of bits in the buffer and a target buffer level to determine bit budgets at each level of rate control.

A problem arises in the availability of the mean absolute difference. H.264 encoders typically employ a technique known as Rate-Distortion Optimization, or RDO, to select a prediction mode. RDO attempts to encode a macroblock using each prediction mode, and selects the mode which minimizes a set of cost functions. These cost functions describe the tradeoff between bitrate and video quality. To solve the cost functions, RDO requires that a QP be specified, and provides a value for mean absolute difference. Meanwhile, the rate controller requires a MAD to compute bit targets, and provides a QP. This scenario creates what is known as the “chicken-and-egg dilemma,” and is solved in the rate controller by predicting the MAD using a linear model.

The JM rate control model operates at several levels. A video sequence is broken up into groups of pictures (GOPs), each of which consists of several frames. The rate controller assigns a bit budget to each GOP, and updates the budget after processing every frame. The controller also assigns budgets to each frame, computed using the quadratic model.
A frame may be further divided into basic units (BUs). A BU may consist of one or more macroblocks, with the only restriction that the number of macroblocks in a BU be an integer divisor of the number of macroblocks within a frame. To the rate controller, basic units are indivisible; each BU that is processed receives a single QP value, and all macroblocks within the BU are processed with that parameter.

Use of the basic unit layer in the JM rate control model replaces the frame-layer model, with two exceptions. First, the initial frame in each GOP (which is always an I-Frame) is always coded using a single quantization parameter. The P-Frame immediately following it is also coded entirely using that same QP. Second, all B-Frames are coded using a fixed QP, calculated at the frame level. As B-Frames were not utilized for this project, they will not be discussed in detail here. All remaining P-frames are processed at the basic unit level, by calculating a frame’s bit budget normally and dividing it evenly among all BU’s in the frame.

The JM rate control model also specifies actions to be taken should the encoder exceed its bit budget at a given layer. At the frame layer, if the rate controller finds that it does not have enough bits to
complete the GOP, a number of frames following it are “skipped” until the rate controller estimates that
the budget will be met again. H.264 possesses special skip prediction modes, in which no macroblock
data needs to be transmitted. A decoder encountering a skipped macroblock will assume an
algorithmically-determined prediction mode and a residual of zero for the macroblock. This has the
effect of generating less than 1 bit per skipped macroblock; the encoder has to transmit only the
number of macroblocks that have been skipped. If a frame’s bit budget is exceeded by any of its BUs,
the rate controller will increase the QP for all remaining BUs in the frame.

The QP provided by the rate controller is subject to two further limitations. QP that varies too much
from frame to frame can cause large variance in visual quality that is noticeable to human observers. To
avoid this, the JM rate control model limits the QP of a frame to within ±2 of the previous frame’s QP.
Similarly, blocking artifacts can occur between neighboring BUs if the QP delta between them is too
large, and the rate controller places limits on this as well. The QP is also bounded between its minimum
and maximum values, which in H.264 are 1 and 51, respectively.

As rate control normally uses floating-point operations, in an embedded environment the rate controller
is best targeted for an embedded processor, where fixed- or floating-point ALUs are more likely to be
available. Use of RC does not generally add significant delay to the encoding process, and as such it is
not a good target for hardware acceleration.

2.2. H.264 Flexible Macroblock Ordering Techniques

The H.264 Baseline Profile includes several tools for adding error resilience to an encoded bitstream. Of
these, only one lends itself readily to RoI coding. Flexible Macroblock Ordering (FMO) provides a means
of partitioning video frames into regions, known as slice groups. A slice group, like a slice, is a subset of
the macroblocks within a video frame. However, slice groups may also consist of one or more slices.
When FMO is in use, each macroblock in the frame is assigned a slice group ID number, and all
macroblocks within each slice group are transmitted together. The only other requirement is that
macroblocks be transmitted in raster order within a slice.

Multiple slice groups allow the option of mapping coded macroblocks to the decoded frame in a number
of flexible ways. Allocation of macroblocks is specified in what is known as a macroblock-to-slice-group
map, and it is transmitted in each PPS if FMO is enabled. The size of the mapping (and thus the overhead
of FMO) is determined by the FMO mode that is in use. H.264 as a standard specifies six specialized macroblock-to-slice-group modes, plus a seventh explicit mode that allows any customized mapping.

Table 2-1  H.264 Macroblock-to-Slice-Group Mappings

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Max. Slice Groups</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interleaved</td>
<td>N</td>
<td>N macroblocks are assigned to each of M slice groups, in round-robin order</td>
</tr>
<tr>
<td>1</td>
<td>Dispersed</td>
<td>N</td>
<td>Macroblocks within each group are dispersed throughout the picture, in an approximation of a checkerboard pattern.</td>
</tr>
<tr>
<td>2</td>
<td>Foreground/Background</td>
<td>N</td>
<td>N rectangular overlapping/non-overlapping regions are each assigned a slice group, with the remainder (background) allocated to another group.</td>
</tr>
<tr>
<td>3</td>
<td>Box-Out</td>
<td>2</td>
<td>A box of specified size is created by starting from the center and spiraling outward. Group 0 is assigned this box; group 1 is assigned all others.</td>
</tr>
<tr>
<td>4</td>
<td>Raster Scan</td>
<td>2</td>
<td>Group 0 is assigned the first M macroblocks in raster scan order; group 1 is assigned all others.</td>
</tr>
<tr>
<td>5</td>
<td>Wipe</td>
<td>2</td>
<td>Group 0 is assigned the first M macroblocks in vertical scan order; group 1 is assigned all others.</td>
</tr>
<tr>
<td>6</td>
<td>Explicit</td>
<td>N</td>
<td>Each macroblock is explicitly assigned a slice group ID.</td>
</tr>
</tbody>
</table>

FMO can be applied as an error resilience tool by allowing slices to be transmitted out-of-order; in this manner, the loss of several slices in a row may be spread throughout the frame, becoming less noticeable to observers (the dispersed mapping type is intended for this purpose). It may also be used as a software reference, allowing macroblocks in one slice group to be processed differently than macroblocks in another, making it well-suited for region of interest applications. This is particularly true when considering the fact that no alterations to the bitstream formatting or decoder are required should RoI coding be implemented; any decoder capable of processing a bitstream with FMO included is also capable of decoding videos containing regions of interest (provided that the RoI coding technique does not explicitly require decoder modifications).
Of these seven mappings, only two are truly of interest for Region of Interest coding: the foreground/background and explicit mappings. Both allow useful RoI shapes to be constructed. The foreground/background mapping type restricts regions to rectangles, but requires fewer overhead bits to transmit the mapping. As only the locations of the top-left and bottom-right macroblocks for each foreground region are required, the total number of values that must be transmitted for map type 2 is $2(N-1)$, where $N$ is the number of slice groups. The explicit mapping type allows any arbitrary shape (limited to macroblock resolution) but requires more bits to transmit, as each macroblock requires its own slice group ID. The size of the mapping is therefore $N_{MBFrame}$, the number of macroblocks in a frame. The exact number of bits required for each mapping is unknown, as each value utilizes an Exp-Golomb variable-length code during transmission.

![Figure 2.4 Example Foreground/Background FMO Mapping [3]](image)

As H.264 bitstreams allow the transmission of multiple picture parameter sets, the transmission of several FMO mappings is possible. Each will incur the overhead penalty of a full PPS, as opposed to just the mapping. In addition, H.264 allows a maximum PPS ID value of 255, limiting the number of mappings. This effectively disallows frame-by-frame RoI tracking (which would be inefficient as well) without breaking the constraints of the standard.

2.3. Video Distortion Modeling

The study of video quality measurement has long been of interest to video coding researchers. The difficulty of designing experiments centered on human opinion of video quality, coupled with the lack of mathematical distortion models that accurately reflect human preference, is still a concern when considering results. This project has selected two common metrics for video distortion, PSNR and SSIM, with some variations specific to RoI coding.
PSNR, or Peak Signal-to-Noise Ratio, is the most commonly-used objective metric for video distortion. It is defined in terms of the maximum pixel value and the Mean Squared Error (MSE) between two video frames. Similar to other SNRs, it is expressed on a logarithmic scale. (2-2) and (2-3) provide the definition for video PSNR.

\[
PSNR = 10 \log_{10} \left( \frac{\text{pixel}_{\text{max}}^2}{\text{MSE}} \right)
\]  

(2-2)

Where

\[
\text{MSE} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (F_1(i, j) - F_2(i, j))^2
\]  

(2-3)

with \( F_1 \) and \( F_2 \) representing the two frames being compared, and \( \text{pixel}_{\text{max}}^2 \) is the maximum pixel value, squared. To extend this definition to include an entire video sequence, the PSNR value of all frames is simply averaged. Additionally, PSNR does not specify whether luma or chroma values are being considered, thus PSNR can be computed for the Y, Cr, and Cb components separately, and also as a whole. For most video sequences, Y PSNR is the best indicator of video quality, second to including both chrominance values as well. One notable problem with PSNR is that it approaches infinity as the frames approach equality, and is undefined for equal frames.

More recently Structural Similarity Index, or SSIM, has also been adopted as an additional objective metric. Though it typically only considers luma values, SSIM is generally thought to be a better indicator of human preference for video quality than PSNR. This is due to the fact that it considers variances within both pixels in both frames, as well as the covariance between frames. It also operates on a much smaller window size than a full frame, typically 8x8 pixels, and allows the windows to overlap, considering all possible windows within the frame. Equation (2-4) describes how to compute SSIM.

\[
SSIM = \frac{(2\mu_1\mu_2 + C_1)(2\sigma_{1,2} + C_2)}{(\mu_1^2 + \mu_2^2 + C_1)(\sigma_1^2 + \sigma_2^2 + C_2)}
\]  

(2-4)

Here \( \mu_1 \) and \( \mu_2 \) represent the mean pixel values for the windows in frames 1 and 2, respectively, \( \sigma_1^2 \) and \( \sigma_2^2 \) represent their variances, and \( \sigma_{1,2} \) represents the covariance of the window samples. \( C_1 \) and \( C_2 \) are
constants used to balance the ratio should the denominator become too small; their values are based on the maximum squared pixel value.

SSIM ranges from -1.0 to +1.0, with +1.0 representing two identical windows. SSIM for two frames is simply the average of the SSIM of all windows in the frames, and SSIM for a video sequence is likewise the average SSIM of all its frames.

Though very computationally intense, SSIM has been shown to correlate more strongly with human perception [7], and as such SSIM values have been included in all experiments conducted for this project, in addition to PSNR values for Y, Cr, Cb, and their combined results. However, a more accurate measure in terms of Region of Interest coding would also consider the PSNR and SSIM of each region. As no software was available to do so, and the JM Reference Software included no such model, a custom implementation was created that could compute required PSNR and SSIM values. The software is described in more detail in Section 3.3.

2.4. Selected RoI Coding Techniques

This thesis project sought to examine low-complexity RoI coding techniques that were thought to be suitable for operating in low-bitrate or error-prone environments. To this end, several different coding techniques were selected from various research sources. These methods are best split into two categories, as discussed in Section 1.4: those that focus solely on video quality (quantization and rate control), and those that focus on error resilience. The theory behind each technique will be discussed in detail below.

2.4.1 Method 1 – Maximum Bit Transfer

This method, presented in [8], provides a simplified rate control model as a means of allocating bits from background macroblocks to foreground macroblocks. As the original method was created targeting H.263, the predecessor to H.264, it has been updated here to reflect the changes made to the JM rate control model – namely, the basic unit layer.

Method 1 specifies a simple principle for bit reallocation: use the maximum amount of quantization on background MBs, and transfer the bit savings to the foreground. The equation is presented as:

\[ B_{MBT} = B_f(Q_{P_f}) + B_b(Q_{P_b}) + h \]  

(2-5)
In (2-5), $h$ represents the number of bits required for encoding header information, $B_f$ and $B_b$ represent bits required by the foreground and background, respectively, and $QP_f$ and $QP_b$ represent the corresponding quantization parameters. $B_{MBT}$ is the bit budget for a frame. MBT operates by setting $QP_b = QP_{MAX} = 51$, and solving for the bits remaining for the foreground. The authors of [8] propose an iterative method to solve for $QP_f$, in which the QP is set to $QP_{MAX}$ and steadily decreased until the difference between the new bit budget and the original has been minimized. However, this technique is computationally intense, as each iteration requires several floating-point operations. Furthermore, it does not leverage H.264’s use of basic units to apply a finer grain of rate control. As such, the following modifications were made:

For the initial I and P frames of each GOP, set the quantization parameter of all background MBs to the maximum value. Set the foreground QP to the initial QP determined by the GOP, which is typically low. Tests on several video sequences showed this to yield sufficiently close bit budgets, without significant spikes in the bitrate, and also excluding the additional iterations, each of which would require the use of the quadratic rate-quantization model.

For all following frames, require that basic unit size be exactly one macroblock. If a basic unit (macroblock) is located within the background, set its quantization value to $QP_{MAX}$. Otherwise, allow the rate controller to calculate a bit budget for each BU manually. The reduction in bits from the background macroblocks will automatically translate to lower QPs in the foreground, and the bit budget for the frame will be met more closely than at the frame level.

