EFFICIENT CODING AND MAPPING ALGORITHMS FOR
SOFTWARE IMPLEMENTATION OF
A REAL-TIME H.263+ COMPLIANT LOW BIT RATE VIDEO CODER

by

BERNA EROL

B.Sc. (Control and Computer Engineering)

Istanbul Technical University, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENT FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1998

© Berna Erol, 1998
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical and Computer Engineering

The University of British Columbia
Vancouver, Canada

Date April 24, 1998
Abstract

The growing interest in video coding applications led to the development of video coding standards. Several successful standards have emerged, such as the ITU-T H.261, H.263, ISO/IEC MPEG-1 and MPEG-2. ITU-T recently announced its latest low bit rate video coding standard, the H.263 Version 2, also known as the H.263+ standard [25]. While this new low bit rate video coding standard provides better compression performance levels than those of the former ones, it is more complex and is more computationally demanding.

In this thesis, we develop coding and mapping algorithms that improve the video coding speed performance, making a reality the software implementation of interactive real-time low bit rate video coding on the Pentium MMX general purpose processor. First, using statistical properties of low resolution and slowly varying video sequences, we manage to significantly reduce the computation times of the most computationally intensive components of video coding, particularly the DCT, the IDCT, quantization and half pixel motion search. We also map some of the SIMD (Single Instruction Multiple Data) oriented functions onto Intel’s MMX architecture. The developed algorithms are implemented using our public-domain H.263+ encoder/decoder software [45]. After the algorithmic optimization and the MMX mapping, the resulting H.263+ video encoder implementation runs approximately 3 times faster than the original unoptimized public-domain encoder implementation. Moreover, our optimized H.263+ compliant video coder implementation can encode/decode more than 15 frames of a QCIF (176x144) video sequence per second without sacrificing video reproduction quality.
# Table of Contents

Abstract ............................................................................................................................. ii
Table of Contents ................................................................................................................ iii
List of Tables ......................................................................................................................... vii
List of Figures ....................................................................................................................... ix
Acknowledgments ................................................................................................................. xi
Chapter 1 Introduction .......................................................................................................... 1
Chapter 2 Background and Preliminaries ............................................................................. 4
  2.1 Introduction ................................................................................................................... 4
  2.2 Basics of Video Coding ................................................................................................. 4
  2.2.1 Motion Estimation and Motion Compensation ....................................................... 5
  2.2.2 Transform Coding, Quantization and Variable Length Coding ....................... 7
  2.3 The H.263 Video Coding Standard .............................................................................. 8
  2.3.1 Baseline Coding ....................................................................................................... 9
    2.3.1.1 Video Input / Output and Picture Formats ................................................... 9
    2.3.1.2 The Baseline Encoder .................................................................................. 10
    2.3.1.3 Decoder ...................................................................................................... 13
    2.3.1.4 Motion Vector Prediction in H.263 .............................................................. 14
    2.3.1.5 Forward and Inverse Quantization .............................................................. 15
    2.3.1.6 3D VLC Coding of Transform Coefficients ............................................. 16
      2.3.1.6.1 Coding of DC coefficients .................................................................. 16
      2.3.1.6.2 Coding of AC coefficients .................................................................. 16
    2.3.2 Coded Video Bit Stream Structure .................................................................... 17
  2.3.3 Negotiable Modes of H.263 .................................................................................... 18
    2.3.3.1 Unrestricted Motion Vector Mode (UMV) ............................................... 19
    2.3.3.2 Advanced Prediction Mode (AP) ............................................................... 19
    2.3.3.3 PB-frames Mode (PB) .............................................................................. 20
    2.3.3.4 Syntax-based Arithmetic Coding mode (SAC) ..................................... 21
List of Tables

Figure 1. Forward motion compensation ........................................ 6
Figure 2. Structure of a macroblock ............................................. 10
Figure 3. Block diagram of the H.263 encoder ............................... 11
Figure 4. Bilinear prediction in half pixel motion estimation .......... 12
Figure 5. Block diagram of the H.263 decoder ............................... 13
Figure 6. Motion vector prediction from surrounding macroblocks .... 14
Figure 7. Zig-zag scan order of a 8x8 block ................................. 16
Figure 8. Hierarchical block structure in a CIF image ..................... 18
Figure 9. PB frame structure .................................................... 20
Figure 10. Forward and bidirectional prediction for a B picture block .. 21
Figure 11. Neighboring blocks used for intra prediction in the Advanced Coding mode .................................................. 23
Figure 12. MMX architecture data types ....................................... 31
Figure 13. Multiply-add word to doubleword ................................ 33
Figure 14. Compare words (greater than) ...................................... 34
Figure 15. Unpacking a byte to a word ........................................ 34
Figure 16. Mapping MMX registers to floating-point registers ...... 37
Figure 17. Pentium and MMX technology pipeline structures .......... 40
Figure 18. Advanced Intra Coding Rate-Distortion Performance for AKIYO sequence ...................................................... 44
Figure 19. Reconstructed image for FOREMAN picture number 100 without (a) and with (b) Deblocking Filter mode and Post Filter set on ............................................ 45
Figure 20. Improved PB frames mode: Rate-Distortion performance for FOREMAN ...................................................... 46
Figure 21. (a) Chrominance and (b) Average PSNR performance for Modified Quantization Mode. ................................................................. 48

Figure 22. Encoding times for the H.263 and H.263+ modes for FOREMAN at 64 kb/s on a 200 MHz PC. ................................................................. 49

Figure 23. Block diagram of the H.263+ encoder implementation. ................................................................. 55

Figure 24. Flowgraph for fast 1-D DCT from Arai, Agui, and Nakajima [1]. ................................................................. 58

Figure 25. Zero block statistics for different predictors. ................................................................. 61

Figure 26. Number of computations of a fast IDCT for different sub zero block patterns. ................................................................. 67

Figure 27. Zero Sub block detection via bit stream parsing. ................................................................. 68

Figure 28. Half pixel motion vector prediction using 8 precomputed integer pixel SADs. ................................................................. 73

Figure 29. Half pixel motion vector prediction using 5 precomputed integer pixel SADs. ................................................................. 74

Figure 30. Half and integer pixel locations in a diamond shape area. ................................................................. 76

Figure 31. PSNR - number of iterations tradeoff in method 3. ................................................................. 77

Figure 32. Data flow to compute absolute value of differences of two unsigned data. ................................................................. 81

Figure 33. Flowgraph of multiplication and downscaling operations performed with MMX instructions. ................................................................. 85

Figure 34. Half pixel search block locations in interpolated reconstructed data. ................................................................. 88

Figure 35. Flowgraph of the data interleaving for MMX Half Pixel ME function. ................................................................. 89

Figure 36. H.263+ interpolation. ................................................................. 90

Figure 37. The inner core of MMX implementation of interpolation. ................................................................. 91

Figure 38. C routine to create an n byte aligned array. ................................................................. 93

Figure 39. Video telephony application over serial link using wireless modems. ................................................................. 108

Figure 40. VIC application using the H.263+ codec. ................................................................. 109
List of Figures

Figure 1. Forward motion compensation ................................................................. 6
Figure 2. Structure of a macroblock ................................................................. 10
Figure 3. Block diagram of the H.263 encoder ..................................................... 11
Figure 4. Bilinear prediction in half pixel motion estimation ................................. 12
Figure 5. Block diagram of the H.263 decoder ..................................................... 13
Figure 6. Motion vector prediction from surrounding macroblocks ...................... 14
Figure 7. Zig-zag scan order of a 8x8 block ....................................................... 16
Figure 8. Hierarchical block structure in a CIF image ........................................... 18
Figure 9. PB frame structure ........................................................................... 20
Figure 10. Forward and bidirectional prediction for a B picture block .................. 21
Figure 11. Neighboring blocks used for intra prediction in the Advanced Coding mode 23
Figure 12. MMX architecture data types ............................................................ 31
Figure 13. Multiply-add word to doubleword .................................................... 33
Figure 14. Compare words (greater than) ............................................................ 34
Figure 15. Unpacking a byte to a word .............................................................. 34
Figure 16. Mapping MMX registers to floating-point registers ............................ 37
Figure 17. Pentium and MMX technology pipeline structures ......................... 40
Figure 18. Advanced Intra Coding Rate-Distortion Performance for AKIYO sequence 44
Figure 19. Reconstructed image for FOREMAN picture number 100 without (a) and with (b) Deblocking Filter mode and Post Filter set on ........................................... 45
Figure 20. Improved PB frames mode: Rate-Distortion performance for FOREMAN 46
Figure 21. (a) Chrominance and (b) Average PSNR performance for Modified Quantization Mode.

Figure 22. Encoding times for the H.263 and H.263+ modes for FOREMAN at 64 kb/s on a 200 MHz PC.

Figure 23. Block diagram of the H.263+ encoder implementation.

Figure 24. Flowgraph for fast 1-D DCT from Arai, Agui, and Nakajima [1].

Figure 25. Zero block statistics for different predictors.

Figure 26. Number of computations of a fast IDCT for different sub zero block patterns.

Figure 27. Zero Sub block detection via bit stream parsing.

Figure 28. Half pixel motion vector prediction using 8 precomputed integer pixel SADs.

Figure 29. Half pixel motion vector prediction using 5 precomputed integer pixel SADs.

Figure 30. Half and integer pixel locations in a diamond shape area.

Figure 31. PSNR - number of iterations tradeoff in method 3.

Figure 32. Data flow to compute absolute value of differences of two unsigned data.

Figure 33. Flowgraph of multiplication and downscaling operations performed with MMX instructions.

Figure 34. Half pixel search block locations in interpolated reconstructed data.

Figure 35. Flowgraph of the data interleaving for MMX Half Pixel ME function.

Figure 36. H.263+ interpolation.

Figure 37. The inner core of MMX implementation of interpolation.

Figure 38. C routine to create an n byte aligned array.

Figure 39. Video telephony application over serial link using wireless modems.

Figure 40. VIC application using the H.263+ codec.
Acknowledgments

First of all, I would like to thank my husband, Arda Erol, for his great love and confidence in me, and for making my life easier during my heavy course load and research work.

My research would not have become a reality without the very valuable supervision of Dr. Faouzi Kossentini and Dr. Hussein Alnuweiri. Their guidance and support are greatly appreciated.

I would like to thank my colleagues Guy Côté, Michael Gallant and Ayman Elnaggar for sharing their valuable technical knowledge with me. I also wish to express my gratitude to Dr. Stephan Wenger for the useful reference materials he has provided me and our research engineer Dr. Alen Docef for making our lives easier with his great technical support. Last, but certainly not the least, I thank all my colleagues and friends for their invaluable friendship.
Chapter 1

Introduction

During the last decade, there has been a significant amount of interest in video coding and its applications such as video conferencing, video e-mailing and video telephony. In the late 1970's, the telecommunications industry had realized that continuous growth of audiovisual services is only possible with international standards. This necessity led to the standardization of a number of international video coding algorithms, yielding the ITU-T H.261, H.263, H.263 Version 2, and ISO/IEC MPEG1 and MPEG2 standards, that address a wide range of applications with different requirements, such as bit rate, quality, complexity, error resilience and delay.

The limited transmission rates supported by public switched telephone networks (PSTN) and wireless networks create a significant problem for digital video communication applications. When the currently available algorithms are used in very low bit rate applications (<64 kb/s), they mostly lead to very bad picture quality and blocking artifacts or require operation at low frame rates yielding low temporal resolution.

In recent years, ITU-T successfully established two video coding standards, the H.263 in December 1995 and the H.263 Version 2 (H.263+) in January 1998, that provide better performance levels than those achieved by the currently available video coding standards,
including the ITU-T H.261, at low bit rates. The basic configurations of these standards are based on the ITU-T Recommendation H.261. In addition to their baseline operation, H.263 and H.263+ have negotiable modes that improve rate-distortion performance and provide robust operation in the presence of channel errors, of course, at the expense of additional complexity, computation and memory requirements.

Until recently, real-time video encoding and decoding were only possible using Application Specific Integrated Circuits (ASICs) or multi DSP platforms, resulting in coding systems such as the VSP3 H.261 from NEC, the VDSP2 MPEG-2 from Matsushita Electric, and the Multimedia Video Processor (MVP), which employs the TMS320C80, from Texas Instruments [2]. Due to the growing interest in multimedia applications and the enormous computing power these applications require, many of the general purpose processors now have multimedia extension units where one instruction can be performed using more than one data input, simultaneously. Examples of such architectural extensions are the VIS extension of Sun’s Ultra Sparc [49], the MAX-2 extension of Hewlett-Packard’s PA-RISC [19], the MMX extension of Intel’s Pentium [21] and the Multimedia Instruction Set extensions of Silicon Graphics’ microprocessor [46]. These enhancements have greatly helped making real-time video applications in software a reality.

In this thesis, we propose two approaches for improving the speed performance of video coding, thereby supporting interactive real time video applications on the Pentium MMX general purpose processor. In the first approach, we develop platform independent efficient video coding algorithms that reduce the encoding time by using statistical properties of low resolution and
slowly varying video sequences. Such algorithms improve the computation times of the DCT, the IDCT, block matching (SAD), quantization and half pixel motion search functions significantly. The second approach is processor-specific, where we map several SIMD (Single Instruction Multiple Data) oriented components of video coding onto Intel's MMX architecture. Such components include the DCT, interpolation, SAD and half pixel motion vector search. All the algorithms and techniques proposed in this thesis are implemented using our public-domain H.263+ encoder/decoder software [45]. After the algorithmic optimization and MMX mapping, the H.263+ video encoder implementation runs approximately 3 times faster than our unoptimized public-domain encoder implementation. Moreover, the resulting H.263+ compliant video coding implementation can encode/decode QCIF video sequences at 15 frames per second on a Pentium MMX 200 MHz processor without sacrificing video reproduction quality.

The thesis is organized as follows. Chapter 2 provides an overview of the H.263 and H.263+ low bit rate video coding standards. It also introduces Intel’s MMX architecture and provides some brief information on its instruction set and superscalar architecture. The performance of a H.263+ compliant encoder, more specifically our public domain H.263+ video coding software [45], is discussed in Chapter 3. Chapter 4 describes the platform independent algorithms, which are developed and implemented, increasing the efficiency of H.263+ encoding. This chapter also discusses the complexity-performance tradeoffs associated with each algorithm. In Chapter 5, the MMX mapping algorithms of some of the compute intensive video encoder functions and their performance results are presented. Finally, conclusions and suggestions for future research are given in Chapter 6.
Chapter 2

Background and Preliminaries

2.1 Introduction

As stated earlier, the main objective of this thesis is to develop an efficient implementation of the H.263+ video encoding algorithm on the Intel’s MMX architecture. In this chapter, we first present basic concepts of video coding. This is followed by a discussion of the H.263 and H.263+ standards. This chapter concludes with a presentation of the basic features of the MMX architecture.

2.2 Basics of Video Coding

In most video sequences, there is usually little change from one video frame to the next one. Video compression and coding algorithms take advantage of such temporal redundancies by coding the difference between a predicted frame and the current frame instead of encoding the current frame itself. A compression scheme that employs only temporal redundancy reduction is referred to as an interframe coder. If the difference between two consecutive video frames is very large, then it is more advantageous to code the frame itself rather than the corresponding difference. Video compression algorithms reduce spatial redundancies within a video frame by employing transform coding as in still image coding. A compression scheme that employs only
spatial redundancy reduction without doing any prediction from the previous frame is referred to as an intraframe coder. The combination of intra and inter frame coding is called hybrid coding. In video coding terminology, an intra coded picture is called an I-picture. A picture that is predicted from the previous picture is called a P-picture. A picture that is bidirectionally predicted from both the previous and future pictures is referred to as a B-picture.

2.2.1 Motion Estimation and Motion Compensation

If the present picture is well predicted from the previous picture, then the difference between these two pictures will be quite small. In order to achieve a good prediction of the picture currently being encoded, the motion of the objects in the video sequence must be taken into account. Although a number of approaches have been investigated, the method that works best is block-based motion compensation. In this method, the picture is divided into blocks of size MxM pixels each. For each block, the previous picture is searched for the best match for the block. This search operation is called Motion Estimation (ME). The relative distance between the best matching block in the previous picture and the current block is called the Motion Vector (MV). Figure 1 illustrates the block based motion estimation procedure. Since it would be computationally very expensive search the entire picture for the best match, ME is usually performed using a limited window called a search window.
There are different measures that can be used to determine the best matching blocks, such as minimizing the mean squared error (MSE) and minimizing the mean absolute error (MAE) which are given by

\[
MSE = M_1(i, j) = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} (X_{m,n} - X_{m+i,n+j}^R)^2, \quad |i| \leq m_2, \quad |j| \leq n_1, \quad M = N, \quad (1)
\]

\[
MAE = M_2(i, j) = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} |X_{m,n} - X_{m+i,n+j}^R|, \quad |i| \leq m_2, \quad |j| \leq n_1, \quad M = N, \quad (2)
\]

where \(N\) and \(M\) are the dimensions of the block and \(X_{m+i,n+j}^R\) is the value of the block element that is located in row \(m+i\) and column \(n+j\) in the reference (previous) frame. Extensive investigations have shown that the MAE performs as well as the MSE in motion prediction [38]. Since the number of operations necessary for calculating MAE is much smaller than that of MSE, MAE error measure is more commonly used.
Motion estimation is the most time consuming part of all video encoders [38], [41]. Therefore, there are many efficient algorithms [2] (e.g. logarithmic search, hierarchical search) have been suggested in order to reduce the number of operations require to find the best matching block. Later in this chapter, we present a fast ME search algorithm [8], developed in our laboratory, that was recently included H.263+ Test Model [26].

2.2.2 Transform Coding, Quantization and Variable Length Coding

After the motion estimation is performed, the difference block between the best matching block and the current block is computed. Then a transform operation is applied to the difference block to reduce the spatial redundancies. Most of the coding algorithms use the Discrete Cosine Transform (DCT) because of its ability to greatly decorrelate signals and its smaller complexity compared to that of other transforms with similar performance, such as the Karhunen-Loeve-Hotelling (KLH) transform [38]. The definition of the 2-D DCT transform for an 8x8 block is given by

\[
y_{kl} = \frac{c(k)c(l)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} x_{ij} \cos \left( \frac{(2i+1)k\pi}{16} \right) \cos \left( \frac{(2j+1)l\pi}{16} \right)
\]

(3)

where \( k, l = 0, 1, ..., 7 \) and \( c(k) = \begin{cases} 
\frac{1}{\sqrt{2}} & \text{if } k = 0 \\
1 & \text{otherwise}
\end{cases} \)

Following transform coding, resulting coefficients are quantized and quantized DCT coefficients, motion vectors, and side information, such as the picture coding type and the quantizer step size are entropy coded using Variable Length Codes (VLCs). Quantization is a lossy coding method where VLC coding is lossless. After the quantization, the coefficient block
is reconstructed by dequantization and application of Inverse DCT (IDCT) and fed back to the coding loop to be used for prediction of temporally future blocks.

2.3 The H.263 Video Coding Standard

The objective of H.263 [24] is to provide significantly better picture quality at the same bit rate than the existing ITU-T H.261 video coding standard [23]. Due to the short standardization schedule, the H.263 was based on the existing H.261 technology. In principle, H.263 is network independent and can be used in a large variety of applications. However, it targets low-bit-rate networks such as the public switched telephone networks (PSTN) and wireless networks, and its target applications are video telephony and video conferencing. Therefore coding techniques that introduce high delays are not used in the H.263 standard.

In summary, the basic requirements of the H.263 standard were as follows:

- Use of available technology.
- Low complexity, thus low cost.
- Interoperability and coexistence with other ITU communication standards.
- Robust operation in the presence of channel errors.
- Flexibility to allow further extension.
- Flexibility of tradeoff between picture quality, complexity and bit rate.

The standard was finalized in December 1995. With its improved baseline coding and four negotiable modes, the H.263 compliant coders outperform the H.261 ones and it is predicted that the H.263 standard will replace the H.261 standard in many applications [16].
2.3.1 Baseline Coding

The baseline coding algorithm of H.263 is similar to that used by H.261, however there are some enhancements and changes to improve rate-distortion performance, including more flexible input picture formats, half pixel motion compensation, and modified VLC coding.

2.3.1.1 Video Input / Output and Picture Formats

The input to H.263 coder consist of non-interlaced pictures that are based on Common Intermediate Format (CIF) in order to provide a single recommendation for both 625 and 525-line television standards. However the standard itself does not define the conversion to CIF from video sources such as NTSC, PAL, SECAM etc. In H.263, the pictures are coded in YC\textsubscript{b}C\textsubscript{r} color format. The YC\textsubscript{b}C\textsubscript{r} format is specially developed by ITU-R for digital video coding. While Y represents the luminance component of the pixel, C\textsubscript{b} and C\textsubscript{r} represents the chrominance components. The YC\textsubscript{b}C\textsubscript{r} signals are scaled and offset versions of the YUV signals.