As this technique violates the constraints placed on QP variance, these constraints were removed for Method 1.

### 2.4.2 Method 2 – Content-Based Bit Allocation

Described in [9], Method 2 provides a multifaceted approach to bit reallocation from background to foreground. It uses information on both the relative size and motion of each region relative to the whole frame, and also allows variable priority to be given to the RoI. The technique was originally designed for use in H.261, where it operated at the frame layer, but it can be extended to work at the frame layer or the basic unit layer, depending on the settings provided to the encoder.

The method assumes that there is a single RoI, of any arbitrary shape, and that the rest of each frame is part of the background. It first considers how large a proportion of bits should be assigned to each
region, based on two factors – size and motion. The size is simply the ratio of the number of macroblocks in a region to the number of macroblocks per frame, as shown in (2-6).

\[ S_f = \frac{N_{mbf}}{N_{mbframe}} \]

\[ S_b = \frac{N_{mbb}}{N_{mbframe}} \]  

(2-6)

Above, \( N_{mbf} \) represents the number of macroblocks in the foreground, \( N_{mbb} \) represents the number of macroblocks in the background, and \( N_{mbframe} \) represents the number of macroblocks in each frame. The algorithm also considers the motion of each region, in terms of its motion vectors after prediction:

\[ M_f = \frac{\sum_{fg} |MV|}{\sum_{frame} |MV|} \]

\[ M_b = \frac{\sum_{bg} |MV|}{\sum_{frame} |MV|} \]  

(2-7)

In (2-7), \( M_f \) represents the sum of the magnitude of all motion vectors in the foreground over the sum of the magnitudes of all motion vectors in the frame; \( M_b \) represents the same ratio for the background.

Once the bit budget \( B_T \) for a frame has been established, the budget may be distributed between foreground and background macroblocks using (2-8).

\[ \tilde{B}_f = (\omega_S S_f + \omega_M M_f)B_T \]

\[ \tilde{B}_b = (\omega_S S_b + \omega_M M_b)B_T = B_T - \tilde{B}_f \]  

(2-8)

In (2-8), \( \omega_S \) and \( \omega_M \) are each scaling factors, between 0 and 1, that determine how much importance to accord the size and motion when distributing bits, respectively. They may be set based on properties of the video, or hardcoded within the encoder. Each of the steps thus far has simply determined initial bits for the foreground and background; no special priority has been given to the RoI yet. Equation (2-9) provides the means for doing so – some percentage of the bits in the background are reallocated to the foreground, giving the final budgets for each.
\[ B_f = \bar{B}_f + P\bar{B}_b \]
\[ B_b = \bar{B}_b - P\bar{B}_b = B_T - B_f \] (2.9)

P is a ratio, from 0 to 1, determining the proportion of bits to reallocate. It can be thought of as the amount of priority to give the RoI.

Because this method was originally designed for H.261, it does not allow the same granularity as H.264 rate control. However, the equations above all translate easily to the basic unit layer in H.264. Simply replace “frame” with “basic unit,” and the adaptation is complete. Each basic unit is processed in the same manner by the algorithm: an initial bit budget for the BU is determined, the size and motion of the foreground and background in the BU is determined, and bits are reallocated from foreground to background MBs accordingly. Once this step is complete, QPs can be calculated using the quadratic model described in (2-1) and each macroblock in the BU can be coded.

One final modification can be made to the algorithm. Because changing QP too rapidly can cause significant blocking artifacts, maximum and minimum bounds may be placed on the change in QP from macroblock to macroblock. In the JM Reference implementation, a maximum delta QP of 2 is specified; this implementation, like Method 1’s implementation, removes the constraint. QP values must still be clipped within the range [1, 51], however.

In terms of complexity, this method does not add significant requirements to the existing H.264 rate control algorithm. The main computations being added are in the motion vector summations, which generally aren’t already done by an encoder. This, along with the extra computations to reassign bits from background to foreground and memory accesses to check whether an MB is in the background or foreground, are the only significant additions to both time delay and computational complexity.

### 2.4.3 Method 3 – Multiple-Priority RoI Coding

The third technique targeting H.264 rate control, specified in [10], is more complex than Methods 1 or 2. It involves setting up a given RoI “shape,” and adjusting parameters such that several priorities are considered. Given a central RoI area, and a number of priority levels, Method 3 assigns the RoI the highest priority, and creates several regions of decreasing priority around it, before reaching the background and assigning it the lowest priority. Higher priority regions are quantized less by assigning larger bit budgets. In this manner, degradation of quality can be controlled gracefully. Furthermore, the
method is capable of automatically handling several RoI shapes, including rectangular, circular, and a “foveated” shape that is based on the shape of the human retina.

The method specifically targets a two-layer approach to Rate Control. This means that it does not consider basic units at all, only assigning bits at the GOP and frame levels. At the GOP layer, the modified bit budget for a given GOP is computed as:

\[
QP_i(1) = \max\left\{QP_{i-1}(1) - 2, \min\left\{QP_{i-1}(1) + 2, \frac{\text{SumPQP}_{i-1}}{NP_{i-1}} - \min\left\{2, \frac{N_{i-1}}{15}\right\}\right\}\right.
\]

\[
QP_{i,j}(1) = \max\left\{QP_{i-1,j}(1) - 2, \min\left\{QP_{i-1,j}(1) + 2, \frac{\text{SumPQP}_{i-1,j}}{NP_{i-1}} - \min\left\{2, \frac{N_{i-1}}{15}\right\}\right\}\right.
\]

(2-10)

\textit{QP}_{i,j}(1)\) represents the initial QP assigned to the i\textsuperscript{th} GOP, for the j\textsuperscript{th} priority level. \textit{SumPQP} represents the sum of the QP of all P pictures in the i\textsuperscript{th} GOP, j\textsuperscript{th} priority level. N and NP represent the number of frames and the number of P-frames in the GOP, respectively. (2-10) assigns the initial QP for a GOP based on the number of frames in the previous GOP and their QPs, and clips the result to within ±2 of the previous GOP’s initial QP.

At the frame level, [10] recommends using the same linear MAD prediction model utilized by H.264’s rate control algorithm, as well as the same quadratic model for QP computation. Bit distribution among priority regions is calculated using (2-11), below.

\[
T_{i,j}(k) = \left( w_s S_{i,j}(k) + w_{\text{MAD}} \text{MAD}_{i,j}(k) \right) \times \left( T_i(k) - m_{i,j}(k) \right)
\]

\textit{T}_{i,j}(k)\) is the relative size of the j\textsuperscript{th} region, in frame k of the i\textsuperscript{th} GOP, \textit{MAD}_{i,j}(k)\) is the normalized predicted MAD of the region, and \textit{w}_s and \textit{w}_{\text{MAD}} are weighting factors that influence the importance of size or distortion respectively, with the constraint that \textit{w}_s + \textit{w}_{\text{MAD}} = 1.

The same linear MAD model as the JM reference software is used for this technique, with one modification. Each priority region receives its own MAD prediction, rather than predicting the MAD of the entire frame (or each basic unit). Theoretically, this will keep the MAD prediction approximately as accurate as the JM reference model, assuming priority regions are roughly analogous to basic units. The
same principle is applied to the quadratic model used to calculate QP, with each region receiving its own model parameters.

To calculate bit reallocation, a priority constant $P_0$ is used. This constant can be supplied as an input, and is used to calculate the priority of all $n-1$ remaining regions, with the highest priority region being given priority $P_0$ using (2-12).

$$P_j = e^{-\frac{j}{3}} P_0, \forall j = 1, ..., n - 1$$  \hspace{1cm} (2-12)

Finally, a bit target can be assigned to each region using (2-13):

$$T_{i,j}(k) = P_j \left( \sum_{m=(j+1)n} T_{i,j+m}(k) \right)$$  \hspace{1cm} (2-13)

Once the target bits are known, the target QP can be computed and rate control proceeds normally.

From a complexity standpoint, Method 3 requires more resources than Methods 1 or 2. It allocates additional storage for the linear and quadratic models for each priority region, and also requires several more calculations – one per region, rather than one for the frame. However, since this method replaces the traditional H.264 basic unit layer, it can be thought of as roughly analogous to the JM rate control implementation, with $N$ basic units per frame (where $N$ is the number of priority regions), with a few extra computations added for RoI coding – the bit reallocations from foreground to background, for example. It was expected that the quality difference, especially in terms of human perception rather than PSNR, would be noticeable between Method 3 and Methods 1 or 2, due to the smoothing out of the QP changes from foreground to background.

2.4.4 Method 4 – Error Resilience via Nonlinear Transform

The authors of [11] have proposed the first technique selected by this project as an RoI method targeting error resilience. As opposed to reallocating bits through modifications of QP, this method instead aims to improve the error resilience of macroblocks within the RoI through redundancy.

Method 4 can be thought of as an intelligent implementation of slice redundancy. As opposed to simply retransmitting entire frames, this method pre- and post-processes a video sequence such that RoI macroblocks are copied – either replacing non-RoI macroblocks or artificially extending the size of the
video sequence to contain the redundant RoI. A nonlinear transform is applied to this effect, such that the encoder and decoder may have no knowledge that the RoI method is in use. This project limited the method to overwriting background macroblocks, and did not consider cases where the frame size was expanded. Figure 2.5 shows an example of an image before and after the transform is applied. Note that not only are RoI macroblocks copied to replace background macroblocks, the background that is being replaced is “squeezed” so as not to lose too much background information.

![Figure 2.5 Example of Nonlinear Transform with Background Loss](image)

The authors of [11] recommend an RoI tracking technique to use in conjunction with this method; however, as RoI tracking was not the focus of this project, no tracking method was utilized. Support for a mobile RoI with this technique is allowed, and was implemented in the modified software in a similar manner to FMO slice groups, with a maximum of one mapping per frame. This particular method limits the RoI shape to a rectangle, and so the RoI specification was further limited in the modified software to slice group mapping type 2 (foreground/background) with two slice groups.

The transform itself acts as a nonlinear coordinate transformation, creating an altered frame $g(x,y)$ from a source frame $f(x,y)$ using (2-14). The equation gives the transform applied in the $y$ direction; the same transform in the $x$ direction can be obtained by replacing $y$ with $x$ (and vice versa) in (2-14). The authors’ setup assumed that each slice was a contiguous row of macroblocks in the frame, and so applying the transform in the $x$-direction would be useless, as it would not provide any assistance should a whole slice be lost. This project assumed that individual macroblocks would be lost, and applied the transform in two dimensions. The parameters $x_{s, RoI}$, $x_{e, RoI}$, $y_{s, RoI}$, and $y_{e, RoI}$ are provided as inputs and represent the boundaries of the RoI. They are required to correspond with macroblock boundaries in
the original frame. Furthermore, the values $x_{ns,Roi}$ and $y_{ns,Roi}$ correspond to the starting positions of the Roi in the transformed frame. In effect, every macroblock in the original frame becomes a pair of macroblocks in the transformed frame, displacing the background and causing it to “shrink” to fit within the altered frame. Though the pair of macroblocks will not perfectly replace each other (due to different prediction methods being selected, for example) the correlation between them would likely be high enough that one could be used to replace the other with little mismatch. In [11], correlations between 0.88 and 0.98 were reported, with 1.00 representing a perfect match.

\[
y_g = \begin{cases} 
(y \mod 16) + m_{y,offset} \ast 32 + y_{ns,Roi} & \text{and} \\
(y \mod 16) + m_{y,offset} \ast 32 + y_{ns,Roi} + 16, & y \text{ in Roi} \\
y \ast s_{y,\text{top}}, & y < y_s \text{ and } x_{s,Roi} \leq x \leq x_{e,Roi} \\
(y - y_e,\text{Roi}) \ast s_{y,\text{bot}} + y_{ne,Roi}, & y > y_e \text{ and } x_{s,Roi} \leq x \leq x_{e,Roi} \\
y, & x < x_{s,Roi} \text{ or } x > x_{e,Roi}
\end{cases}
\]

\[(2-14)\]

where $s_{y,\text{top}} = \left( \frac{y_{ns,Roi}}{y_{s,Roi}} \right)$, $s_{y,\text{bot}} = \left( \frac{v_{\text{height}} - y_{ne,Roi}}{v_{\text{height}} - y_{e,Roi}} \right)$, and $m_{y,offset} = \left[ \frac{y - y_{s,Roi}}{16} \right]$.

The corresponding inverse transform is given in (2-15).

\[
\hat{y} = \begin{cases} 
(y_g \mod 16) + m_{y,offset} \ast 16 + y_{s,Roi}, & y_g \text{ in Roi} \\
\frac{y_g}{s_{y,\text{top}}}, & y_g < y_s \text{ and } x_{ns,Roi} \leq x_g \leq x_{ne,Roi} \\
\frac{y_g - y_{ne,Roi}}{s_{y,\text{bot}}} + y_e, & y_g > y_e \text{ and } x_{ns,Roi} \leq x_g \leq x_{ne,Roi} \\
y_g, & x_g < x_{ns,Roi} \text{ or } x_g > x_{ne,Roi} 
\end{cases}
\]

\[(2-15)\]

where $m_{y,offset} = \left[ \frac{y_g - y_{ns,Roi}}{32} \right]$.