<table>
<thead>
<tr>
<th>Format</th>
<th>Number of pixels for luminance (Horizontal)</th>
<th>Number of pixels for luminance (Vertical)</th>
<th>Number of pixels for chrominance (Horizontal)</th>
<th>Number of pixels for chrominance (Vertical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub QCIF</td>
<td>128</td>
<td>96</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>QCIF</td>
<td>176</td>
<td>144</td>
<td>88</td>
<td>72</td>
</tr>
<tr>
<td>CIF</td>
<td>352</td>
<td>288</td>
<td>176</td>
<td>144</td>
</tr>
<tr>
<td>4CIF</td>
<td>704</td>
<td>576</td>
<td>352</td>
<td>288</td>
</tr>
<tr>
<td>16CIF</td>
<td>1408</td>
<td>1152</td>
<td>704</td>
<td>576</td>
</tr>
</tbody>
</table>

Table 1. Resolutions of the H.263 picture formats.
H.263 supports sub-QCIF, QCIF, CIF, 4CIF, and 16CIF input picture formats whereas H.261 supports only sub-QCIF and QCIF formats. These standardized picture formats and their spatial resolutions are listed in Table 1. For all of these picture formats, the luminance component sampling is done for each pixel in the picture and the chrominance component sampling is done for half of the pixels in each of the horizontal and vertical directions. The CIF, 4CIF, and 16CIF picture formats are optional for encoders as well as decoders. The encoder, however, should support at least one of the QCIF or sub-QCIF formats. Support for both of these formats, QCIF and sub-QCIF, is mandatory for decoders. This requirement of two mandatory formats for the decoder and only one for the encoder is a result of a compromise between high resolution and low cost. This way, the more expensive encoder will need to support only one of the resolutions.

2.3.1.2 The Baseline Encoder

Each picture in the input video sequence is divided into macroblocks. A macroblock consists of four 8x8 luminance blocks, one 8x8 C_b block and one 8x8 C_r block as shown in Figure 2. The motion compensated prediction operation is performed at the macroblock level.
Figure 3. Block diagram of the H.263 encoder.

(ME1: Integer pixel motion estimation and intra/inter decision; ME2: Half pixel motion estimation; M1: Input frame store; M2: Decoded frame store; PRED: Make prediction block; MBTYPE: Decided block type and block pattern; VLC(C): Variable-length coder for transform coefficients; VLC(M): VLC for motion vectors; CC: Coding control; DCT: Discrete Cosine Transform; IDCT: Inverse DCT; Q: Quantizer; IQ: Inverse Quantizer)

All pictures are inter coded in H.263 encoding except for the first picture. Even when a picture is inter coded, it can contain some macroblocks that are intra coded. The decision for each macroblock is made during the encoding process. Figure 3 shows the encoding diagram for the H.263 baseline encoder. First, an integer pixel motion vector is calculated for the 16x16 luminance block. Then, a comparison is made between the currently encoded block and the
displaced block in the previous picture. If the difference is small enough, the block passes through a motion compensated predictive coding process. If the difference is large, then the macroblock is \textit{intra} coded, i.e. the 8x8 luminance and chrominance blocks are transform coded with 2D DCT and then the transform coefficients are quantized and VLC coded.

The H.263 standard itself does not define the decision-making processes for \textit{inter} or \textit{intra} coding. There are test models defined for the standard and these test models define the decision making processes of all H.263 encoders.

After integer pixel motion estimation, if a block is decided to be \textit{inter} coded, then the motion search continues with half-pixel search around the current motion vector position. Half pixel values are found using bilinear interpolation as shown in Figure 4. Half pixel search is optional for H.263. Once motion estimation is completed, the difference between the motion compensated block and the current block is then DCT coded, quantized and VLC coded. Information bits for the motion vectors are also added to the bit stream.

\begin{align*}
A & \quad X_a & O_b & \quad X_b \quad \text{Integer pixel position} \\
& O_c & O_d & \quad X_c \quad \text{Half pixel position} \\
C & \quad X_c & X_d \\
\end{align*}

\begin{align*}
A &= a \\
B &= (A + B) / 2 \\
C &= (A + C) / 2 \\
D &= (A + B + C + D + 2) / 4
\end{align*}

\textbf{Figure 4. Bilinear prediction in half pixel motion estimation.}
Chapter 2: Background and Preliminaries

Similar to a Differential Pulse Code Modulator (DPCM), H.263 decodes and reconstructs the image in the encoder. Then the reconstructed picture is added to the current predicted picture in order to obtain the decoded picture, which is then stored in memory (M2).

2.3.1.3 Decoder

![Block diagram of the H.263 decoder.](image)

Figure 5. Block diagram of the H.263 decoder.

(M: Decoded frame store; MBTYPE: Decided block type and block pattern; VLD(C): Variable-length decoder for transform coefficients; VLD(M): VLD for motion vectors; IDCT: Inverse Discrete Cosine Transform; IQ: Inverse Quantizer)

Figure 5 shows a block diagram of the H.263 video decoder. First, the bit stream is parsed and variable-length decoded to obtain the coefficients, the motion vectors and other side information. After the coefficients are decoded by inverse quantization, IDCT is applied to the coefficient blocks. If the block is intra coded, the reconstructed block is equal to the result of the inverse
transformation. For inter coded blocks, the reconstruction is formed by motion compensation, i.e. summing the predicted block and the inverse transformed block.

2.3.1.4 Motion Vector Prediction in H.263

Motion vectors are coded using differential coding at the encoder. A median-based MV predictor is used in order to predict the motion vectors. The motion vector is obtained by adding predicted vectors to the difference vectors (MVD) at the decoder. In the case of one motion vector per macroblock, the candidate predicted motion vectors are taken from the three surrounding macroblocks as shown in Figure 6. The predicted horizontal and vertical motion vector components are calculated independently. For each component, the predicted value is the median value of the three candidates. The same motion vector is used for all of the luminance blocks. The motion vectors for chrominance blocks are obtained by dividing each of the component values of luminance MV by two, due to the 2:1 chrominance resolution.

MV: Current motion vector
MV1, MV2, MV3: The motion vectors that are used to predict MV.

Figure 6. Motion vector prediction from surrounding macroblocks.
2.3.1.5 Forward and Inverse Quantization

Depending on the prediction method selected at the encoder, coefficients that are going to be quantized can have a wide variation of statistics. In the case of intra prediction, the first element of the coefficient block, also referred as DC coefficient, take much larger value than the rest of the coefficients, which are also called AC coefficients. If the block is inter coded, then all the coefficients take smaller values. In order to efficiently quantize both inter and intra coded blocks, one quantizer is used for the intra DC coefficient and one of 31 predefined quantizers is used for the AC coefficients.

The decision levels of the quantizers are not defined within the standard. It is suggested, that the quantizer for the DC coefficient should be a uniform quantizer with a step size 8. Each of the other 31 quantizers use equally spaced reconstruction levels with a central dead-zone around zero and with a step size of an even value in the range 2 to 62. The absolute values of the reconstruction levels of non-zero coefficients are calculated as

\[
|\text{Reconstruction Level}| = \text{Quant} \times (2 \times |\text{LEVEL}| + 1) \quad \text{if Quant = "odd"}, \quad (4)
\]

\[
|\text{Reconstruction Level}| = \text{Quant} \times (2 \times |\text{LEVEL}| + 1) - 1 \quad \text{if Quant = "even"},
\]

where Quant is the quantizer step size and LEVEL is the quantized coefficient. Note that this process does not allow even valued numbers and prevents accumulation of IDCT mismatch errors. The sign of the quantized transform coefficient is signaled at the end of the VLC code word. After inverse quantization, the reconstruction levels of all coefficients other than the DC coefficient are clipped to the range -2048 to 2047.
Chapter 2: Background and Preliminaries

2.3.1.6 3D VLC Coding of Transform Coefficients

One of the main improvements that the H.263 standard introduces over H.261 is the new VLC tables and the 3D VLC coding technique.

2.3.1.6.1 Coding of DC coefficients

Since human visual system is more sensitive to blocking artifacts, which are mainly due to the DC coefficient, this coefficient is treated separately from the other 63 coefficients. The DC coefficient is coded differentially. The coding employs a first order predictor which is given by

\[ \text{DIFF} = \text{DC}_i - \text{DC}_{i-1}, \]  

where \( \text{DC}_i \) is the DC coefficient of the current block and \( \text{DC}_{i-1} \) is the DC coefficient of the previous block.

2.3.1.6.2 Coding of AC coefficients

After quantization, most of the AC values that represent higher frequencies become zero. To exploit this, before VLC coding, the 63 quantized AC coefficients are scanned in a zigzag scan order as illustrated in Figure 7.

![Zig-zag scan order of a 8x8 block.](image)

Figure 7. Zig-zag scan order of a 8x8 block.
In H.261, nonzero coefficients are called LEVELs and are coded using Huffman-like variable length coding. Because many AC coefficients become zero after quantization, runs of zeros along the zigzag scan direction are also identified and compacted. This is done by using RUN VLCs which indicate the number of the zero coefficients followed by a non-zero coefficient. If the rest of the AC coefficients are zero, then a VLC that represents an End Of Block (EOB) is used and coding is completed. The H.263 standard improves this technique by including an EOB for each of the coded AC-coefficients instead of separately coding it. Therefore, only one code represents the LEVEL of the coefficient, the number of the zero coefficients followed by a non-zero coefficient (RUN), and whether that particular AC coefficient is the LAST one of the non-zero coefficients. The VLCs are optimized so that blocks with very few non-zero AC coefficients are represented with short VLCs. Variable length codes are not defined for all the combinations of LEVEL, RUN and LAST. If a VLC is not found for a specific combination, the combination is coded by a 7 bit ESCAPE code followed by 1 bit LAST, 6 bit RUN and 8 bit LEVEL codes.

2.3.2 Coded Video Bit Stream Structure

The H.263 syntax for the coded video bit stream has a hierarchical representation, as shown in Figure 8, with four data layers, namely a Picture layer, a Group of Block (GOB) layer, a Macroblock (MB) layer and a Block (8x8) layer. Each layer is composed of data and corresponding header information. Picture properties, such as temporal reference, picture format, picture type, quantizer and use of optional modes are kept in the Picture layer. Each picture is divided into groups of blocks (GOBs) in order to enable quick resynchronization after
transmission errors. It is possible to update the quantizer at the GOB layer. The Macroblock layer consists of macroblock specific information such as motion vectors, \textit{Coded Macroblock Indication} (COD) which indicates if a macroblock is coded or not, \textit{Coded Block Pattern} (CBP) that shows which of the blocks are coded within a macroblock and macroblock level quantizer. Last, the block layer consists of 64 (8x8) VLC coded residual data.

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{hierarchical_block_structure.png}
\caption{Hierarchical block structure in a CIF image.}
\end{figure}

\section{Negotiable Modes of H.263}

In addition to the core encoding and decoding algorithms described above, H.263 includes four negotiable advanced coding modes: Unrestricted Motion Vectors, Advanced Prediction, PB-frames and Syntax Based Arithmetic Coding. The first two modes are used to improve inter
Chapter 2: Background and Preliminaries

picture prediction. The PB-frames mode improves temporal resolution with little bit rate increase. If the Syntax Based Arithmetic coding mode is enabled, arithmetic coding replaces the default Huffman-based VLC coding. These optional modes allow developers to tradeoff compression performance and complexity. In the following, we provide a brief description of each of these modes. A more detailed description of such modes can be found in [16].

2.3.3.1 Unrestricted Motion Vector Mode (UMV)

In baseline H.263, motion vectors can only reference pixels that are within the picture area. Because of this, macroblocks at the border of a picture may not be well predicted. When the Unrestricted Motion Vector mode is used, motion vectors can take on values in the range [-31.5, 31.5] instead of [-16, 15.5] and are allowed to point outside the picture boundaries. The longer motion vectors improve coding efficiency for larger picture formats, i.e. 4CIF or 16CIF. Moreover, by allowing motion vectors to point outside the picture, a significant gain is achieved if there is movement along picture edges. This is especially useful in the case of camera movement or background movement.

2.3.3.2 Advanced Prediction Mode (AP)

In this mode, overlapped block motion compensation is used for the luminance component of the P-pictures. Four motion vectors per macroblock, that is one vector per 8x8 luminance block, are allowed, instead of one motion vector per macroblock, when it is advantageous. Also, the motion vectors are allowed to point outside the picture as in the Unrestricted Motion Vector mode. The encoder has to decide which type of motion vectors to use. Four motion vectors require more
bits, but yield better prediction. The use of this mode generally provides a considerable improvement, and results in less blocking artifacts.

2.3.3.3 PB-frames Mode (PB)

\[
\begin{array}{cccc}
I & B & P & B & P & B & P \\
\end{array}
\]

Figure 9. PB frame structure.

In this mode, a frame structure consists of a P-picture and a B-picture as illustrated in Figure 9. The quantized DCT coefficients of the B- and P-pictures are interleaved at the macroblock layer such that a P-picture macroblock is immediately followed by a B-picture macroblock. Therefore, the maximum number of blocks transmitted at the macroblock layer is twelve rather than six. The P-picture is forward predicted from the previously decoded P-picture. The B-picture is bi-directionally predicted from the previously decoded P-picture and the P-picture currently being decoded as illustrated in Figure 10. The forward and backward motion vectors for a B-macroblock are calculated by scaling the motion vector from the current P-picture macroblock using the temporal resolution of the P- and B-pictures with respect to the previous P-picture. If this motion vector does not yield a good prediction, it can be enhanced by a delta vector. The delta vector is obtained by performing motion estimation, within a small search window, around the calculated motion vectors.

When decoding a PB-frame macroblock, the P-macroblock is reconstructed first, followed by the B-macroblock, since the information from the P-macroblock is needed for B-macroblock
prediction. When using the PB-frames mode, the picture rate can be doubled without a significant increase in bit rate.

Figure 10. Forward and bidirectional prediction for a B picture block.

2.3.3.4 Syntax-based Arithmetic Coding mode (SAC)

Baseline H.263 employs Huffman based variable length coding as a means of entropy coding. In this mode, all variable length encoding/decoding operations of H.263 are replaced with arithmetic encoding/decoding operations. Since VLC and arithmetic coding are lossless coding schemes, the SNR and reconstructed frames will be the same, but bit rate will be reduced by approximately 5% due to the more efficient arithmetic codes.

2.4 H.263 Version 2 (H.263+)

The objective of H.263 Version 2, also known as H.263+ in the standards community, is to broaden the range of applications, improve compression efficiency and improve error resilience. H.263+ offers many improvements over H.263. It allows the use of a wide range of custom source formats, as opposed to H.263, where only five video source formats defining picture size,
picture shape and clock frequency can be used. This added flexibility opens up H.263+ to a broader range of video scenes and applications, such as wide format pictures, resizeable computer windows and higher refresh rates. Picture size, aspect ratio and clock frequency can be specified as part of the H.263+ bit stream. Moreover, with 12 new negotiable modes, the H.263+ not only improves the rate-distortion performance but also increases the error resilience and provides a scalable syntax. Scalability can improve the delivery of video information in error-prone, packet-lossy, or heterogeneous environments by allowing multiple display rates, bit rates, and resolutions to be available at the decoder.

### 2.4.1 H.263+ Optional Modes

Next, the 12 new optional coding modes of the H.263+ video coding standard, including the modification of H.263's Unrestricted Motion Vector mode when used within an H.263+ framework, are described.

#### 2.4.1.1 Unrestricted Motion Vector Mode (UMV)

The definition of the Unrestricted Motion Vector mode in H.263+ is different from that of H.263. When this mode is employed within an H.263+ framework, new reversible VLCs (RVLCs) are used for encoding the difference motion vectors. Reversible VLCs are easy to implement, as a simple state machine can be used to generate and decode them. The idea behind RVLCs is that decoding can be performed by processing the received motion vector part of the bit stream in the forward and reverse directions. If an error is detected while decoding in the forward direction, motion vector data is not completely lost as the decoder can proceed in the reverse direction; this improves error resilience of the bit stream. Furthermore, the motion vector range is extended to
up to 256, depending on the picture size. This is very useful given the wide range of new picture formats available in H.263+.

2.4.1.2 Advanced Intra Coding Mode (AIC)

This mode improves compression performance when coding intra macroblocks. In this mode, intra-block prediction from neighboring intra blocks, a modified inverse quantization of intra DCT coefficients, and a separate VLC table for intra coded coefficients, are employed. Block prediction is performed using data from the same luminance or chrominance components (Y, Cr or Cb).

![Diagram of neighboring blocks used for intra prediction in the Advanced Coding mode.](image)

**Figure 11. Neighboring blocks used for intra prediction in the Advanced Coding mode.**

As illustrated in Figure 11, one of three different prediction options can be signaled: DC only, vertical DC&AC, or horizontal DC&AC. In the DC only option, only the DC coefficient is predicted, usually from both the block above and the block to the left, unless one of these blocks is not in the same picture segment or is not an intra block. In the vertical DC&AC option, the DC
Chapter 2: Background and Preliminaries

and first row of AC coefficients are vertically predicted from those of the block above. Finally, in the horizontal DC&AC option, the DC and first column of AC coefficients are horizontally predicted from those of the block to the left. The option that yields the best prediction is applied to all blocks of the subject intra macroblock.

The difference coefficients, obtained by subtracting the predicted DCT coefficients from the original ones, are then quantized and scanned differently depending on the selected prediction option. Three scanning patterns are used: the basic zig-zag scan for DC only prediction, the alternate-vertical scan (as in MPEG-2) for horizontally predicted blocks or the alternate-horizontal scan for vertically predicted blocks. The main part of the standard employs the same VLC table for coding all quantized coefficients. However, this table is designed for inter macroblocks and is not very effective for coding intra macroblocks. In intra macroblocks, larger coefficients with smaller runs of zeros are more common. Thus, the Advanced Intra Coding mode employs a new VLC table for encoding the quantized coefficients, a table that is optimized to global statistics of intra macroblocks.

2.4.1.3 Deblocing Filter Mode (DF)

This mode introduces a deblocking filter inside the coding loop. Unlike in post-filtering, predicted pictures are computed based on filtered versions of the previous pictures. A filter is applied to the edge boundaries of the four luminance and two chrominance 8 x 8 blocks. The weight of the filter's coefficients depends on the quantizer step size for a given macroblock, where stronger coefficients are used for a coarser quantizer. This mode also allows the use of four motion vectors per macroblock and motion vectors to point outside picture boundaries. The
above techniques, as well as filtering, result in better prediction and a reduction in blocking artifacts.

2.4.1.4 Slice Structured Mode (SS)

A slice structure, instead of a GOB structure, is employed in this mode. This allows the subdivision of the picture into segments containing variable numbers of macroblocks. The slice structure consists of a slice header followed by consecutive complete macroblocks. Two additional submodes can be signaled to reflect the order of transmission: sequential or arbitrary, and the shape of the slices: rectangular or not. These add flexibility to the slice structure so that it can be designed for different environments and applications.

2.4.1.5 Supplemental Enhancement Information Mode (SEI)

In this mode, supplemental information is included in the bit stream in order to offer display capabilities within the coding framework. This information includes support for picture freeze, picture snapshot, video segmentation, progressive refinement and chroma keying [25]. These options are aimed at providing decoder supporting features and functionalities within the bit stream. For example, such options will facilitate interoperability between different applications within the context of windows-based environments.

2.4.1.6 Improved PB-frames Mode (IPB)

This mode is an enhanced version of the H.263 PB-frames mode. The main difference is that the H.263 PB-frames mode allows only bi-directional prediction to predict B-pictures in a PB-frame, whereas the Improved PB-frames mode permits forward, backward and bi-directional
Chapter 2: Background and Preliminaries

predictions. Bi-directional prediction methods are the same in both modes except that, in the Improved PB-frames mode, no delta vector is transmitted. In forward prediction, the B-macroblock is predicted from the previous P-macroblock, and a separate motion vector is then transmitted. In backward prediction, the predicted macroblock is equal to the future P-macroblock, and therefore no motion vector is transmitted. Use of the additional forward and backward predictors makes the Improved PB-frames less susceptible to significant changes that may occur between pictures.

2.4.1.7 Reference Picture Selection mode (RPS)

In the H.263 baseline, a picture is predicted from the previous picture. If a part of the subject picture is lost due to channel errors or packet loss, the quality of future pictures can be severely degraded. Using this mode, it is possible to select the reference picture for prediction in order to suppress temporal error propagation due to inter coding. The information which specifies the selected picture for prediction is included in the encoded bit stream.

2.4.1.8 Temporal, SNR, and Spatial Scalability Mode

This mode specifies syntax to support temporal, SNR, and spatial scalability capabilities. Scalability means that a bit stream consists of a separately decodable base layer, and associated enhancement layers. This structure is especially desirable for error-prone and heterogeneous environments to counter limitations such as constraints on bit rate, display resolution, network throughput, and decoder complexity.
Chapter 2: Background and Preliminaries

Temporal scalability provides a mechanism for enhancing perceptual quality by increasing the picture display rate. This is achieved via bi-directionally predicted B-pictures, inserted between two P pictures and predicted from either one or both of these P pictures. SNR scalability is achieved by using a finer quantizer to encode the difference picture in an enhancement layer. This additional information increases the SNR, thus the quality of the overall reproduced picture. Spatial scalability and SNR scalability are closely related, the only difference is that spatial scalability provides an increased spatial resolution in the enhancement layer. Spatial scalability allows for the creation of multi-resolution bit streams to meet varying display requirements/constraints for a wide range of applications.

2.4.1.9 Reference Picture Resampling Mode (RPR)

This mode describes an algorithm to warp the reference picture prior to its use for prediction. It can be useful for resampling a reference picture having a different source format than the picture being predicted. It can also be used for global motion estimation, or estimation of rotating motion, by warping the shape, size and location of the reference picture.

2.4.1.10 Reduced Resolution Update Mode (RRU)

This mode allows the encoder to send update information for a picture encoded at a lower resolution, while still maintaining a higher resolution for the reference picture, to create a final image at the higher resolution. This is most useful in the case of movement over picture boundaries, motion of large objects and highly active motion scenes with detailed backgrounds.
2.4.1.11 Independently Segmented Decoding Mode (ISD)

In this mode, picture segment boundaries are treated as picture boundaries in the sense that no data dependencies across the segment boundaries are allowed. This includes estimation of motion vectors and texture operations across picture boundaries. Use of this mode prevents the propagation of errors, thus providing enhanced error resilience and recovery capabilities.