The paper also introduced an error concealment method, should data be lost in either the foreground or background. In the interest of comparing the error resilience of this method to the other methods, no error concealment was implemented, except to replace a lost Roi macroblock with its redundant “twin.” This process, rather than attempting error concealment at the decoder, used the reconstructed output frames from the encoder.

The advantages of this technique are fairly straightforward – it can be applied in addition to H.264 encoding/decoding, without any encoder or decoder modifications. As such, the pre/post-processing
work could be offloaded to a separate hardware or software unit, if desired. The technique is more effective than simply transmitting slices redundantly, especially when the RoI is considered, because it does not need to retransmit entire frames, and in fact will transmit no more information than would appear in a single frame. The biggest disadvantage is that this technique requires that additional work be done before encoding and after decoding, meaning that the postprocessor must be present to recover the original video frames after decoding is complete. Background quality will also suffer degradation, even with zero errors in the bitstream, should Method 4 be enabled without frame size expansion.

2.4.5 Method 5 – RoI Coding with Flexible Macroblock Ordering

In comparison with the other methods, Method 5 was the least complex in terms of changes required to the JM Reference Software. The reason for this is that it utilizes FMO mapping type 6 (explicit mapping) to define a more complicated mapping for the RoI than for the background. Given the proposed changes to the FMO functionality of the JM software, this method may either support a fixed or a moving RoI, but its efficiency in terms of bitstream overhead increases with the motion of the RoI. Method 5 is specified in [12].

Effectively, Method 5 uses an explicit FMO mapping, where each macroblock’s slice group is identified individually, to combine two other FMO mappings – type 2 (foreground/background) and type 1 (dispersed). Type 2 is used to distinguish between RoI and non-RoI, and type 1 is used within the RoI to enhance error resilience. The total number of slice groups for a single RoI would be three. Two slice groups would be added for each additional RoI. Figure 2.6 shows the slice group mapping for a single region of interest.
In the figure, the entire background is placed within one slice group, and is given the highest slice group ID number. The RoI is split into two groups, mapped using the dispersed pattern. The black squares represent macroblocks within slice group 1, and the white squares represent macroblocks within group 0. By transmitting slices out of order in this manner, the error resilience of the RoI is improved, as loss of a single slice (or even a slice group) would at least spatially distribute the errors.

This method’s choice of mapping is of interest in that it provides less efficiency than simply using mapping type 1; the dispersed mapping type transmits only its type information and the number of slice groups, and the encoder and decoder compute the slice group IDs for each macroblock automatically. The primary benefit of this technique is in its ability to distinguish between the RoI and background – the background can be defined as the slice group with the highest ID, and all foreground regions can be found below it, with each pair of IDs belonging to a single rectangular region.

The authors of [12] present their own technique that involves offsetting the RoI quantization parameter by a certain amount. This technique does not allow for very close rate control, and is unlikely to perform well in a bandwidth-limited channel, especially when meeting the target bitrate is critical. The JM rate
control model provides more flexibility and control over the buffer regulation, and Methods 1 and 2 already provide similar functionality.

As stated before, the complexity of this method is minimal, and only requires the precomputation of the mapping for each frame. This precomputation only needs to be performed when the RoI moves, and so for a fixed RoI the mapping is transmitted in the first picture parameter set and the method is finished.

2.4.6 Method 6 – Explicit Spiral Interleaving

The final error resilience RoI method also focuses on Flexible Macroblock Ordering, but utilizes Redundant Slices as well. Rather than use a combination of FMO mapping types 1 and 2, the authors of [13] propose a combination of types 0 (interleaved) and 3 (spiral) that they term Explicit Spiral Interleaved (ESI). Figure 2.7 provides a basic example.

![Explicit Spiral Interleave (ESI) FMO Mapping](image)

Figure 2.7 Explicit Spiral Interleave (ESI) FMO Mapping [13]

Slices are ordered using an inward spiral pattern, similar to mapping type 3. The spiral pattern is also broken up using interleaving. The authors selected this pattern as they believed it would result in maximum spatial dispersal should a large number of macroblocks be lost. Spatial dispersal of errors is useful for two reasons – errors in a frame are less noticeable or annoying to human observers if they are dispersed, and error concealment tools are more effective if more neighboring macroblocks are available.

The authors also proposed utilizing redundant slices to retransmit all slices containing the region of interest. Though this introduces redundant bits into the bitstream, it provides a replacement for any lost macroblocks in the RoI.
One caveat to redundant slices is that the redundant representations are not required to use the same coding parameters as the original slices; they may, for example, use higher quantization to decrease the number of bits in the secondary slices. This does create a discrepancy between the encoder and decoder in the event the decoder needs to replace a missing slice with a redundant slice. Due to the increased complexity of coding the same slice twice using different parameters, this project will only consider the case where redundant slices use the same coding parameters as the corresponding primary slices.

The paper also included a dynamic model for using redundant slices based on the characteristics of the channel. Because this project presupposed no knowledge about the channel, except that it is error-prone and bandwidth-limited, this feature was not included in Method 6. Redundant slices were also unable to be used, as they were not part of the Baseline profile. This method was selected simply to gauge the effectiveness of its FMO mapping versus Method 5’s.

In terms of complexity, this method is very similar to Method 5. Very little additional computations are required to transmit the redundant slices; they mainly affect the size of the bitstream. And because the FMO mapping calculations are precomputed (at least in the event of a fixed RoI), there is little difference in overall complexity between the two.
Chapter 3  Design

3.1. Baseline Software Code Modifications

The code changes for this project took place in two phases. The first phase removed all unnecessary elements that increased the size of the code, the size of the executable, and made the code less readable. To complete this phase, all non-Baseline features were removed from the JM Reference Software, version 17.2. Several other features that were not particular to the Baseline Profile but were deemed unnecessary for the projects utilizing the code, including handling of non-YUV videos and H.264 levels used for coding high definition videos, were also removed.

Removing all non-Baseline features reduced the number of modifications that would be required to implement the chosen RoI coding techniques. It also simplified the computations required for RoI coding – for example, by removing the necessity of considering B-frames during encoding. Furthermore, the changes reduced the size of the executable by nearly 40%. This is important from a hardware/software codesign perspective, which future work may consider.

The removal process was approached on a per-feature basis. Each H.264 profile was examined in turn, and features exclusive to that profile that were not used in Baseline were removed. An attempt was made to target higher-complexity features first, and so the removal process first targeted the High Profile, followed by Main, then Extended. As seen in Figure 1.2, all remaining features would then belong to (or be compliant with) the H.264 Baseline Profile.

The following sections detail the specific features that were removed during each phase of the Baseline code modifications.

3.1.1 High Profile Feature Removal

The following features that were part of the High Profile(s) in H.264 have been removed from the JM software:

- **Video Formats** – All support for 4:2:2 and 4:4:4 chroma subsampling were removed, and samples (luma and chroma) were limited to 8 bits each. In other words, all features included in the Fidelity Range Extensions were removed.
- **Monochrome Video** – All support for the 4:0:0 subsampling format, in which only luma samples are provided, was removed. It is unclear why this feature was limited to the High profile, however.

- **Separate Color Plane Coding** – This feature was added alongside 4:4:4 subsampling, allowing each color plane (R, G, and B or Y, Cb, and Cr) to be coded separately. Basically this forced the encoder to afford all three planes the same priority as luma, where in Baseline H.264 the chroma components would take 1/4 the time (at the very maximum) of luma samples. Typically chroma samples take even less time to process, as they are not considered separately for motion estimation.

- **Intra-Only Profiles** – These profiles were defined in addition to the High Profiles, with the purpose of providing low-complexity, high-quality compression. These profiles cannot come close to the compression capabilities of the other H.264 profiles, but they can operate at very fast framerates and do not degrade video quality as much as predictive coding.

- **Quantization Scaling Matrices** – These customizable matrices allowed the QP values for individual samples to be scaled up or down, affording a greater degree of flexibility to the quantization process at the expense of more memory and computational complexity. The scaling matrices could also be adaptively controlled, as described below.

- **RDOQ** – Short for Rate-Distortion-Optimized Quantization, this set of algorithms allowed several quantizations to be performed for each block, using a different matrix each time. The best one in terms of both quality and compression would be selected, and the algorithm would keep track of its state, allowing it to adapt as the video is processed.

- **Cr and Cb QP Control** – When operating in the YUV color space (i.e. there are no separate color planes), this feature allows separate control of the Cr and Cb components. Typically chroma is simply quantized at an offset from the luma component. This feature allows a different offset to be applied to each chroma plane.

- **8x8 Transform Adaptivity** – This feature adds another set of transforms, both DCT and Hadamard, to the standard. The 8x8 transforms, as their name implies, operate on larger blocks of pixels and can provide higher compression of coefficients and better quality. The price is the cost of more than doubling the number of transforms supported in the standard, increased memory requirements (storing 8x8 matrices in addition to 4x4 and 2x2), and increased computational complexity (simplified 8x8 matrix multiplication).
• **Lossless Predictive Coding** – This feature provided a means of coding P- and B- pictures losslessly, in a more advanced way than simply removing quantization. Ordinarily H.264 provides lossless coding in the form of an IPCM mode, in which a macroblock is simply copied over into the bitstream. These features substantially outperform IPCM mode.

• **MVC** – Short for Multiview Video Coding, this set of features was added to the standard along with the Multiview High and Stereo High profiles. It supports 3D and multi-perspective video coding from two (in the case of Stereo High) or more (in the case of Multiview High) viewing angles. It is easily disabled in the code through the use of preprocessor directives.

### 3.1.2 Main Profile Feature Removal

A single feature was removed during this stage: CABAC. Short for Context Adaptive Binary Arithmetic Coding, it is an alternative that some syntax elements in H.264 can use – coded coefficients, for example, as opposed to utilizing standard variable-length codes. This feature requires the addition of an arithmetic coding engine, as well as storage of the 400+ context models used to code different syntax elements. Normally, H.264 utilizes CAVLC, for the majority of its syntax elements, and standard Exp-Golomb VLCs for several others.

### 3.1.3 Extended Profile Feature Removal

The following features that are part of the Extended Profile in H.264 were removed from the JM software:

• **Interlaced Video Support** – This is a set of features that allows interlaced output – the coding of each frame as two fields, and provides two forms of coding – PicAFF and MBAff, Picture and Macroblock Adaptive Frame-Field Coding, respectively. PicAFF allows each frame to be specified as progressive or interlaced. MBAFF allows each 16-wide by 32-high “macroblock pair” to be coded as (a) a pair of regular macroblocks or (b) a pair of field macroblocks. MBAFF requires substantial changes to certain steps in encoding, including a new zigzag scan for reordering coefficients and different treatment of slices, macroblock availability, and picture ordering.

• **B-Slices** – This feature, known as bi-prediction, has become standard in most DPCM CODECs. It allows for two lists, List 0 and List 1, of reference pictures (one in each “direction”). Each B-coded macroblock utilizes two references – both can be from List 0, both can be from List 1, or one can be from each list. Each reference uses the full range of motion estimation options.
available in P-coded macroblocks, from partitioning to subpixel refinement. It can provide much better compression performance, at the expense of motion estimation (the primary bottleneck in H.264) taking twice as long. This means that it generally isn’t used in real-time video coding.

- **Weighted Prediction** – This is very similar to bi-prediction, but for P-coded macroblocks. P-coding only uses List 0, and typically allows one reference per MB. Adding Weighted Prediction support allows P-MBs to use two references. A motion vector is formed for each one, and the final predicted motion vector is calculated as the weighted average of these. The feature adds complexity to P-MBs but can provide higher compression.

- **SI/SP-Slices** – These two slice types were created mainly for streaming video, allowing for efficient random access without affecting the video’s bitrate by a large amount.

- **Data Partitioning** – This is the major error resilience feature that the Baseline Profile does not include. It allows slices to be split into three partitions: A, B, and C. Partition A contains all of the vital data for the slice, while B and C are of lower priority. A slice can be reconstructed if partition A is error-free and either of partitions B and C are error-free. If additional error protection is supplied to partition A, the resulting bitstream can theoretically handle higher error rates than Baseline H.264.

- **Hierarchical Coding** – This feature is for the most part associated with B-coding, and allows a hierarchy of reference frames to be set up. The typical coding pattern for a Group of Pictures (or GOP, meaning all frames located between two IDR frames) is IDR-P-B-B-...-P-B-B-...-IDR. This option allows further customization of the GOP by specifying a hierarchy for B-frames. B-frames with higher levels receive higher priority. Several modes are possible, with the last being explicit GOP, described below.

- **Explicit GOP** – This feature is also associated with B-coding, and allows input parameters to specify an explicit pattern for how B- and P-macroblocks are coded. Typically a hierarchical coding pattern is used, as described above. It is useful when the specific features of the video being encoded are known ahead of time, such as when high-motion scenes will occur.