2.4.1.12 Alternative Inter VLC Mode (AIV)

Large quantized coefficients and small runs of zeros, typically present in intra blocks, become more frequent in inter blocks when small quantizer step sizes are used. When this mode is enabled, the intra VLC table designed for encoding quantized intra DCT coefficients in the Advanced Intra Coding mode can be used for inter block coding.

2.4.1.13 Modified Quantization Mode (MQ)

In H.263, the modification of quantizer value at the macroblock level is limited to a small adjustment (± 1 or ± 2) in the value of the most recent quantizer. The Modified Quantization mode allows the modification of the quantizer to any value, and thus provides the rate control methods more flexibility. Also, this mode increases chrominance quality significantly by using a smaller quantizer step size for the chrominance blocks relative to the luminance blocks. In H.263, when a quantizer smaller than 8 is employed, quantized coefficients exceeding the representable range of [-127, +127] are clipped. The Modified Quantization allows the representation of coefficients that are outside the range of [-127, +127], improving the picture quality at high bit rates.
2.4.2 Fast Motion Vector Search Algorithm

In the H.263+ Test Model, ITU-T suggests the use of one of two motion vector search algorithms: Full search or a fast search algorithm that was developed at UBC [8]. The fast search algorithm results in an encoder that is approximately 5 times faster than that using the full search algorithm, while still maintaining similar PSNR performance levels. The fast search algorithm proceeds by sequentially searching diamond-shaped layers, each of which contains the four immediate neighbors of the current search center. Layer \( i+1 \) is then centered at the point of minimum MAE, also referred as Sum of Absolute Differences (SAD), of layer \( i \). Thus successive layers have different centers and contain at most four untested candidate motion vectors. The search is stopped only when the following two conditions are met.

1. All candidate motion vectors in the current layer have been considered.
2. The minimum SAD value of the current layer is larger than that of the previous layer.

2.5 Intel's MMX Architecture

The computational demands in many of the multimedia and communication applications are enormous, due mainly to the large amount of data to be processed and the compute-intensive repetitive operations of the underlying processing algorithms. Recently, Intel has announced their development of the Multimedia Accelerator (MMX), which is an extension to their current Intel Architecture (IA) to improve the performance of communication, signal processing and multimedia applications. After analyzing a wide range of software applications, including graphics, MPEG video, music synthesis, speech compression, image processing, games, speech recognition, and videoconferencing, Intel scientists and others determined that although the
applications are different, the underlying compute intensive routines have common characteristics [36], [37]. These characteristics include:

- Small native data types (for example, 8-bit pixels, 16-bit pixels)
- Localized recurring operations
- Much inherent parallelism.

Taking the above characteristics into account, Intel hardware architects designed a Single Instruction Multiple Data (SIMD) architecture where one instruction performs the same operation on multiple data elements, simultaneously. This architecture provides, for most applications, an average processing speed increase of 150%-200% when compared to the speeds of the same applications run on the same processor without MMX.

2.5.1 MMX Data Types, Features and Instructions

Intel's MMX instruction set can be seen as an extension to the Intel Architecture (IA) instruction set. Besides providing 57 new instructions, MMX introduces four new data type.

2.5.1.1 MMX Data Types

In conventional Intel architecture processors, when processing 8 or 16-bit data, the existing 32 or 64-bit CPU band width and processing resources are actually underutilized. ALU, multiplier and other units only modify the low order 8 or 16 bits and higher order bits are unused. In order to eliminate this inefficiency Intel architects developed an architecture where small data elements can be processed independently, but simultaneously, thereby utilizing all the CPU processing units. The architecture is based on a maximum data width of 64 bits, as Pentium and P6 generations of processors use 64-bit wide busses.
Figure 12 shows the new data types supported by MMX architecture which are three packed data types (packed byte, packed word and packed doubleword) and a 64-bit quadword. In addition, the MMX architecture defines eight new 64-bit registers, each of which can be directly addressed using the register names MM0 to MM7. Notice that 64 bits can be used to represent a quadword, two packed doublewords, four packed words or eight packet bytes [21].

<table>
<thead>
<tr>
<th>Packed byte (eight 8-bit elements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 56 55 48 47 40 30 24 23 16 15 8 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packed word (four 16-bit elements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 48 47 32 31 16 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packed doubleword (two 32-bit elements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 32 31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quadword (64-bit element)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
</tr>
</tbody>
</table>

Figure 12. MMX architecture data types.

2.5.1.2 Saturation Arithmetic

One of the important features of MMX instructions is saturation arithmetic. In regular fixed-point arithmetic when an operation overflows or underflows, the most significant bit is lost. For example, the addition of two unsigned 16-bit numbers residing in a 16-bit register may result in an unsigned 17-bit result. Naturally, this number is too large to be represented in a 16-bit register and while the result’s low order 16 bits appears in the 16-bit register, the seventeenth bit does not fit and therefore lost. There is usually a status flag that shows overflow or underflow, by setting it
flag value to ‘1’. Unless a special attention is paid, this kind of behavior may cause serious problems, especially in graphics applications. However, in saturation arithmetic, when such an overflow or underflow occurs, instead of generating a 17-bit number and loosing the most significant bit, the corresponding instruction clamps the 17-bit result to the largest possible unsigned number that can be represented by 16 bits, i.e. FFFF in hexadecimal format. MMX supports two types of saturation arithmetic:

- **Unsigned saturation:** In the case of underflow, the register is set to ‘0’, and in the case of overflow, the register holds the largest number, i.e. $2^{n}-1$, where $n$ is the bit number.

- **Signed saturation:** If underflow occurs then the register takes the smallest number possible, i.e. $-2^{n-1}$, if overflow occurs the register takes $2^{n-1}-1$ ($n$: bit number).

In the MMX architecture, saturation arithmetic is not a mode which may be activated by setting a control bit. Instead, some instructions inherently perform saturation during their operation. For example, some MMX add instructions employ conventional wrap-around arithmetic while others employ saturation arithmetic.

### 2.5.1.3 MMX Instructions

MMX adds a new set of instructions that perform parallel operations on multiple data elements which are packed into a 64-bit register. Since most of the multimedia applications use 16-bit data, the new instruction set is optimized mostly for the 16-bit data type. The 8-bit data type (byte), is supported with the same instructions, with the exception of multiplication instructions, as the 16-bit word. Limited support is provided for 32-bit doubleword data. The MMX instructions are summarized in the following sections.
Chapter 2: Background and Preliminaries

2.5.1.3.1 Packed addition and subtraction with optional saturation

These instructions exist for byte, word and doubleword types. Each single add or subtract operation is independent of the others and takes place in parallel. The result can be a wraparound, unsigned saturation or signed saturation.

2.5.1.3.2 Packed multiplications

\[
\begin{array}{c|c|c|c}
A3 & A2 & A1 & A0 \\
\hline
\times & \times & \times & \times \\
B3 & B2 & B1 & B0 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c}
A3 \times B3 & A2 \times B2 & A1 \times B1 & A0 \times B0 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c}
A3 \times B3 + A2 \times B2 & A1 \times B1 + A0 \times B0 \\
\hline
\end{array}
\]

Figure 13. Multiply-add word to doubleword.

The multiplication instructions support 16-bit word data. There are two types of multiplication instructions. The first one performs four 16-bit x 16-bit multiplies and lets the user choose the low or the high order parts of the 32-bit multiply result. Therefore, both the input and the result are packed in 16-bit data types. The second type of multiplication is a multiply-accumulate operation. This instruction starts from a packed 16-bit data type and returns a packed 32-bit data type. It multiplies respective elements from both its sources and generates four 32-bit results. It adds two adjacent products producing a 32-bit result, and then adds the two other adjacent products to generate the final doubleword as illustrated in Figure 13. Therefore, this instruction performs four multiplies and two 32-bit accumulation operations in one instruction.
2.5.1.3.3 Packed compare instructions

These instructions compare packed 8 bytes, four 16-bit words, or two 32-bit double words simultaneously. Figure 14 shows a compare operation where the result is a mask of 1s and 0s depending on whether the condition is true or false.

\[
\begin{array}{cccc}
45 & 23 & 8 & 11 \\
> & > & > & > \\
22 & 64 & 41 & 2 \\
= \\
111...1 & 000...0 & 000...0 & 111...1 \\
\end{array}
\]

Figure 14. Compare words (greater than).

2.5.1.3.4 Packed shift instructions

There are two types of MMX shift instructions. The first one consists of regular arithmetic right, logical right and left shift operations that are performed on four 16-bit words or two 32-bit doublewords. The second type of shift operation is performed on the quadword and it can be either a logical left shift or logical right shift.

2.5.1.3.5 Conversion instructions

\[
\begin{array}{cccc}
B7 & B6 & B5 & B4 \\
B3 & B2 & B1 & B0 \\
\end{array}
\quad
\begin{array}{cccc}
A7 & A6 & A5 & A4 \\
A3 & A2 & A1 & A0 \\
\end{array}
\]

\[
\begin{array}{cccc}
B3 & A3 & B2 & A2 \\
B1 & A1 & B0 & A0 \\
\end{array}
\]

Figure 15. Unpacking a byte to a word.
These instructions perform conversions between the packed data types. This is especially important when an algorithm needs a higher precision to store its intermediate results. The *unpack* instruction unpacks a smaller precision data type to a higher precision data type. This instruction is also used as an interleaved merge operation as shown in Figure 15. This operation is very useful in many applications such as matrix transpositions, conversion between color planes and interpolation operations. The unpack operation is performed as a special case of this instruction where one of the operands is filled with zeros. In Figure 15, if the B register is filled with zeros then unpacking from a byte to a word is performed.

2.5.1.3.6 *Logical instructions*

Logical instructions are performed on 64-bit quadword. The logical operations that are supported are AND, OR, XOR and ANDNOT.

2.5.1.3.7 *Memory Transfer instructions*

Memory instructions provide data transfer between memory and the MMX registers, as well as data transfer between two MMX registers. These instructions operate on a 64-bit quadword as on a 32-bit doubleword. When a 32-bit doubleword is used, the low order 32 bits of a 64-bit MMX register is used for the data transfer.

2.5.1.3.8 *Empty MMX State operation*

This instruction is used to maintain compatibility with the conventional Intel architecture and it is covered in more detail in the “MMX Registers and IA Floating-Point Registers” section.
2.5.2 Backward Compatibility

Besides speeding up the multimedia applications, it is also very important that MMX processors retain backward compatibility with the existing architectures. All existing software and operating systems that runs on Intel processors have to run using the MMX-based processors without modification. Moreover, if MMX is supported, software programs should be able to detect it and make use of the added capabilities. Software developers can employ the “CPUID” instruction in order to detect existence of MMX technology within the processor. Executing the “CPUID” instruction sets a bit in the result, and thus during run-time, a software program can query the microprocessor to determine whether MMX is supported.

Compatibility with software that is written for conventional IA can not be achieved easily when it comes to operating systems that support a multitasking programming environment. New applications that use MMX instructions should be able to multitask with any other applications. During multitasking, the operating system performs a context-switching, i.e. it saves all the registers in memory and restores them when the next time that task gets to run. This places a constraint on MMX, in the sense that it cannot create new registers, new modes or new states. Otherwise, operating systems that are not designed for MMX-based architectures would not know these new registers or modes, and would not be able to save them during context-switching. Intel’s solution is to hide MMX registers and states into existing floating-point registers and states.
2.5.2.1 MMX Registers and IA Floating-Point Registers

For software transparency, an existing operating system views the MMX instructions as an extension of floating-point instructions. This is achieved by using the existing IA floating-point space for temporary storage for MMX data as shown in Figure 16. MMX instructions write values to the low order 64 bits of the 80-bit floating-point registers. When an MMX instruction accesses the registers, the rest of the 64 bits are set to "1". If a floating-point instruction accesses these registers at that point, it will see the values in the registers as NAN (Not A Number).

![Figure 16. Mapping MMX registers to floating-point registers.](image)

Sharing the registers with floating-point registers revealed some design problems. One problem was the stack architecture of floating-point registers. Since there is need for random access to MMX registers, the stack architecture is replaced by another architecture that provides random access to the registers. Another problem was that an application may require use of
MMX registers and floating-point registers. An application can use both of these registers as long as it does not use them simultaneously. Partitioning the floating-point and MMX codes into long execution blocks makes transition events infrequent and simple to handle.

The IA defines a tag field for each floating-point register. Each of these fields indicates whether the corresponding register is empty, full or a special case of a floating-point value, e.g. NAN. When a new value is pushed into a floating-point register, the corresponding tag field becomes set. When a value is popped off a floating-point register then the corresponding tag is not set anymore. The software convention followed by today’s IA programmers is to leave the floating-point stack empty after using it. Some operating systems employ floating-pointer tags to save only the valid floating-point registers during context-switching. Random access of MMX registers makes it difficult to implement the effect that the floating-point instructions have over the stack structured floating-point tags. This problem was solved as follows: the first time an MMX instruction accesses a floating-point register, the tag bits of all the floating-point registers are set so that all the registers will be saved during a context switch. Intel also supplies an instruction to clear all the floating-point tags. After completion of each MMX code section, application programmers should use the EMMS (Empty MMX State) command to unset all the tags.

2.5.3 The Superscalar Architecture of MMX

The Pentium processor is an advanced superscalar processor which means that it can handle two or more instructions simultaneously. It employs two general-purpose integer pipelines and a pipelined floating-point unit, allowing the processor to execute two integer instructions
simultaneously. Moreover, a software-transparent dynamic branch-prediction mechanism is used to minimize branching related pipeline stalls.

MMX instructions were designed to run using integer pipelines despite the use of the floating-point registers to hold data. Since MMX instructions operate on packed integer types, it makes sense to use integer pipelines. Pentium processors can issue two integer instructions, one in each integer pipe, in every clock cycle. The first pipe is referred to as the "U" pipe and the second as the "V" pipe. During decoding of any given instruction, the next two instructions are checked, and if possible, they are issued such that the first one is executed in the U-pipe and the second in the V-pipe. If it is not possible to issue two instructions, then the next instruction is issued to the U-pipe and none is issued to the V-pipe. In this case, the second instruction waits for the next cycle and becomes the first instruction in the next instruction pair. In a Pentium processor with MMX, two integer instructions, one integer instruction and one MMX instruction, or two MMX instructions can be executed simultaneously. MMX instructions, with the exception of the multiply operation, execute in one cycle. The multiply instruction is completed in 3 cycles, but because of the pipelined architecture, it is possible to issue one multiply instruction within every clock cycle.

2.5.3.1 MMX Pipeline Structure

The MMX pipeline structure adds additional stages to the conventional Intel architecture pipeline as shown in Figure 17. The basic integer pipeline structure of a Pentium processor consists of the following stages:

- PF: Instruction prefetch
Chapter 2: Background and Preliminaries

- IF: Instruction fetch
- D1: Instruction decode
- D2: Instruction paring and dispatch
- E: Execution and memory access
- WB: Write back

![Diagram of pipeline stages: IF, D1, D2, E, WB]

**Figure 17. Pentium and MMX technology pipeline structures.**

The pipeline structure for MMX instructions is different than that of the Pentium instructions.

The flowchart of MMX pipeline is given below:

- PF stage: Prefetches instructions.
- F stage: The prefetched instructions are parsed into instructions. The prefixes are decoded and up to two instructions are pushed into the FIFO. Two MMX instructions can be pushed if each of the instructions is less than or equal to 7 in byte length.
- D1 stage: Integer, floating-point and MMX instructions are decoded at this stage.
- D2 stage: Source values are read.
- E stage: The instruction is committed for execution.
- Mrw stage: Operands are read from memory.
- Mex stage: Execution stage for MMX instruction: ALU, shift, pack and unpack are executed and completed at this stage. This is the first clock cycle of multiply instruction.
Chapter 2: Background and Preliminaries

- WM/M2 stage: The single clock operations are written. This is the second stage of multiply operation.
- M3 stage: This is the third stage of multiply instruction.
- Wmul stage: Multiplier results are written.

Instruction parsing is decoupled from the instruction decoding with an “Instruction First In, First Out (FIFO) buffer”, which is situated between the F and Decode 1 (D1) stages as shown in Figure 17. The FIFO has slots for up to four instructions. This FIFO does not add additional latency when it is empty. During every clock cycle, two instructions can be pushed into the instruction FIFO (depending on availability of the code bytes, and on other factors such as prefixes). Instruction pairs are pulled out of the FIFO into the D1 stage. Since the average rate of instruction execution is less than two instructions per clock cycle, the FIFO is normally full. As long as the FIFO is full, it can buffer any stalls that may occur during instruction fetch and parsing. If such a stall occurs, the FIFO prevents the stall from causing a stall in the execution stage of the pipe. If the FIFO is empty, then an execution stall may result from the pipeline being "starved" for instructions to execute. Stalls at the FIFO entrance may result from long instructions or prefixes.
Chapter 3

Performance of H.263+Video Coding Standard

3.1 Introduction

In order to implement a real-time H.263+ encoder, it is first necessary to implement an H.263+ standard compliant encoder and decoder. At the beginning of this thesis work, the definition of the H.263+ standard was still in progress and UBC signal processing and multimedia group agreed to contribute to this work by providing a publicly available official ITU-T standard compliant software encoder/decoder (codec) [45]. Our software codec is based on a publicly available H.263 codec software which was developed by a company named Telenor from Norway. The H.263+ became an International Standard in January 1998 and our H.263+ encoder/decoder software is now very well known and widely referenced within the video coding community.

3.2 Performance Levels of Individual Modes

In this chapter, simulation results based on our implementation of an H.263+ encoder are presented. The results illustrate the tradeoffs between compression performance, complexity, encoding/decoding speed and memory requirements of each of the implemented modes: Advanced Intra Coding mode, Deblocking Filter mode, Improved PB-frames mode, Alternative
Inter. VLC mode, and Modified Quantization mode. Complexity and performance results on scalability and error resilience/recovery modes are discussed in [7] and [51] respectively.

The average peak signal-to-noise ratio (PSNR) of all encoded pictures is used as a measure of objective quality, and is given by

$$PSNR = 10 \log \frac{1}{M} \sum_{m=1}^{M} \frac{255^2}{(o_m - r_m)^2},$$

where $M$ is the number of samples and $o_m$ and $r_m$ are the amplitudes of the original and reconstructed pictures respectively. The test sequences that are used in the simulations have QCIF resolution and consist of 300 frames. Unless otherwise specified, rate control strategies are not employed. Instead, a quantizer step size is fixed for an entire sequence. Rate-distortion graphs are obtained by selecting different values for the quantizer step size. Also, unless otherwise indicated, all results are obtained using the fast search motion estimation implementation.

3.2.1 Advanced Intra Coding mode (AIC)

This mode significantly improves compression of intra macroblocks. Prediction lowers the number of bits required to represent the quantized DCT coefficients, while quantization without a dead zone improves the picture reproduction quality. This is illustrated in Figure 18, which presents the coding result for the first intra picture (i.e. where all the macroblocks are intra coded) of the Y-component of the video sequence AKIYO. Compression performance improvements of 15-25% are achieved. However, the Advanced Intra Coding mode only improves compression performance of intra coded macroblocks. Thus, negligible compression
improvements are achieved for low activity video sequences, where most macroblocks are inter coded.

![Advanced Intra Coding Mode: First Intra frame of Akiyo](image)

**Figure 18. Advanced Intra Coding Rate-Distortion Performance for AKIYO sequence.**

Based on our implementation, the associated encoding time increases by 5% on average, due to the prediction method selection operations. This mode requires slightly more memory to store the reconstructed DCT coefficients, needed for intra prediction. The increase in decoding time is negligible, as only a few additions are required to predict an intra coded macroblock.

### 3.2.2 Deblocking Filter mode (DF)

The Deblocking Filter mode improves subjective quality by removing blocking and mosquito artifacts common to block-based video coding at low bit rates. The effects of the deblocking filter are more pronounced when combined with the post filter described in the TMN-8 model [26]. This post filter is usually present at the decoder and is outside the coding loop. Therefore, prediction is not based on the post filtered version of the picture. To illustrate the improvement in subjective quality, the sequence FOREMAN was encoded using both the deblocking and post
filters at 24 kbps and 10 fps. Figure 19 shows the reconstructed image for picture number 100. The Deblocking Filter mode allows the use of four motion vectors per macroblock. This requires additional motion estimation, increasing the computational load, and thereby resulting in a 5-10% additional encoding time.

![Reconstructed image for FOREMAN picture number 100 without (a) and with (b) Deblocking Filter mode and Post Filter set on.](image)

**Figure 19.** Reconstructed image for FOREMAN picture number 100 without (a) and with (b) Deblocking Filter mode and Post Filter set on.

### 3.2.3 Improved PB-frames mode (IPB)

The PB-frames mode of H.263 can double the picture rate without significantly increasing the bit rate. The increase in bit rate is small due mainly to bits saved by coarser quantization of B-macroblocks. While this causes the B-picture to have a lower quality than the P-picture, the increased temporal resolution results in much better overall subjective quality. The PB-frames mode provides good compression performance levels, especially for low motion video sequences. However, since only bi-directional prediction is used for the B-picture of a PB-frame, when irregular motion is present in the video sequence, the quality of the B-picture decreases considerably. The Improved PB-frames mode of H.263+ addresses this problem by allowing forward only or backward only prediction, in addition to bi-directional prediction, of B-
macroblocks. While the prediction of B pictures in this mode is not as effective as that of true B-pictures, it does make the H.263+ IPB-frames mode more robust than H.263 PB-frames mode.

![Graph showing rate-distortion performance for FOREMAN.](chart)

**Figure 20. Improved PB frames mode: Rate-Distortion performance for FOREMAN.**

Our simulation results show that a significant improvement in PSNR is achieved as compared to the H.263 PB-frames mode when an active video sequence is coded as illustrated in Figure 20. The figure shows the rate-distortion performances levels achieved by the H.263 baseline, H.263 PB-frames mode and H.263+ Improved PB-frames mode for the active video sequence FOREMAN at 10 frame per second (fps). On the other hand, the PSNR gain over the H.263 PB-frames mode is smaller for video sequences that have moderate motion.