### 3.1.4 Removal of Additional Features

The features discussed above were all removed from the JM software because they are specifically not allowed in the Baseline Profile of H.264. The following features were also removed because they were unnecessary for low-complexity video coding, or were not needed for any of the tasks incorporating the modified software.
- **Decoder** – The decoder executable was removed from the modified code entirely. The decision to do so was based primarily on the fact that all tasks utilizing the software were interested in encoder performance. A separate code project was created incorporating solely the decoder, with a single code modification made to remove the constraints placed on QP delta.

- **Levels above 2.0** – These levels were deemed unnecessary for low complexity video coding, especially in low-bitrate environments. Level 1, the minimum allowed H.264 level, sets a maximum bitrate of 64 Kbps and a maximum resolution of QCIF (176x144) at 15 frames per second. Level 2.0 supports maximum bitrates of 2 Mbps, at resolutions of up to CIF (352x288) at 30 frames per second. Levels above this were required to support bitrates that were outside the scope of this thesis project. Table 3-1 provides a list of all the remaining levels and their parameters.

<table>
<thead>
<tr>
<th>Level</th>
<th>Max MB/second</th>
<th>Max MB/frame</th>
<th>Max Bitrate (Kbps)</th>
<th>Example Resolutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1,485</td>
<td>99</td>
<td>64</td>
<td>QCIF @ 15 fps</td>
</tr>
<tr>
<td>1b</td>
<td>1,485</td>
<td>99</td>
<td>128</td>
<td>QCIF @ 15 fps</td>
</tr>
<tr>
<td>1.1</td>
<td>3,000</td>
<td>396</td>
<td>192</td>
<td>QCIF @ 30 fps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CIF @ 7.5 fps</td>
</tr>
<tr>
<td>1.2</td>
<td>6,000</td>
<td>396</td>
<td>384</td>
<td>CIF @ 15.2 fps</td>
</tr>
<tr>
<td>1.3</td>
<td>11,880</td>
<td>396</td>
<td>768</td>
<td>CIF @ 30 fps</td>
</tr>
<tr>
<td>2.0</td>
<td>11,880</td>
<td>396</td>
<td>2,000</td>
<td>CIF @ 30 fps</td>
</tr>
</tbody>
</table>

- **Image Processing** – These options allowed images to be pre- or post-processed. They were deemed unnecessary as they increase the size of the code without providing useful video coding features, and can be performed by other software if necessary.

- **Alternative I/O Formats** – The JM Software originally provided support for TIFF and AVI input. Because the projects using this software only use raw YUV format video, the other formats were removed.

- **Quantization Offset Matrices** – Offset matrices allow explicit definition of the initial quantization offsets that are applied. Though allowable in the Baseline Profile, this feature was unnecessary for the projects utilizing the software, as it requires higher complexity without much advantage.
• **Explicit Sequences** – This functionality allowed each picture to be ordered explicitly using a configuration file, as opposed to ordering pictures via one of the built-in H.264 settings. The feature was removed because it added unnecessary code and complexity.

### 3.1.5 Baseline Modification Results

The non-Baseline features of the JM Reference software were estimated to account for approximately 40% of the software. Removing these features, one would expect to see a corresponding decrease in both the size of the source code and the space required by the compiled executable. In gathering these results, the following procedure was used:

1. The size of the original JM software was measured by compiling the encoder executable (lencod.exe) and examining its size.

2. The number of lines of source code in the original software was computed using a script that counted the total number of lines (including comments and whitespace) in all files in the `lencod` and `lcommon` directories. The `lcommon` directory corresponded to code that is common between the encoder and decoder.

3. The software was forked and modified to incorporate only Baseline functionality, by removing all features described above. Rather than delete the appropriate lines of code, all modifications were made by commenting out code, allowing for easy referral to “deleted” code for easily examining and reverting changes. Comments utilized specially-formatted tags.

4. A script was used that stripped out all specially-marked lines of code.

5. The size of the modified JM software was measured by compiling the encoder executable (the only executable in the modified project, also called lencod.exe) and examining its size.

6. The number of lines of code in the modified software was measured by counting all lines in the modified code directory. This is equivalent to step 2, as all decoder code was also removed from the Baseline software.
The results of the final code size reduction are shown in Table 3-2.

<table>
<thead>
<tr>
<th>Source Code Lines</th>
<th>Executable Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Software</td>
<td>104,392</td>
</tr>
<tr>
<td>Modified Software</td>
<td>60,909</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Code Lines</th>
<th>Executable Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.10 MB (2,210,816 bytes)</td>
</tr>
<tr>
<td></td>
<td>1.30 MB (1,391,616 bytes)</td>
</tr>
</tbody>
</table>

The reduction in source code was about 38.2%, and the reduction in executable size about 41.6%, both of which were close to the initial estimate of 40%.

It should be briefly noted that there were 60,909 lines of code total in the modified software, after removing all additional comments added during the modifications. These results did not consider any of the additional modifications discussed in the following sections.

### 3.1.6 Remaining Features

This section details the features that remained in the Baseline JM encoder after modifications. The list corresponds to the full set of features in the H.264 Baseline Profile.

- **4:2:0 YUV Video Formats** – These formats are the only ones supported by Baseline H.264. Furthermore, samples are limited to 8 bits each.
- **Core Transforms** – All core Hadamard and DCT transforms remain in the software. This includes a core DCT transform, a 4x4 Hadamard transform for DC luma coefficients, and a 2x2 Hadamard transform for DC chroma coefficients. The inverse of each transform is also included.
- **Core Quantization** – All quantization features from the Baseline profile are included, with additional adaptive rounding support, a feature that adaptively controls quantization for luma.
- **VCL, NAL, RTP** – The Video Coding Layer bitstream is the base output of H.264. The Baseline JM software also features the capability of creating Network Abstraction Layer bitstreams, as well as RTP bitstreams.
- **Distortion Options** – The JM software originally included Sum of Absolute Differences (SAD), Sum of Squared Errors (SSE), and Sum of Absolute Transformed Differences (SATD). All of these are still intact as distortion options.
- **Motion Estimation** – The full range of ME algorithms are still available, minus bi-prediction and Weighted Prediction. This includes Full Search, Fast Full Search, UMHEX, UMHEX Simple, and Enhanced Protective Zonal Search (EPZS). Full subpixel interpolation is also supported.
• **Rate-Distortion Optimization** – RDO is still fully supported in Baseline H.264. It attempts to code each MB using all possible modes, and selects the best one in terms of bitrate and quality.

• **VLC, CAVLC** – These are the only Variable-Length Coding options available in Baseline H.264.

• **Deblocking Filter** – The deblocking filter has been left unchanged.

• **FMO, ASO, Redundant Slices** – These error resilience techniques allow for the reordering of slices and macroblocks, and the retransmission of important slices.

Additionally, some features were left in the code that pertained to non-Baseline features. These segments were not thought to constitute a significant portion of the executable size. They include Supplemental Enhancement Information (SEI) features, some of which are not relevant to the Baseline Profile, and Video Usability Information (VUI) for the same reasons. Several unused structs and data members were also left in the code, due to data alignment errors that were encountered upon their removal. These structs were typically found to be dynamically allocated via `malloc()` in the code, and were not thought to consume a significant portion of memory during coding.

3.2. **Error Modeling Code Modifications**

As one of the focuses of this project was H.264 encoding in error-prone environments, a suitable means of incorporating errors into the H.264 bitstream was required. Any model designed for inserting transmission errors in video bitstreams, however, has several issues to contend with.

In a compressed video bitstream, the relative importance of every bit can be thought to increase roughly in proportion to the compression ratio. Of course, certain bits are more important than others, and in an H.264 bitstream some bits are vital to the operation of the decoder. Should an error occur in a single bit in the decoder, one of the following four cases will occur:

1) The bit error is in a non-vital syntax element, and the decoder proceeds as normal. The error does not significantly affect the decoding process.

2) The bit error is in a non-vital syntax element, but causes a deviation in decoder behavior. The decoded video is noticeably different from the encoder’s reconstruction.

3) The bit error causes an invalid VLC and the decoder cannot proceed. The decoder seeks a synchronization point (generally the next NAL unit) and proceeds, discarding a large portion of the video.

4) The bit error is in a vital syntax element, and the decoder cannot proceed. Decoding halts.
Case 1 is of little interest, as the decoded video is not significantly affected. As H.264 encodes the majority of its syntax elements using some form of VLC, the likelihood of case 3 occurring is fairly high. And as this project was concerned with gathering and comparing data across multiple videos, case 4 was to be avoided entirely.

The solution attempted to simulate a combination of cases 2 and 3, by creating two separate forms of errors – coefficient errors and motion vector errors. Both target different weaknesses in predictive video coding, without preventing the decoding process from completing. The solution also pseudorandomly inserted errors at a rate that closely matched the specified inputs, with additional benefits coming from its enhanced accuracy and reproducibility of tests.

Two input parameters – MBErrorRate and MVErrorRate – were added to the JM Reference Software in this phase of development. MBErrorRate specified a percentage of transform coefficients to be dropped and MVErrorRate specified a percentage of motion vector information to be dropped. The process of “dropping” such information is described below. Coefficient and motion vector information was always discarded for macroblocks as a whole; in other words, MBErrorRate is equivalent to the percentage of macroblocks whose coefficients have been discarded. Dropping data for whole macroblocks is in keeping with case 3 described above, though it is somewhat inaccurate. Typically in a packet-switched network video data would be transmitted as one or more NAL units per data packet. Each NAL unit would group coded pixel data as slices; as such, data would be dropped one packet (several slices) at a time. After implementing such an error model, it was found that for lower-resolution videos, dropping several slices from a single frame produced error rates that were too high for recovery or concealment.

The error modeling algorithm operated on each macroblock, and acted as follows:
<table>
<thead>
<tr>
<th>Pseudocode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. if (Macroblock is in IPCM or SKIP mode)</td>
</tr>
<tr>
<td>2. {</td>
</tr>
<tr>
<td>3. DropCoefficients = 0;</td>
</tr>
<tr>
<td>4. DropMotionVectors = 0;</td>
</tr>
<tr>
<td>5. return;</td>
</tr>
<tr>
<td>6. }</td>
</tr>
<tr>
<td>7. totalMBs = FrameNo * FrameSizeInMBs + CurrentMBNumber;</td>
</tr>
<tr>
<td>8. if (MBErrorRate &gt; 0 &amp;&amp; (!NoIntraErrors</td>
</tr>
<tr>
<td>9. {</td>
</tr>
<tr>
<td>10. if (FrameNo &gt; 4)</td>
</tr>
<tr>
<td>11. {</td>
</tr>
<tr>
<td>12. actualDropRate = numCoeffsDropped / totalMBs;</td>
</tr>
<tr>
<td>13. diff = actualDropRate - MBErrorRate;</td>
</tr>
<tr>
<td>14. CurrentMBDropRate = CurrentMBDropRate - diff;</td>
</tr>
<tr>
<td>15. CurrentMBDropRate = max(0, min(100, CurrentMBDropRate));</td>
</tr>
<tr>
<td>16. }</td>
</tr>
<tr>
<td>17. if (random(100) &lt; CurrentMBDropRate)</td>
</tr>
<tr>
<td>18. {</td>
</tr>
<tr>
<td>19. DropCoefficients = 1;</td>
</tr>
<tr>
<td>20. NumCoeffsDropped++;</td>
</tr>
<tr>
<td>21. }</td>
</tr>
<tr>
<td>22. }</td>
</tr>
<tr>
<td>23. if (MVErrrorRate &gt; 0 &amp;&amp; Macroblock is not INTRA)</td>
</tr>
<tr>
<td>24. {</td>
</tr>
<tr>
<td>25. if (FrameNo &gt; 4)</td>
</tr>
<tr>
<td>26. {</td>
</tr>
<tr>
<td>27. actualDropRate = numMVsDropped / totalMBs;</td>
</tr>
<tr>
<td>28. diff = actualDropRate - MVErrrorRate;</td>
</tr>
<tr>
<td>29. CurrentMVDropRate = CurrentMVDropRate - diff;</td>
</tr>
<tr>
<td>30. CurrentMVDropRate = max(0, min(100, CurrentMVDropRate));</td>
</tr>
<tr>
<td>31. }</td>
</tr>
<tr>
<td>32. if (random(100) &lt; CurrentMVDropRate)</td>
</tr>
<tr>
<td>33. {</td>
</tr>
<tr>
<td>34. DropMotionVectors = 1;</td>
</tr>
<tr>
<td>35. NumMVsDropped++;</td>
</tr>
<tr>
<td>36. }</td>
</tr>
<tr>
<td>37. }</td>
</tr>
</tbody>
</table>
The pseudocode described in Table 3-3 has several important properties. First, it ignores macroblocks coded in either IPCM or SKIP modes. The IPCM “prediction” mode represents a special case in which prediction is not performed – pixel data is transmitted directly by the encoder. The mode is used to specify an upper limit on the number of bits per coded macroblock, and is rarely used. As significant additional work would be required to drop IPCM coefficients, and they were found to occur with near-zero probability in any video sequence, they were ignored. Skip mode was also ignored, as it represented a macroblock that was not coded, and for which no information was explicitly transmitted.