The H.263+ Improved PB-frames mode substantially increases the encoder/decoder complexity requirements. The complexity of the Improved PB-frames mode is slightly larger than that of the PB-frames mode due to the additional prediction modes. The computational load associated with the H.263+ Improved PB-frames mode is also usually larger than that of the H.263 PB-frames mode due to the forward prediction method of the Improved PB-frames mode.
Chapter 3: Performance of H.263+ Video Coding Standard

Like the H.263 PB-frames mode, the H.263+ Improved PB-frames mode requires more memory both at the encoder and the decoder because of the need to store two pictures in memory. There is also one frame delay associated with both of the above modes. This may present a problem in real-time applications.

3.2.4 Alternative Inter VLC mode (AIV)

This mode allows the intra macroblock quantized DCT coefficient VLCs of the Advanced Intra Coding mode to be used for some inter coded blocks. This mode of operation is useful at high bit rates, when short runs of zeros and large coefficient values are present, as the Advanced Intra Coding mode run-length VLCs are designed for such statistics. Best results for this mode are obtained when fine quantizers are used, as can be seen in Table 2. At very high bit rates, bit savings of as much as 10% can be achieved. The added complexity that this mode introduces is negligible, especially in software applications. In fact, less than 2% additional encoding/decoding time is usually required.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Quantizer Step Size</th>
<th>Y PSNR</th>
<th>Bits- (w/o mode)</th>
<th>Bits - Alternate Inter VLC mode</th>
<th>Bit savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKIYO</td>
<td>4</td>
<td>43.79</td>
<td>9354</td>
<td>8891</td>
<td>463 (5%)</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>39.47</td>
<td>4128</td>
<td>4073</td>
<td>55 (1%)</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>36.94</td>
<td>2411</td>
<td>2405</td>
<td>6 (0%)</td>
</tr>
<tr>
<td>FOREMAN</td>
<td>4</td>
<td>41.41</td>
<td>47659</td>
<td>44118</td>
<td>3541 (7%)</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>37.15</td>
<td>22036</td>
<td>21175</td>
<td>861 (4%)</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>34.73</td>
<td>13498</td>
<td>13201</td>
<td>297 (2%)</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>33.12</td>
<td>9434</td>
<td>9320</td>
<td>114 (1%)</td>
</tr>
</tbody>
</table>

Table 2. Average bit savings for inter coded pictures of the Alternate Inter VLC mode.
3.2.5 Modified Quantization mode (MQ)

To fully illustrate the capabilities of this mode, the TMN-8 [26] rate control method is used for the simulations in this section. Figure 21 (a) shows the chrominance PSNR performance for the video sequence FOREMAN with and without the Modified Quantization mode enabled. From this figure, it is clear that the chrominance PSNR increases substantially at low bit rates. Naturally, this causes a drop in luminance PSNR as less bits remain to represent the luminance coefficients. However, this drop is rather insignificant and the overall PSNR performance is usually improved. Figure 21 (b) shows that the overall PSNR performance is indeed higher when the Modified Quantization mode is enabled. The Modified Quantization mode adds very little computation time and complexity to the coder.

![Graphs showing Modified Quantization mode performance]
3.3 Computation Times and Compression Performance Improvements of Individual Modes

Figure 22 illustrates the added encoding computation times of the individual implemented H.263 and H.263+ optional modes. The results were obtained by encoding 300 frames of the video sequence FOREMAN at 64 kbps and 10 fps on a Pentium 200 MHz computer. The new TMN-8 rate control method was used for these simulations.

![Figure 22. Encoding CPU time for Different Modes](image)

**Figure 22.** Encoding times for the H.263 and H.263+ modes for FOREMAN at 64 kb/s on a 200 MHz PC.

Additional computational resources required by the H.263+ modes are negligible in our software decoder implementation, and real-time decoding of an H.263+ compliant bit stream can be supported. The encoder's speed of the H.263 baseline coder with any H.263 or H.263+ individual mode enabled is at most 15% larger than that of H.263. Setting the PB-frames mode on results in a reduction in encoding time, since only a restricted motion estimation operation is performed for the B-picture of a PB-frame. Encoding time is at most half of that of full search motion estimation when the fast search method is used (except for the H.263 PB-frames mode).
A summary of compression performance improvements resulting from the use of individual modes is given in Table 3. Results are presented for low and high bit rates using three QCIF video sequences at 10 fps: an active video sequence, FOREMAN, a sign language video sequence, SILENT, and a typical low motion videophone sequence, AKIYO. It can be observed that a given mode is not always suitable for any bit rate and/or any sequence. For example, the Alternate Inter VLC mode achieves compression gains only at high bit rates. The Deblocking
Chapter 3: Performance of H.263+ Video Coding Standard

Filter mode usually yields a decrease in PSNR, but the resulting picture subjective quality is generally much better. However, this mode may result in excessive blurriness at very low bit rates. Another observation is that the Modified Quantization mode does not lead to compression gains at high bit rates for low motion sequences, as the extended quantized coefficient range and the finer chrominance quantization are rarely used. Finally, the Unrestricted Motion Vector mode shows PSNR improvements for sequences with motion across picture boundaries (in FOREMAN for example), or at CIF and larger resolutions.

3.4 Compression Performance and Complexity of Mode Combinations

With the large number of possible mode combinations, it becomes difficult for implementers to select combinations that are suitable for their applications. The ITU-T video experts group decided to include non-normative mode combinations as guidelines for implementers [25]. The recommended mode combinations are based on the performance of individual modes. The performance criteria are the improvement in subjective quality, the impact on delay, and the additional complexity, computation, and memory demands.

The Level 1 preferred combination of modes includes Advanced Intra Coding, Deblocking Filter, Supplemental Enhancement Information, Full Frame Freeze only and Modified Quantization. The Level 2 preferred combination of modes includes, in addition to the Level 1 modes, the Unrestricted Motion Vector, Slice Structure, and Reference Picture Resampling modes. Finally, the Level 3 preferred combination of modes includes Level 2 and Level 1 preferred modes and Advanced Prediction, Improved PB-frames, Independent Segment Decoding and Alternate Inter VLC modes.
In our experiments, an error-free environment is assumed. Thus, the H.263+ modes involving error resilience, although part of the preferred mode combinations, are here excluded. Similarly, the Full Frame Freeze mode is not included in our simulations as it provides enhanced display capabilities but does not impact performance. Also, the Reference Picture Resampling mode is not here considered since it is currently not available in [45].

<table>
<thead>
<tr>
<th></th>
<th>Fast ME</th>
<th></th>
<th>Full ME</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P-pictures</td>
<td>B-pictures</td>
<td>time</td>
<td>P-pictures</td>
</tr>
<tr>
<td><strong>AKIYO, 8 kbps</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>33.9 dB</td>
<td>N/A</td>
<td>16.9 sec</td>
<td>33.9</td>
</tr>
<tr>
<td>AIC+DF+MQ (Level 1)</td>
<td>+0.74</td>
<td>N/A</td>
<td>+28%</td>
<td>+0.74</td>
</tr>
<tr>
<td><strong>FOREMAN, 128 kbps</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>35.83</td>
<td>N/A</td>
<td>25.3 sec</td>
<td>35.83</td>
</tr>
<tr>
<td>AIC+DF+MQ+UMV+AP+AIV</td>
<td>+1.05</td>
<td>N/A</td>
<td>+34%</td>
<td>+1.1</td>
</tr>
<tr>
<td>AIC+DF+MQ+UMV+AP+AIV +IPB (Level 3)</td>
<td>+1.52</td>
<td>-0.42</td>
<td>+15%</td>
<td>+1.53</td>
</tr>
</tbody>
</table>

Table 4. Mode combinations for FOREMAN and AKIYO.

Table 4 presents results for the Level 1 and Level 3 mode combinations for the video sequences AKIYO and FOREMAN, respectively. Based on our experiments, using a higher level of mode combinations provides better compression performance, especially for highly active video sequences such as FOREMAN, by as much as 1.5 dB at high bit rates. Moreover, note that the encoding time is, at most, approximately 25% greater, even for the Level 3 combinations of modes.
Chapter 4

Efficient Low Bit Rate Video Coding Algorithms

4.1 Introduction

In this chapter, we propose efficient platform independent coding algorithms and techniques that significantly improve the performance of the most computationally intensive components of video coding. The proposed algorithms include zero block prediction prior to DCT, partial IDCT computations, fast half pixel motion estimation, quantization with look-up tables and partial SAD computations.

There has been a significant effort towards developing efficient video coding algorithms in both industry and academia to improve the performance of video encoding and decoding. Most of these efforts have concentrated on developing efficient ways of motion vectors estimation and DCT/IDCT computation. However, it is possible to improve the already existing algorithms by exploiting specific statistical properties of coarsely quantized, slowly varying, low resolution sequences.

The main focus of this thesis is to implement a real-time H.263+ encoder in software. At this point it is important to define the parameters for real-time video encoding. In H.263+, besides custom picture frequencies, there are two predefined input video frequencies: 25 Hz and
30 Hz. However, it is currently not possible to have a software implementation on a single general purpose processor that will support encoding at these frequencies. Very low picture frequencies, such as a couple of frames per second, would not provide an acceptable quality in applications such as video telephony and video conferencing where video is multiplexed with speech. For such applications, an acceptable rate would be at least 10 frames per second for a QCIF (176 x 144) resolution. There are commercially available software implementations that can encode at this rate and resolution. Many of these implementations employ less computationally intensive encoding algorithms that have lower compression levels, such as motion JPEG or H.261. However, some applications, including wireless video telephony may require bit rates as low as 8 Kbits/sec, which can only be achieved, while maintaining acceptable reproduction quality, using more advanced video compression algorithms. In turn, these compression algorithms, such as those compliant with the H.263+, are computationally more expensive.

The block diagram of our H.263+ encoder software implementation is given in Figure 23. For simplicity, negotiable mode branches are not shown on the diagram. As illustrated in the figure, the integer pixel and half pixel accuracy motion vector searches and the interpolation are performed on the whole picture at a time, where the DCT, IDCT, Quantization, Dequantization and Variable Length Coding (VLC) are performed at the macroblock level. This way, we can start transmitting a part of the bit stream even though the encoding of the complete frame is not completed.
Chapter 4: Efficient Low Bit Rate Video Coding Algorithms

Figure 23. Block diagram of the H.263+ encoder implementation.
Chapter 4: Efficient Low Bit Rate Video Coding Algorithms

Table 5 shows the relative computational costs of some of the H.263+ encoder functions, which are obtained by profiling the software with the fast motion vector search algorithm enabled and without using rate control. These functions are also highlighted in Figure 23. As seen from the Table 5, any speed performance improvements of integer pixel motion estimation (ME), half pixel ME, DCT, IDCT and quantization will affect the overall encoding time significantly. Note that the integer pixel ME computations still take considerable time, although a fast search method is employed.