The error insertion algorithm also attempts to meet the MBErrorRate and MVErrorRate percentages as closely as possible. To do so, it computes the actual percentage of coefficients/motion vectors that have been dropped, takes the delta between the actual and target percentages, and applies the difference to the probability of an error. Should the random number generator cause the drop rates to fall too high or low, this method should compensate. However, to avoid dropping several macroblocks in a row early in the video sequence in an attempt to meet the target, the first five frames were allowed to process with no compensation.

To actually drop coefficient data without interfering with the decoding process, some compromises were necessary. The primary goal was to alter the functionality of the encoder as little as possible during this phase, to avoid altering the bitstream unnecessarily and to avoid increasing or decreasing the encoding time by too large a factor, which may artificially influence the results of later work. Thus, the primary values that were altered were CPB, or Coded Picture Block, values. CPBs are used in H.264 in conjunction with entropy coding to reduce the number of bits required for residual transmission. For every commonly-occurring residual pattern there exists a small hard-coded CPB value. CPB values exist, for example, where residual DC coefficients are non-zero and all AC coefficients are zero. The coefficient dropping function utilized the zero CPB – a value specifying that all residual data is equal to zero. By setting the CPB to zero, later code modules would simply transmit the CPB and would ignore the (still valid) residual data, as there is no need to transmit individual residuals if the CPB is 0. Using this method has two added benefits. First, it makes no alterations to the actual coefficient data, meaning that the encoder can proceed to reconstruct frames normally, as though no errors had occurred; this was desired, as the project was interested in modeling errors during transmission, not encoding. Second, the decoder was capable of fully decoding the bitstreams containing errors, and would in fact be implementing a naïve form of error concealment by doing so – as nearly all pixel data is transmitted in the form of residuals, the decoder is in fact adding zero to the reference macroblock, and utilizing
neighboring pixels or previous frames to reconstruct the dropped coefficients. It can further be shown that zero values propagate through the transform and quantization processes, as well as their inverses. Thus, dropping coefficients via transmitting a CPB of zero is a safe and legitimate alternative to inserting errors in coefficient values after encoding. One further modification was made after initial results had been gathered: a ‘NoIntraErrors’ input parameter was created. This parameter acted as a Boolean that, when true, prevented errors from being inserted in any Intra frames in the video. This allowed the first (IDR) frame of the video to be preserved. The reason for doing so was the relative importance of IDR frames in video sequences.

Figure 3.1 CIF Coastguard Video with (a) Unmodified Encoder, (b) Frame 0, 13.3% coefficient error, (c) Frame 10, (c) Frame 50
For motion vectors, a similar process was used, although there exists no CPB analogue for MVs. Rather, motion information is transmitted in H.264 in the form of motion vector differences (the delta from the previous MB’s motion vectors). For every macroblock whose MVs were dropped, all motion vector difference values were set to 0, effectively forcing the decoder to use the same motion vector values as neighboring macroblocks.

There existed one final concern with regard to the selected error model – the artificial modification of the size of the encoded video. By setting CPB and motion vector difference (MVD) values to 0, the

Figure 3.2 CIF Foreman Video with 13.3% MV Error (a) Frame 0, (b) Frame 5, (c) Frame 50
bitstream would be reduced for every macroblock whose information is dropped. For MVDs, this loss can be considered negligible, as MVD values are often close to zero, with the exception of high-motion sequences and scene changes. For coefficients, the amount of artificial shrinking depends on the complexity of the original block; the more complex the block, the more bits artificially removed through coefficient dropping. Several tests were conducted to study this effect; by setting MBErrorRate to 26.6% and measuring the difference in frame bit counts, an average percent difference of 2.92% was obtained at QCIF resolution, and a difference of 1.78% was obtained at CIF resolution, using the ‘Foreman’ video sequence. All other values were kept equal: a target bitrate of 64 Kbps, 0% MV error, and rate control set to frame-layer (all other values were defaults). For some frames, the bit count was in fact observed to increase; it is unclear why this occurred, as debugging demonstrated equal prediction modes across all macroblocks in both sequences.

For sequences greater than 30 frames, and in all tests conducted on the error model, the difference between target error rates and actual error rates after coding were found to have a maximum error of 0.7%, for both coefficients and motion vectors.

3.3. Moving RoI Modifications

Though the H.264 JM Reference Software provided an implementation for Flexible Macroblock Ordering, allowing for fixed RoI coding, it did not possess the features necessary for specifying an RoI that moved from frame to frame. Thus further modifications were required to enable this functionality.

Additional input parameters were inserted into the Baseline Reference Software: MovingRoI and NumRoIMaps. If set to ‘1’, MovingRoI would enable FMO mappings that are capable of varying from frame-to-frame. NumRoIMaps was required to allow the encoder to allocate the correct amount of memory for its slice group maps, and to know how to parse the slice group configuration files.

The unmodified JM reference encoder accepted configuration files that specified the slice group mappings for FMO map types 2 and 6. For map type 2, each pair of lines would specify the top-left and bottom-right macroblock addresses for a region, in raster scan order, thus for a video with 2 regions of interest, a 4-line configuration file would be required. FMO mapping type 6 required one line per macroblock, specifying the slice group ID of that macroblock.

After the moving RoI modifications were completed, the formatting of these files was changed slightly. Not only could multiple mappings be specified, but the starting frame at which each mapping took effect
was required as well. This was specified on the first line where each region began. For a configuration file with a single region of interest, and two RoI maps, the configuration file would consist of six lines: for each region, the first line would specify the starting frame (always 0 for the first RoI map), and the second and third lines would specify the corners of the region.

The modified software would generate a Picture Parameter Set for each RoI map, and would transmit all parameter sets at the beginning of the H.264 bitstream. When the target frame for an RoI map was encountered during encoding, the encoder would specify the new PPS ID in the bitstream. In doing so, a means of creating moving RoIs was implemented, without significant code modifications and without altering the H.264 bitstream format.

3.4. Video Distortion Software

As stated in Chapter 2, to obtain more insightful results for the quality of videos produced through this project’s experiments, custom software was created for computing PSNR and SSIM values. In addition to computing PSNR and SSIM traditionally, the software included computations for Y, Cb, and Cr PSNR, as well as PSNR and SSIM of each individual region, should FMO be considered.

The program was designed to output textual information to the console window in a predictable format, so that its output could be easily parsed; doing so allowed it to be merged easily into the automated testing process discussed in Chapter 4. If Flexible Macroblock Ordering is not in use, the program outputs overall PSNR, Y, Cr, and Cb PSNR, and SSIM values for the video, as well as for each individual frame. If FMO is in use, it also produces per-region values for overall PSNR by region, and SSIM by region.

The software was written in C#, using the .NET Framework 4.0, and is only capable of processing videos using raw YUV 4:2:0 planar formatting. It also requires the same configuration files discussed in Section 3.3 that specifies the slice group mappings, should per-region analysis be desired.

3.5. RoI Coding Technique Modifications

Following implementation of all supporting software and the Baseline, moving RoI, and error model modifications to the JM Reference Software, the selected RoI techniques were implemented. These techniques, as described in Chapter 2, have been split into two categories – those targeting rate control and those targeting error resilience. The three techniques targeting rate control were implemented
through further modifications to the JM Reference Software; the remaining techniques targeting FMO and the nonlinear transform were implemented via a separate program written in C#.

One overall input parameter was added to the JM source code that allowed an RoI coding method to be selected. This value could be set to 0 (default) to disable RoI coding and allow the encoder to function normally, or be set to a value in the range [0,6] to select one of the RoI coding techniques. A value of 1 corresponds to RoI Method 1, 2 to Method 2, 4 to Method 3, and so on. Methods 4, 5, and 6 were included for completeness, and did not cause any alterations to the encoding process. Selecting one of these coding methods did however cause the encoder to check other input parameters for conformity. The encoder would ensure, for example, that FMO was enabled if either of Methods 4 or 5 were selected. The encoder would also allow only certain combinations of methods to be selected. Users could select any one of Methods 1, 2, or 3, combined with any one of Methods 4, 5, or 6. No other combinations were allowed.

Method 1 was implemented through straightforward modification of the JM Reference Software’s rate control code. No further input parameters were required; if enabled, Method 1 would select background macroblocks and force their quantization parameter to the maximum value, without altering any values in the rate control quadratic model.

Method 2 required the addition of several input parameters, and further alterations to the rate control code. The parameters OmegaM and OmegaS were added as a means of specifying the weight to assign to size and motion, as described in [9]. The weights were required by the software to equal 1.0 when added together. A third input parameter, P, specified the proportion of bits to reallocate from foreground to background.

Examination of the JM Reference Software source code revealed that significant effort would be required to decouple the necessary rate control variables enough such that Method 2 could be fully implemented. As such, the size and motion factors were computed as specified in [9], but the target bitrate computation was left unmodified. Rather, the QP values were altered directly, scaling according to a modification of Equations (2-8) and (2-9) wherein coefficients of all terms were scaled down by 0.25, to account for the quantization parameter’s more direct effect on distortion. These modifications to the original Method 2 implementation resulted in a less accurate bitrate target, but still allowed demonstration of the method’s premise. The method was further limited by specifying that rate control be performed at frame-level; this allowed all computations to be performed correctly.
Method 3 was implemented using a C# preprocessing program to generate the required FMO configuration files for the encoder. Using a rectangular RoI shape and five priority levels, the preprocessor generated a Type 6 (Explicit) slice group mapping with multiple priority tiers using the same Type 2 (Foreground-Background) mappings used by Methods 1 and 2. As Method 3 also targeted rate control, the decoupling issue affecting Method 2’s implementation reappeared. As such, an approximation to the target bitrate calculations was devised. After computing the target QP for a video frame, code for Method 3 would utilize this value as a “target average QP” to determine the QP values for each priority region. Using the RoI’s QP as a starting point, the QP of each priority region was increased by 2. Using a weighted average of each priority region, each set of possible QP’s was attempted, and the set with a weighted average closest to the target was selected. The weighted average calculation was based on the size of each region, and is shown below in (3-1).

$$QP_{\text{avg}} = \frac{QP_0 S_0 + QP_1 S_1 + QP_2 S_2 + QP_3 S_3 + QP_4 S_4}{N_{\text{mb,frame}}}$$  \hspace{1cm} (3-1)

Where $QP_{\text{avg}}$ is the weighted average of the frame’s priority region QP values, $QP_i$ is the QP of the $i^{th}$ priority region, and $S_i$ is the number of macroblocks in the $i^{th}$ priority region.

C# was selected as an implementation platform for the preprocessor and distortion programs due to its rapid development capabilities and robust debugging support, as well as due to the timing constraints involved in further examining and modifying the JM source code. These three techniques all utilized some form of preprocessing, and imposed no computational penalty on encoding each individual frame; there was an exception to this in some increased overhead added by Methods 5 and 6, but this was simply due to their use of the FMO feature.

Method 4 was implemented as a video preprocessor in C#, processing each frame using the nonlinear transform as specified in [11] and described in section 2.4.4. To avoid increasing video size after the transform, the RoI size was limited to half of the video dimensions in either direction, minus 2. An algorithm was employed to select the new RoI boundaries from the old, by expanding it in each direction. As macroblock errors were not limited to entire rows, as in [11], both the x and y coordinate transforms were applied during preprocessing. The end result was the creation of four redundant macroblocks in the processed frame for each macroblock in the RoI of the original frame. The inverse transform was similarly implemented to postprocess videos after decoding them. Figure 3.3 and Figure 3.4 demonstrate the transform and its inverse applied to one video frame.
Methods 5 and 6 were implemented similarly. By specifying either method in the command-line arguments, the C# preprocessor program would prompt for RoI parameters, including number of maps and the location of each region, and would construct configuration files for the lencod.exe encoder accordingly; these configuration files always utilized FMO mapping type 6 (explicit).
In the context of this project, testing refers to the experiments conducted using the various selected RoI coding techniques, the environment setup for gathering experimental data, and the automated testing process used to verify changes to the JM Reference Software.

4.1. Testing Environment

4.1.1 Hardware/Software Setup

All software testing was done in a 32-bit Microsoft Windows environment. Debugging was performed manually utilizing the Visual Studio 2010 debuggers for both C and C#. The various software programs utilized by the project were:

- Visual Studio 2010, for development and debugging of C and C# code
- .NET Framework 4.0, a requirement for running C# programs
- Microsoft PowerShell IDE, for script development
- lencod.exe, Idecode.exe, the JM Reference encoder (modified and unmodified) and decoder, version 17.2
- cs_dist.exe, custom software implemented for calculating video distortion in C#
- cs_preprocessor.exe, custom software implemented for pre- and post-processing videos and configuration files

Experiments were limited to two desktop PCs, one with a 2.66GHz dual-core CPU and 3.25GB of RAM, the other with a 2.66GHz dual-core CPU and 2.0GB of RAM. Consistency in encoding/decoding time was not a factor in this project, as encoding 150 video frames took an average of approximately 10 minutes for QCIF frames and 20 minutes for CIF frames, with a variation on the order of minutes.

4.1.2 Scripting Environment

Scripts were written to automate the running of the encoder and decoder for each experiment, the gathering of output data into a single source, and the regression testing of the modified lencod.exe executable as each set of modifications was made. Scripts were primarily written using PowerShell, with supporting scripts written using standard Windows command shell scripts.
The goal in creating the regression testing script was to define a fairly robust set of tests that verified the functionality of the Baseline code changes to the JM Reference Software. Test cases were selected to vary all important input parameters to the encoder, such as rate control parameters, motion prediction algorithm and features, use of FMO, and Rate-Distortion Optimization mode. All of these features and parameters are Baseline Profile compatible, and by testing them it was possible to compare the outputs of the Baseline encoder to those of the original encoder. This was done by first running the original encoder to generate a set of known “good” output bitstreams as well as reconstructed videos. The script would then automatically run each test on the Baseline version of the software, and do bitwise comparisons of each pair of outputs to ensure the correct functionality. The test script incorporated between 16 and 20 tests and took approximately 15 minutes per run. Variations on the script were created test fewer cases, longer sequences, etc. at each phase of the removal process. The script was also used as a regression test after the error model was inserted, the moving RoI features were added, and after each RoI technique was implemented.

The experiment automation script was written exclusively in PowerShell, and allowed experimental data to be gathered in batches using “test files.” Test files utilized a custom text format that specified arguments to the encoder, such as the default configuration file and input video files to be used, and the parameters to be varied during the test. The script would then execute the modified software using the parameters specified, sort the outputs into appropriate directories, run the (unmodified) JM decoder, use the cs_dist.exe application to gather distortion information, and parse all output files generated by these programs for experimental data. This data was placed into two different forms of comma-separated value (CSV) file for easy reference and observation: one file for the overall tests, and a per-frame results file for each individual test.

The experimental data gathered includes:

- All test input parameters (discussed below)
- Average bitrate
- Average I-Frame bitrate
- Average P-Frame bitrate
- PSNR for Y, Cr, and Cb values
- Overall PSNR
- SSIM Y, U, and V as output by lencod.exe
• Average QP information
• Average bits per frame (I and P)
• Encoding time
• Motion Estimation time
• Per-region PSNR and SSIM values, if applicable

Encoding time and motion estimation time were not used for any conclusions, as their variance was found to be too high, even while encoding the same video repeatedly. Additionally, the JM encoder has been observed to perform orders of magnitude slower than other commercial and open-source H.264 encoders, making them better measurements for speed performance.

For each encoded video frame, the following data was also gathered:

• Frame Number
• Frame Type
• Number of Bits
• QP
• PSNR (Overall, Y, Cr, and Cb)
• SSIM
• Encoding and Motion Estimation times
• Per-region PSNR and SSIM values, if applicable

The experiment script did not delete any textual outputs, which contain far more data than is listed here; these items were selected due to their relevance to RoI coding in terms of complexity and video quality.

4.2. Experimental Parameters

The parameters selected for this project were chosen to highlight rate-distortion performance for the various implemented RoI coding techniques in bitrate-constrained, error-prone environments. Five videos were selected for the experiments, with each video first being processed in a series of control group tests. The control group tests did not utilize any RoI coding techniques and instead simply varied rate control parameters, MBErrRate and MVErrRate, with a given set of encoding parameters kept at their default values.
The five test videos selected were:

- Akiyo
- Coastguard
- Crew
- Foreman
- Silent

These videos were selected for three reasons: all were available freely online, at http://media.xiph.org/video/derf/, all were available in both CIF and QCIF resolutions, and all possessed one or more features that may be considered regions of interest.
The range of default values used across the control tests are shown below, in Table 4-1.

<table>
<thead>
<tr>
<th>Table 4-1</th>
<th>Control Group Encoder Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Frames</td>
<td>150</td>
</tr>
<tr>
<td>Hadamard Transforms</td>
<td>ON</td>
</tr>
<tr>
<td>Search Algorithm</td>
<td>Fast Full Search</td>
</tr>
<tr>
<td>GOP Structure</td>
<td>IPPP...</td>
</tr>
<tr>
<td>Reference Frames</td>
<td>1</td>
</tr>
<tr>
<td>MV Resolution</td>
<td>Full-pel</td>
</tr>
<tr>
<td>Error Metric</td>
<td>SAD</td>
</tr>
<tr>
<td>Search Range</td>
<td>±16 (QCIF), ±32 (CIF)</td>
</tr>
<tr>
<td>Symbol Mode</td>
<td>CAVLC</td>
</tr>
<tr>
<td>RD Optimization</td>
<td>OFF</td>
</tr>
<tr>
<td>Target Bitrates (kbps)</td>
<td>64 (QCIF only), 128, 256, 512 (CIF only)</td>
</tr>
<tr>
<td>Error Rates</td>
<td>0%, 6.65% (CIF only), 13.3%, 26.6% (QCIF only)</td>
</tr>
<tr>
<td>Basic Unit Size</td>
<td>One Frame, One Row</td>
</tr>
<tr>
<td>RoI Parameters</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The set of control tests is the set product of each of the rows in Table 4-1; that is, each video was encoded using both QCIF and CIF resolutions, with target bitrates of 32 and 64 Kbps for QCIF videos, and 64 and 128 Kbps for CIF videos. Error rates for MB coefficients and MVs were varied as specified, and the Basic Unit size was set to one frame, row, and macroblock, respectively. Other specified parameters remained constant across all tests, and were similar to those used in [6] as a recommendation for tests of algorithmic complexity. Selecting these parameters therefore allowed this project’s results to be comparable to other literature that used the same recommendations. As an additional comparison point with the RoI coding techniques, an additional set of tests was performed, varying only target bitrate and frame size, that utilized a Type 2 (Foreground/Background) FMO slice group mapping to gather data on per-region PSNR and SSIM.

It should be noted that all of the tests that included errors enabled the “NoIntraErrors” option described in Section 3.2. In preliminary experiments, it was found that disabling this option caused the errors in the initial IDR frame to propagate to the surrounding frames, significantly increasing the error rate. The Intra-predicted blocks in the initial frame also introduced a “whitewashing” effect in certain videos, obscuring the results in other frames. This effect is noticeable in Figure 3.1. Table 4-1 also lists a 6.65% error rate for coefficient and motion vectors; this value was used in the CIF coding runs as a means of better demonstrating how error rates affected video quality, as preliminary tests found that motion vector errors caused considerable distortion in decoded videos.
The tests for each RoI coding method were selected to best characterize variations in the technique’s parameters. The majority of the experiments were performed with a fixed region of interest for each video, and a subset of the control group parameters.

As Method 1 (Maximum Bit Transfer) applied QP modifications at the macroblock level, the experiments for testing the method all used macroblock-sized basic units during rate control. Regions of interest were limited to rectangular areas specified as Foreground/Background FMO maps; these same maps were used in the control group experiments. Otherwise, the experiments matched the control group parameters.

Method 2 (Joint Bit Assignment) utilized the same set of parameters as Method 1, with the exception that basic unit sizes were constrained to frames rather than macroblocks.

For Method 3 (Multiple Priority Regions), a different slice group mapping was utilized. To specify each priority region as a separate slice group, the preprocessor software written in C# was utilized to convert the Foreground/Background maps used in testing Methods 1 and 2 into a Type 6 (Explicit) FMO slice group mapping. The slice group mapping used 5 priority regions. The RoI was specified as region 0, and contained the same region as the foreground in the previous Type 2 mappings. Regions 1 – 3 represented the other priority regions, and each was formed using a 1-macroblock rectangular boundary around its inner regions. Region 4 specified the background, and was composed of all remaining macroblocks. Figure 4.1 shows one hypothetical mapping for Method 3.
RoI Method 4 (Nonlinear Transform) utilized the same set of tests as Method 1, with some exceptions. First, basic unit size was set to 1 frame for all Method 4 experiments; as Method 4 targeted error resilience as opposed to rate control, varying rate control parameters was not necessary. Second, no slice group mappings were used. Rather, the C# preprocessing program was utilized to process 150 frames of each input video using the nonlinear transform specified in [11]. The preprocessor performed the transform in both the X and Y dimensions, X first, using two passes. Specification of the initial regions of interest was via Foreground/Background encoder configuration files, using smaller regions of interest than Methods 1, 2, and 3. The regions were made smaller due to the restrictions imposed by Method 4, discussed in Section 3.5. Method 4 tests were run only on the videos Akiyo, Foreman, and Silent. Coastguard and Crew were both found to have regions of interest that did not effectively fit within Method 4’s constraints; both videos possessed regions of interest that were too wide before being shrunk, and the region of interest captured too little detail of interest after shrinking. Finally, all
decoded videos were post-processed using the same transform in the reverse direction, utilizing the error-free reconstructed video files as references for error concealment.

Methods 5 and 6 (Checkerboard and Spiral Interleave) used the same set of experimental parameters. Basic unit sizes were set to one frame each, and the Method 1 experiments were repeated. Type 6 (Explicit) slice group mappings were used in both sets of experiments, with the only difference being the mapping itself. Method 5’s mappings were generated using the advanced FMO mapping discussed in Section 2.4.5, while Method 6’s mappings were generated using the Explicit Spiral Interleave mapping discussed in Section 2.4.6. Both mappings were generated via the C# preprocessing program.
Chapter 5  Results & Analysis

5.1. Accuracy of Error Model

A sample of experiments performed on the error model demonstrates that it produced coefficient and motion vector error rates within 1% of the specified target. Figure 5.1 and Figure 5.2 below demonstrate the accuracy of the coefficient and motion vector error models across control group and RoI coding method tests. The x axes in the figures represent the difference in error rate percentage, and the y axes represent the counts.

![Coefficient Error Rate Accuracy, 13.3% Target](image1)

![Coefficient Error Rate Accuracy, 26.6% Target](image2)

Figure 5.1  Coefficient Error Model Accuracy Histograms
As can be seen from the graphs, the error rate varied considerably more than the 1% maximum that was expected. Closer examination revealed that the higher error rates were due exclusively to lower target bitrates. The sole reason for the increased inaccuracy in error rate was due to the fact that the error models did not account for skipped macroblocks. Skipped macroblocks in H.264 do not transmit coefficients or motion vectors, and thus impose nearly 0 additional bits on the bitstream. At such low target bitrates, the frequency of skipped macroblocks increases, to a point where the majority of macroblocks are being coded in skip mode. At this point, the error model cannot drop coefficients or motion vectors, and cannot “catch up” unless a macroblock is no longer skip coded. This issue was more prominent in the CIF video runs, where bitrate was doubled, but frame size quadrupled. As skipped
macroblocks cannot have errors inserted in them, and merely propagate errors from the previous macroblocks, they can be thought of almost as an inherent error resilience mechanism. Errors in a skipped block depend on the integrity of the non-skipped blocks, placing a much higher importance on non-skipped blocks.

5.2. Results for Rate Control RoI Methods

5.2.1 Control Group

The first three RoI coding methods selected for this project targeted the rate control algorithm of either H.264 or one of its predecessors. When examining rate control algorithms, the important metrics are rate-distortion performance and target bitrate accuracy. The first metric can best be examined through the use of rate-distortion curves.

The control group experiments all utilized H.264’s default rate control algorithm, with two different basic unit sizes. The first set of experiments set the basic unit size equal to the size of a frame; the second set the basic unit size equal to a single row of macroblocks, as was recommended by [6] for a good tradeoff between bitrate adherence and rate-distortion performance. The rate-distortion curves for these two settings are shown in Figure 5.3 and Figure 5.4, using both PSNR and SSIM.
Figure 5.3  PSNR Rate-Distortion Curves for Control Group
As expected, the curves display a logarithmic relationship, with the input video used determining the y-offset of the curve. This relationship is commonly seen in plots of distortion versus bitrate. The SSIM curves in Figure 5.4 provide a better demonstration of the expected results than the PSNR curves in
Figure 5.3. Also of note is the fact that at the lower bitrates used in the experiments, the rate-distortion performance between row-level and frame-level rate control does not differ significantly. These results suggest that either of the basic unit size settings in the control group may serve as a valid comparison point to the RoI coding method results.

Figure 5.5 and Figure 5.6 together demonstrate the per-region distortion characteristics of the “Akiyo” and “Foreman” input videos. These videos were utilized as reference points for comparison to the rate control RoI methods, Methods 1 – 3. The foreground and background exhibit similar distortion
properties in both videos. The per-region PSNR plot of Akiyo and the per-region SSIM of the Foreman video both demonstrate higher background region quality than foreground region quality, due to the decreased motion of the background relative to the foreground; more effective motion estimation will contribute directly to higher PSNR and SSIM.

![Control Group Per-Region PSNR, "Foreman" CIF](chart1.png)

![Control Group Per-Region SSIM, "Foreman" CIF](chart2.png)

**Figure 5.6 Control Group Per-Region SSIM**

To gauge the accuracy of the rate controller, the mean deviation from the target bitrate was used. For the control group tests, the difference between the actual bitrate and the target bitrate was computed.
For each target, the mean deviation across all runs in the control group was taken. These results are shown below as Figure 5.7.

The deviation from the target bitrate is shown to be higher at lower target bitrates. This was to be expected; the H.264 encoder is biased toward skipping macroblocks over increasing quantization to the maximum amount, and has an inherent maximum amount of information that can be discarded for a given video. Thus, when encoding videos below a certain bitrate, it becomes more beneficial to reduce the framerate or resolution rather than increase quantization. Figure 5.7 also demonstrates that in the
control group experiments the encoded videos’ actual bitrates did not vary more than an average of 20% from the target for CIF videos; when not considering the lower target of 64 kbps, the average deviation is below 4%. The maximum recorded bitrate deviation across all QCIF control group experiments was 3.98%, and was 91.39% for CIF results. When filtering out 64 kbps targets, the maximum CIF deviation decreased to 9.79%.

5.2.2 Method 1

Method 1 modified the JM software’s default rate control algorithm such that all background macroblocks were set to maximum quantization, and the remaining macroblocks’ QP values were set by the standard rate control algorithm, decreased by a set ratio. As such, it was expected that the video’s distortion would decrease proportional to the size of the RoI, and that significant blocking artifacts would occur at the boundary between the foreground and the background. Figure 5.8 shows the overall rate-distortion curves for Method 1.
As compared to the control group’s results, the distortion becomes considerably higher when using Method 1. From video to video, the PSNR and SSIM both appear to vary considerably as well. This variation is influenced by the varying size and shape of the RoI, as well as the properties of the
foreground and background regions. The “Akiyo” video, for example, used a significantly smaller region of interest than the “Foreman” video, and thus might be expected to have worse quality. However, the video “Akiyo” contains less motion than “Foreman” as well, and this is most likely the reason for its higher PSNR and SSIM values.

Unfortunately, the increase in overall video distortion does not correspond to a gain in quality for the foreground, as seen in a sample frame from the Akiyo video in Figure 5.9. The overall decrease in quality of both the foreground and the background of the frame on the right should be noted. However, there is a noticeable difference in the quality of the facial region of the right-hand frame versus the background. Analysis of the per-region distortion confirms this to be the case.
Unexpectedly, distortion decreases sharply in the foreground for the “Akiyo” video at extremely low bitrates. This phenomenon was isolated to the “Akiyo” video, and was only seen at CIF frame sizes, and the cause was unclear. Similar analysis of the “Foreman” video, shown in Figure 5.11, demonstrates the expected per-region distortion curves. The other three videos’ per-region curves were more similar to “Foreman” than “Akiyo.”
The primary reason for the overall decrease in quality corresponds with the size of the RoI relative to the background. In “Akiyo” and other videos where the size of the RoI was small in comparison with the rest of the frame, a 75% decrease in the foreground QP was not sufficient to match setting the background region’s QP to the maximum value of 51. Figure 5.12 demonstrates this discrepancy by displaying the average QP of all P-frames in each video as target bitrate increases.

Figure 5.11  Method 1 Per-Region Distortion for CIF "Foreman" Video
As was the case with the rate-distortion curves, the average QP decreased as the size of the RoI increased relative to the rest of the frame. As Method 1 took few factors into account other than the
current QP and the region of the macroblock, this was to be expected. Figure 5.12 demonstrates that
the relationship between bitrate and QP for Method 1 is approximately linear for the range of bitrates
tested, beyond 64 kbps.

![Target Bitrate Accuracy, Method 1](image)

Figure 5.13  Method 1 Target Bitrate Accuracy

Method 1 also performed significantly worse in terms of conformance with the target bitrate. At 64
kbps, it performed with less accuracy, similar to the results seen in the control group. However, mean
deviation decreased steadily and eventually became negative as target bitrate increased. The reason for
this was the maximum QP value assigned to the background region; the amount of bits “saved” in doing
so remained constant, but the bit budget slowly increased. Thus, it is reasonable to assume that
somewhere after 64 kbps, the curve becomes linear, with the Method 1 encoder using fewer and fewer
bits than the target specified.

5.2.3  Method 2

Analysis of the rate-distortion characteristics of Method 2 revealed it to be more similar to the control
group’s results than to Method 1. Though the PSNR and SSIM were on average lower for Method 2 than
for the control group, the standard relationship between bitrate and distortion was restored by applying
the rate control algorithm’s target QP at the frame-level and adjusting bitrates relative to it. At QCIF
resolutions, the Coastguard and Akiyo videos did not display the same trend; the result is more
prominent in the graph of SSIM. This discrepancy was likely due to the smaller size of their regions of
interest, leading to increased overall QP. “Foreman” best showed the relationship, as it had foreground and background regions that were closer to equal in size, with higher degrees of motion (due to camera panning) in each.

![Method 2 PSNR Rate-Distortion Curves](image)

![Method 2 SSIM Rate-Distortion Curve](image)

Figure 5.14  Rate-Distortion Curves for Method 2
Transition from Method 1 to Method 2 also demonstrates more accurate rate-distortion for each region of the videos. Specifically, the Akiyo and Silent videos, which previously demonstrated more distortion in the foreground than background (or distortions that were approximately equal) now properly show an increase in foreground quality. In Figure 5.16 the Foreman video shows a similar trend, with a wider gap between the foreground and background distortions.
Target bitrate analysis of Method 2 also showed promising results. Though it shares the same issue at 64 kbps as both the control group tests and Method 1, Method 2 performed similarly to the control group at higher target bitrates, and closely matched the results of the control group tests with frame-level rate control enabled. It did show an increasing trend beyond 128 kbps, indicating an optimum point along the curve; the existence of this point suggests that by varying Method 2’s parameters, the method may be able to outperform the control group for a given input video – valuable information if the properties of the input video are known prior to encoding. More data would be required to prove this to be true, however.
5.2.4 Method 3

The rate-distortion characteristics of Method 3 differed significantly from both Methods 1 and 2. The curves appeared to be more constant than linear or logarithmic in nature, as would be expected. This was likely due to the constraints imposed upon Method 3’s algorithm: it sought to find a set of QP’s separated by 2 that conformed to the average video QP. Given such small frame sizes as CIF and QCIF, the expansion of the RoI at macroblock resolution places difficult-to-achieve conditions on the pseudo-rate-control algorithm employed. Figure 5.18 shows the rate-distortion curves for Method 3.

![Figure 5.17  Method 2 Target Bitrate Accuracy](image-url)
Figure 5.18  Rate-Distortion Curves for Method 3

Method 3 PSNR Rate-Distortion Curves (CIF Frame Size)

Method 3 SSIM Rate-Distortion Curves (CIF Frame Size)
The per-region results for Method 3 also display unexpected results: the regions are not ordered by distortion, as in previous experiments. The likely cause of this was the fact that regions were separated by a QP of 2, but had different size and motion characteristics. If the size and motion differences were
significant in comparison to the QP difference, then displaying distortion metrics by region will not display regions ordered by quality.

Method 3 also does not show signs of conforming to the target bitrate. Results showed that it in fact had the worst performance of the 3 rate control targeted methods. This was again due to the algorithm used, which enforced a QP difference of 2 between adjacent regions. At such low bitrates, such a constraint is not possible unless the size of each region exhibits specific properties.

The results for RoI coding method 3 indicate that at low bitrates achieving a “smooth” transition from foreground to background is not possible, except in a portion of videos, if the region of interest is small enough. Even if Method 3 were to be fully implemented, with each region receiving its own quadratic (QP) and linear (MAD) models, the result would likely be a series of large QP deltas, which would succeed in conforming to the target bitrate (comparable to the control group results), but would still contain significant blocking at region boundaries. The result would be equivalent to extending Method 2 for more than 2 regions.

![Target Bitrate Accuracy, Method 3](CIF Frame Size)

**Figure 5.20** Method 3 Target Bitrate Accuracy
5.3. Results for Error Resilience RoI Methods

5.3.1 Control Group

The RoI coding methods that target rate control, discussed previously, all showed indications of increasing the quality of the foreground relative to the background. However, they do nothing with regard to error resilience. Methods 4 – 6 focused on maintaining the quality of the RoI in the presence of increasing error rates. The error model discussed in Section 3.2 was capable of inserting errors in both macroblock coefficients and motion vectors for pseudo-randomly selected macroblocks within the encoded video stream. As such, the primary metric for gauging performance was PSNR and SSIM as each of these rates were increased. Figure 5.21 and Figure 5.22 demonstrate the control group’s performance for increasing coefficient error percentages. The videos “Akiyo” and “Foreman” are used in the figures.
Figure 5.21  Baseline Coefficient Error Response, "Akiyo" CIF
The figures above demonstrate increasing distortion as bitrate decreases; this effect has been previously shown in the rate-distortion curves. More importantly, the figures demonstrate the effect of coefficient errors on the distortion measurements—namely, that across all target bitrates introduction of coefficient errors causes an increase in distortion. The effects vary depending on the video used, but all five videos used for this project demonstrated results similar to those in Figure 5.21. The charts demonstrating effect on SSIM more reliably show the effects of error introduction than PSNR, and it can also be observed that for the PSNR graphs, there are diminishing returns; as more and more errors are inserted, the PSNR decreases less and less. For PSNR, this is largely due to the logarithmic nature of the
y-axis, but the diminishing effects can also be seen in the SSIM plots, albeit to a lesser extent. The results for CIF and QCIF frame sizes were largely similar in shape.

![Figure 5.23 Baseline Motion Error Response, "Akiyo" CIF](image-url)
Figure 5.23 and Figure 5.24 demonstrate the effects of motion vector error insertion into the video bitstreams for “Akiyo” and “Foreman.” As can be seen in the figures, the effect of motion vector errors is noticeably larger than that of coefficient errors. A surprising effect also emerges beyond 13.3% macroblock motion loss: distortion appears to remain roughly constant and in some cases decrease as the error rate is increased from 13.3% to 26.6%. The reason for this is not immediately apparent, but can be described as follows. At a certain motion vector error rate (which will differ from video to video), the insertion of additional motion vector errors has little effect on video quality. At that point, quality has degraded to a point where the video is no longer recognizable. As such, additional motion vector
errors are in effect moving existing pixels, sometimes into positions that cause distortion to decrease. It can be reasoned, then, that increasing motion vector error beyond this point will simply cause the distortion to vary randomly, possibly with some small decrease in quality. Additional motion vector errors should, however, also affect the rate at which video quality degrades. These effects are demonstrated in Figure 5.25.
The per-frame analysis shows that insertion of coefficient errors causes an approximately linear degradation in PSNR, with the slope of the line increasing in magnitude as the error rate increases. Motion vector errors have an earlier and larger effect on the video quality, but distortion eventually
stabilizes after approximately 60 frames (2 seconds) of video are coded. The reason for this stabilization is similar to the discussion above; around frame 60 the bitstream reaches “saturation” with regard to motion vector errors, and additional motion vector loss has little effect.

5.3.2 Method 4

Method 4 provided a unique solution to the problem of error resilience. It traded background quality for foreground macroblock redundancy. As such, the use of Method 4 in an error-free environment would result in an immediate reduction in video quality. Alternatively, the Method 4 transform could choose to expand the frame size to account for the additional macroblocks; this would require a larger bitrate or more quantization to account for the larger frame.

Background quality reduction is one of several issues with Method 4. Another is the inherent blocking artifacts introduced. Method 4 operates by creating copies of each macroblock, and spreading the copies out throughout the new (expanded) RoI. In doing so, it places macroblocks with “rough” edges adjacent to each other. The H.264 in-loop deblocking filter recognizes these edges as pre-existing, and will not smooth them out; this has the effect of nearly eliminating the benefit of the deblocking filter during encoding of the transformed frames. When the inverse transform is applied, the macroblocks are joined back together, but the blocking artifacts remain.

Figure 5.27 demonstrates both the background and blocking artifacts inherent to using Method 4 without frame expansion. The left-hand side of the figure is from the control group encoding of the “Akiyo” video using 512 kbps; the right-hand side of the figure is from the equivalent encoding run using Method 4, after preprocessing, encoding, and postprocessing the video frame.
Figure 5.26  Rate-Distortion Comparison, Method 4 vs. Control Group
Figure 5.26 shows the different rate-distortion curves for “Akiyo,” “Foreman,” and “Silent,” the three videos utilized for the Method 4 experiments. The same plots show the rate-distortion curves for the control group encodings of those three videos. First note that the shape of each Method 4 R-D curve matches the shape of the corresponding control group curve, indicating that the Method 4 transform does not significantly alter the properties of the video. Second, note that each Method 4 curve is offset in the negative y-direction from its control group counterpart, indicating a flat reduction in video quality after applying the forward and inverse Method 4 transforms.
Figure 5.28   Method 4 Coefficient Error Response, “Akiyo” CIF
Analysis of the effect on distortion as coefficient error rate increases demonstrates two interesting results. First, distortion appears relatively unaffected as error rate increases. This constant distortion indicates that the errors inserted into the low-quality background are having a small impact on video quality, and that the redundant foreground macroblocks are enhancing the bitstream’s error resilience. Second, video quality appears to increase as bitrate decreases, which is directly opposite what was expected. The reason for this effect is unclear; the increase in distortion is minuscule (on the order of 0.1 to 0.2 decibels per doubling of bitrate) and does not appear in all videos.
Figure 5.30 shows the error resilience properties of Method 4, at lower coefficient error rates. The top-left frame is with 0% errors, the top-right frame contains 6.65% errors, and the bottom frame contains 13.3% errors. Small discrepancies in the facial region of the frame indicate that the macroblocks came from various sources; namely, the redundant macroblocks created during the forward transform. In the “Akiyo” video, the background also contains very little motion, meaning that coefficient loss has little effect on the video quality, except in the foreground region. Figure 5.31 shows the same effects on the “Foreman” video, where a higher degree of motion causes the error resilience to begin to degrade beyond a 6.65% coefficient error rate.
Figure 5.31  Method 4 Coefficient Error Resilience, CIF "Foreman," Frame 60, 512 kbps

Figure 5.32 and Figure 5.33 display the response of Method 4 to motion vector errors. Unlike coefficient errors, motion vector errors appear to cause the video distortion to appear mostly random. Likely this is due to the effects discussed in Section 5.3.1; namely, motion vector errors propagating throughout the frame cause so many macroblocks to become “corrupted” that even a 4x redundancy in the foreground cannot improve error resilience. Figure 5.34 proves this to be the case.
Figure 5.32  Method 4 Motion Error Response, "Akiyo" CIF
Figure 5.33 Method 4 Motion Error Response, "Foreman" CIF
Figure 5.34  Method 4 Motion Error Resilience, CIF “Foreman,” Frame 30, 512 kbps

In the above set of frames, “Foreman” frame 30 is displayed with 0% motion vector errors, 6.65% motion vector errors, and 13.3% motion vector errors. As can be seen from the top-right frame, even a small motion vector error rate causes catastrophic loss of information within the RoI. Thus, Method 4 is no more effective than the Baseline JM encoder when dealing with motion vector loss at small frame sizes.

5.3.3 Method 5

The results for RoI coding method 5, in which a checkerboard-like pattern was used in the foreground, did not differ significantly from the baseline results. This was to be expected; the only coding parameter that was changed was the transmission order of the macroblocks in each frame. Such an adjustment can be expected to improve error resilience (from a human observer’s perspective) in the event of lost slices in the bitstream, where each slice is several contiguous macroblocks. However, in the case of random
macroblock and motion vector errors, both subjective and objective measurements of video quality do not differ from the control group. This was found to be the case for both QCIF and CIF frame sizes. The results for QCIF and CIF frame sizes are shown below.

For comparison with the control group, the videos “Akiyo” and “Foreman” have been used. The QCIF results are shown for “Akiyo,” while “Foreman” shows CIF results. The QCIF results are used to demonstrate effects at error rates above 13.3%, while the CIF results demonstrate that a similar effect exists even when the error rates are varied less.

**Figure 5.35**  Method 5 Coefficient Error Response, "Akiyo" QCIF

---

---
Figure 5.36  Method 5 Coefficient Error Response, "Foreman" CIF
Figure 5.37  Method 5 Motion Error Response, "Akiyo" QCIF

Method 5 Motion Vector Error Rate Analysis, Akiyo (QCIF Frame Size)

PSNR, Y (dB)

MV Error Rate (%)

0 13.3 26.6

256 kbps 128 kbps 64 kbps 32 kbps

Method 5 Motion Vector Error Rate Analysis, Akiyo (QCIF Frame Size)

SSIM, Y

MV Error Rate (%)

0 13.3 26.6

256 kbps 128 kbps 64 kbps 32 kbps
5.3.4 Method 6

Method 6 did not differ significantly in implementation from Method 5; it merely offered a wider range of options for the reordering of the foreground macroblocks. As this has already been demonstrated to be ineffective when random macroblocks are lost, the results are similar to both the control group and those of Method 5. Figures 5.39 – 5.42 are comparable to the equivalent graphs from the control group and Method 5 results sections.
Figure 5.39  Method 6 Coefficient Error Response, "Akiyo" QCIF
Figure 5.40 Method 6 Coefficient Error Response, "Foreman" QCIF
Figure 5.41  Method 6 Motion Error Response, "Akiyo" QCIF
Figure 5.42 Method 6 Motion Error Response, "Foreman" QCIF
Chapter 6 Conclusions

Method 1 demonstrated itself to be worse on all counts than the standard H.264 rate control algorithm. This was not unexpected; the naïve RoI implementation of Method 1 was simply not robust enough to perform well in a variety of circumstances. Performance was highly dependent on the size of the RoI relative to the frame size, and blocking artifacts were clearly visible in decoded frames. However, the target bitrate accuracy showed considerable savings in terms of bitrate. If a reduction in bit budget is desired without considerably affecting the quality of the RoI, Method 1 becomes a low-complexity option for doing so. It also becomes a viable alternative if low bitrates are necessary and a floating point arithmetic unit (and thus an accurate rate controller) is unavailable.

Method 2 provided a more robust RoI coding implementation than Method 1. While it failed to achieve the same quality measurements as the Baseline tests, this is a common trend among RoI coding techniques. Often the decrease in quality uniformly among background macroblocks will cause an overall increase in PSNR and SSIM, even though the average QP for each frame remains roughly constant.

The implementation of Method 3 used in this project attempted to demonstrate the tradeoffs between QP smoothness, video quality, and bitrate. At low bitrates and small frame sizes, initial results show that sacrificing one of the three is likely a necessity; Method 3 performed worse than even Method 1. Minor improvements could be made to the algorithm, adjusting the step size for QP to be larger than 2.

Methods 1 – 3 also highlight an additional fundamental issue with low-bitrate RoI coding: the requirement that the delta in QP between adjacent macroblocks be less than or equal to 2. While not a strict requirement in the H.264 standard, the JM decoder imposes the limit and will not decode any video breaking it. This project modified the decoder to circumvent the limit; however, if other decoders follow the same pattern, then any RoI method targeting rate control would likely have to restrict its choice of decoders to those that do not use such limits.

Method 4 proved capable of enhancing error resilience with respect to coefficient loss, but not motion vector loss. As I-frames constitute only a small portion of a video sequence for most uses of the H.264 encoder, Method 4 would prove ineffective at handling coefficient loss of more than 6.65% in videos that display a relatively large degree of motion. And due to the fact that bitstream errors are generally not relegated solely to either coefficient or motion vector loss, Method 4 is unlikely to perform well
without other redundancy methods in place to protect specifically against motion vector loss. It is worth considering the case of an Extended Profile H.264 encoder, which contains slice partitioning tools that allow vital portions of the slice header to be made redundant; such an encoder could be used to enhance motion vector resilience while increasing the bitstream size less than creating entire redundant slices.

Methods 5 and 6 showed no benefit versus the control group’s results, and would represent an overall worse implementation given the constraints on the environment. Use of FMO constitutes a (small) additional overhead added to the bitstream, and a slight increase in coding complexity, as each FMO map must be computed and stored. At such small resolutions as QCIF, CIF, and their equivalents, loss of a slice is either likely to be the loss of a large portion of a frame (if multiple macroblocks are transmitted per slice) or the loss of random individual macroblocks, which has been simulated here. In the case of the former, rearranging macroblocks during transmission will help only marginally, and only to a human observer. In the case of the latter, rearrangement will not help at all, as has been demonstrated here.
Future Work

This thesis demonstrated the use of several RoI coding techniques in low-bitrate, error-prone scenarios. In doing so, it placed several constraints on the encoding and decoding processes. As such, there are several areas that future work could focus on.

Videos were limited to QCIF and CIF frame sizes, which alone do not demonstrate how each RoI coding method scales with increasing frame size. Thus, a larger variety of input videos and resolutions would better verify the effectiveness of each. For larger frame sizes, the approximations of Methods 1, 2, and 3 would likely not perform as well, more specifically Method 1, which was “tuned” for those specific resolutions. Experimentation with other forms of subsampling than 4:2:0 would be beneficial as well; though the H.264 Baseline Profile does not support 4:4:4 or 4:2:2, future CODECs are likely to make increasing use of them as computational resources improve.

The speed performance of each RoI coding method also remains to be tested. As the JM reference software was found to be considerably slower and multiple PCs were required to perform all of the experiments in this project in a timely manner, valid timing results were not obtainable. Other open-source H.264 encoding solutions such as x264 have been shown to provide drastically better performance and have features comparable to the JM software; modifying such an encoder with the selected RoI techniques would provide a more suitable platform for speed performance testing. Likewise, attempting a hardware or hardware-software implementation of the H.264 encoder would provide more insight into how well the techniques perform on low-complexity platforms. However, as each of the techniques discussed here targeted either rate control or macroblock ordering, they would not be expected to increase encoding time significantly, as motion estimation remains the largest bottleneck in DCT-based video CODECs.

Finally, the experiments conducted during this project may be extended in two other ways: the use of moving regions of interest and the combination of different RoI coding techniques. Moving RoIs were demonstrated to be feasible even in Baseline Profile H.264 through the use of multiple Picture Parameter Sets. Furthermore, this technique does not impose additional complexity on the encoder, it merely creates additional overhead in the bitstream. Such moving regions would not be expected to alter the performance of any of the RoI methods discussed here; rather, it would allow for better control of the region of interest such that for a given frame, the foreground could contain fewer macroblocks. Doing so would create less of a discrepancy between the foreground and background as well as enhance
the quality of the overall video. By combining error resilience RoI techniques with quantization techniques, it becomes possible to create a low complexity encoder that is capable of producing videos of reasonable quality even in bitrate-constrained, error-prone environments.
References


Appendix

A Selected Control Group Charts

- **Control Group PSNR Rate-Distortion Curves (QCIF Frame Size, BU Size=99)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Video PSNR (dB): 20, 25, 30, 35, 40, 45, 50
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent

- **Control Group PSNR Rate-Distortion Curves (QCIF Frame Size, BU Size=11)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Video PSNR (dB): 20, 25, 30, 35, 40, 45, 50
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent

- **Control Group SSIM Rate-Distortion Curves (QCIF Frame Size, BU Size=99)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Video SSIM: 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 1.0
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent

- **Control Group SSIM Rate-Distortion Curves (QCIF Frame Size, BU Size=11)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Video SSIM: 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 1.0
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent

- **Control Group Average P-Frame QP (QCIF Frame Size, BU Size=99)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Average QP (P-Frames): 10, 15, 20, 25, 30, 35, 40, 45
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent

- **Control Group Average P-Frame QP (QCIF Frame Size, BU Size=11)**
  - Target Bitrate (kbps): 0, 32, 64, 96, 128, 160, 192, 224, 256, 288
  - Average QP (P-Frames): 10, 15, 20, 25, 30, 35, 40, 45
  - Curves for different video types:
    - Akiyo
    - Coastguard
    - Crew
    - Foreman
    - Silent
B  Selected QCIF Method 1 Results
C Selected QCIF Method 2 Results
D Selected QCIF Method 3 Results

Method 3 PSNR Rate-Distortion Curves (QCIF Frame Size)

Method 3 SSIM Rate-Distortion Curves (QCIF Frame Size)

Method 3 Average P-Frame QP (QCIF Frame Size)

Target Bitrate Accuracy, Method 3 (QCIF Frame Size)

Method 3 Per-Region PSNR, “Coastguard” QCIF

Method 3 Per-Region SSIM, “Coastguard” QCIF

Method 3 Per-Region PSNR, “Silent” QCIF

Method 3 Per-Region SSIM, “Silent” QCIF
E  

Selected Method 4 Results

- **PSNR Rate-Distortion Curves, Method 4 vs. Control Group (CIF)**
- **SSIM Rate-Distortion Curves, Method 4 vs. Control Group (CIF)**

- **Method 4 Coefficient Error Rate Analysis, Silent (CIF Frame Size)**
- **Method 4 Motion Vector Error Rate Analysis, Silent (CIF Frame Size)**
F Selected Method 5 Results

Method 5 Coefficient Error Rate Analysis, Coastguard (QCIF Frame Size)

Method 5 Motion Vector Error Rate Analysis, Coastguard (QCIF Frame Size)

Method 5 Coefficient Error Rate Analysis, Silent (QCIF Frame Size)

Method 5 Motion Vector Error Rate Analysis, Silent (QCIF Frame Size)
Selected Method 6 Results

Method 6 Coefficient Error Rate Analysis, Foreman (QCIF Frame Size)

Method 6 Motion Vector Error Rate Analysis, Coastguard (QCIF Frame Size)

Method 6 Coefficient Error Rate Analysis, Silent (QCIF Frame Size)

Method 6 Motion Vector Error Rate Analysis, Silent (QCIF Frame Size)