<table>
<thead>
<tr>
<th>Function</th>
<th>Computational Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference IDCT</td>
<td>25%</td>
</tr>
<tr>
<td>Integer Pixel ME</td>
<td>12%</td>
</tr>
<tr>
<td>Fast DCT</td>
<td>11%</td>
</tr>
<tr>
<td>Half Pixel ME</td>
<td>10%</td>
</tr>
<tr>
<td>Quantization</td>
<td>9%</td>
</tr>
</tbody>
</table>

Table 5. Relative computational costs of the most computationally intensive H.263+ encoder functions prior to optimization.

4.2 DCT

The DCT is one of the main building blocks of the H.263+ encoder and one of the most computationally expensive functions since it has to be computed for every block. Despite its small implementation complexity, the number of operations involved in calculating the DCT is still significant. For example, a 2-D DCT transform of an 8x8 block requires 2048 floating-point multiplications and 1792 additions. A number of fast DCT algorithms have been developed to
greatly reduce the number of operations [1], [6], [11], [27], [43]. Most of these algorithms are
variations of Lee’s 1984 pioneer work [28] or are based on variants of Winograd’s FFT.

To select a fast 2-D DCT algorithm suitable for our H.263+ encoder, we have considered
several properties of the algorithms. Since our implementation involves only software,
implementation complexity was not a primary issue of concern. However, we did require the
DCT algorithm to have a regular structure so that any irregular memory access patterns, which
may cause cache misses, are avoided.

Many of the fast DCT algorithms are aimed only at reducing the number of multiplications,
however, at the expense of increasing the number of additions. Today, most of the processors
include fast floating-point co-processor units and the cost of a multiplication operation is not
much more than the cost of an addition or a shift operation. Hence, the total number of operations
is more important than the number of multiplications for our software implementation.

Another consideration is the consecutive number of multiplications required by the fast DCT
algorithm. The DCT transform consists of many floating-point operations. If these operations are
converted to fixed-point operations some speed-up can be achieved. Since fixed-point operations
have limited accuracy, some error is accumulated after each operation. This error is much larger
for a multiplication operation. Thus, it is important to minimize the number of consecutive
multiplications in order to keep the accumulated error at a minimum.
Figure 24. Flowgraph for fast 1-D DCT from Arai, Agui, and Nakajima [1].
Solid circles represent additions and the arrows represent negating the data.
\[ a_1 = 0.707, \ a_2 = 0.541, \ a_3 = 0.707, \ a_4 = 1.307, \ a_5 = 0.383. \]

The selected fast DCT algorithm, which is based on the Fast Fourier Transform (FFT) and developed by Arai, Agui and Nakajima [1], satisfies most of the above requirements. Using this algorithm, a 2-D DCT of an 8x8 block requires 114 multiplications and 464 additions. However, what make it special are the properties it has for a scaled DCT implementation. In the case of forward DCT, 64 of these multiplications can be performed after the transform, and therefore, they can be combined with the quantization operation. Similarly, 64 of the IDCT multiplications can be done prior to the transform, which can be combined with the dequantization operation. Figure 24 shows the data flowgraph of an 8-point 1-D scaled fast DCT transform. The IDCT flowgraph can be obtained simply by reversing the direction of the flowgraph in Figure 24.
The scaled DCT algorithm uses floating-point arithmetic. The input data to the transform is 8-bit signed data and the output is 16-bit signed data. When converting this algorithm to fixed point arithmetic, the worst case scenario in precision loss should be considered. Even for intermediate variables 16-bit accuracy, which is very far from floating-point accuracy, is used. However, ITU-T does not have impose on DCT accuracy specification in the H.263+ coding standard. Since this standard is designed for low bit rate video coding where the quantizer has very high values, this precision loss in the DCT computations is tolerable and can be viewed as a loss in the quantization process.

Even though using fast algorithms greatly reduces the number of computations of the DCT and IDCT operations, the DCT still takes 15% and the IDCT takes 7% of the encoding time when the fast search algorithm is used for motion estimation. However, further speed performance gains can still be achieved by taking advantage of the fact that the DCT is used within a low bit rate video coding framework.

### 4.2.1 Zero Block Prediction Prior to DCT

Statistical properties of low bit rate video coding can be used to estimate the blocks that are not going to have any nonzero coefficients. A typical H.263+ video coder is based mostly on inter picture prediction, unlike in MPEG where intra pictures are inserted quite frequently between a number of inter pictures. Also, typical sequences used in H.263+ applications, such as video telephony or video conferencing sequences are generally low-activity video sequences. Therefore, the prediction error of each macroblock is small. After applying the DCT to the prediction error and performing quantization, many of the blocks will be zero blocks (i.e. all their
Chapter 4: Efficient Low Bit Rate Video Coding Algorithms

coefficient values are zero). If these blocks can be predicted prior to computing the DCT, then the corresponding DCT computations can be eliminated. In this case, the prediction operation should be repeated for each block and thus adds some overhead. The overall computation time, C, obtained after predicting of zero blocks can be calculated as

\[ C = X (1 - \alpha) + Y, \]

where X is the DCT computation time, Y is the computation time of prediction of zero blocks and \( \alpha \) is the percentage of blocks which are predicted to have all zero coefficients. \( \alpha \) can be also expressed by \( \beta \times \delta \), where \( \beta \) is the percentage of blocks that have all zero coefficients and \( \delta \) is the accuracy of prediction of zero blocks. The DCT computation time, X, is taken into account only for non zero blocks where the computation time of prediction for zero blocks, Y, always adds to the overall computation time since it has to be computed for every block.

4.2.1.1 Zero block Predictor

The computation time of prediction of zero blocks, Y, depends on the prediction operation. The following four predictor functions were considered to exploit the properties of the block before applying the DCT.

1. Variance of the block = \( \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} (\text{block}[i][j] - \text{average})^2 \); average = \( \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} \text{block}[i][j] \)

2. Sum of squares = \( \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} (\text{block}[i][j])^2 \)

3. Sum of absolute differences = \( \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} |\text{block}[i][j] - \text{average}| \)
4. Sum of absolute values = \( \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} \text{abs}(\text{block}[i][j]) \)

Figure 25. Zero block statistics for different predictors.

Figure 25 shows the statistical results for each predictor function. The simulations were performed using the FOREMAN video sequence and a quantizer value of 13. As can be seen from the figure, there does not appear a deterministic method to predict zero blocks. The \textit{Sum of squares} and \textit{sum of absolute values} predictor functions perform better than the other two functions in distinguishing zero blocks from the non-zero ones. Other simulations that were
performed using different sequences and quantizers also showed very similar characteristics.

Because of its associated low computation demands, the sum of absolute values is here selected as the predictor function.

<table>
<thead>
<tr>
<th>QP</th>
<th>α (high)</th>
<th>α (low)</th>
<th>β (high)</th>
<th>β (low)</th>
<th>δ (high)</th>
<th>δ (low)</th>
<th>threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;9</td>
<td><strong>Too low to consider</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.284</td>
<td>0.563</td>
<td>0.607</td>
<td>0.906</td>
<td>0.468</td>
<td>0.622</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0.268</td>
<td>0.527</td>
<td>0.642</td>
<td>0.918</td>
<td>0.417</td>
<td>0.574</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>0.258</td>
<td>0.483</td>
<td>0.665</td>
<td>0.922</td>
<td>0.387</td>
<td>0.523</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>0.412</td>
<td>0.657</td>
<td>0.691</td>
<td>0.931</td>
<td>0.596</td>
<td>0.705</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>0.396</td>
<td>0.625</td>
<td>0.708</td>
<td>0.938</td>
<td>0.559</td>
<td>0.667</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>0.380</td>
<td>0.601</td>
<td>0.731</td>
<td>0.944</td>
<td>0.520</td>
<td>0.637</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>0.495</td>
<td>0.708</td>
<td>0.744</td>
<td>0.948</td>
<td>0.665</td>
<td>0.747</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>0.479</td>
<td>0.696</td>
<td>0.761</td>
<td>0.955</td>
<td>0.630</td>
<td>0.729</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>0.467</td>
<td>0.682</td>
<td>0.772</td>
<td>0.957</td>
<td>0.605</td>
<td>0.713</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>0.454</td>
<td>0.682</td>
<td>0.788</td>
<td>0.961</td>
<td>0.576</td>
<td>0.709</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>0.447</td>
<td>0.667</td>
<td>0.795</td>
<td>0.963</td>
<td>0.562</td>
<td>0.692</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>0.529</td>
<td>0.741</td>
<td>0.808</td>
<td>0.966</td>
<td>0.655</td>
<td>0.766</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>0.524</td>
<td>0.731</td>
<td>0.816</td>
<td>0.968</td>
<td>0.643</td>
<td>0.755</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>0.583</td>
<td>0.785</td>
<td>0.827</td>
<td>0.971</td>
<td>0.706</td>
<td>0.809</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>0.575</td>
<td>0.772</td>
<td>0.833</td>
<td>0.972</td>
<td>0.691</td>
<td>0.795</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>0.575</td>
<td>0.769</td>
<td>0.843</td>
<td>0.974</td>
<td>0.682</td>
<td>0.789</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>0.562</td>
<td>0.761</td>
<td>0.847</td>
<td>0.975</td>
<td>0.664</td>
<td>0.781</td>
<td>6</td>
</tr>
<tr>
<td>26</td>
<td>0.617</td>
<td>0.805</td>
<td>0.854</td>
<td>0.977</td>
<td>0.722</td>
<td>0.824</td>
<td>7</td>
</tr>
<tr>
<td>27</td>
<td>0.612</td>
<td>0.787</td>
<td>0.859</td>
<td>0.978</td>
<td>0.712</td>
<td>0.805</td>
<td>7</td>
</tr>
<tr>
<td>28</td>
<td>0.608</td>
<td>0.783</td>
<td>0.866</td>
<td>0.979</td>
<td>0.701</td>
<td>0.800</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>0.609</td>
<td>0.783</td>
<td>0.870</td>
<td>0.980</td>
<td>0.699</td>
<td>0.799</td>
<td>7</td>
</tr>
<tr>
<td>30</td>
<td>0.652</td>
<td>0.814</td>
<td>0.877</td>
<td>0.981</td>
<td>0.744</td>
<td>0.830</td>
<td>8</td>
</tr>
<tr>
<td>31</td>
<td>0.647</td>
<td>0.812</td>
<td>0.879</td>
<td>0.981</td>
<td>0.736</td>
<td>0.828</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6. α, δ, β and thresholds for different quantizers.

high : high activity sequence, low : low activity sequence (more typical for H.263+).
4.2.1.2 Prediction Accuracy

The existence of zero blocks in a bit stream strongly depends on the characteristics of the original video sequence and the quantizer value. In high-activity sequences, the difference between two corresponding blocks that belong to two consecutive frames is quite significant. Also, coarse quantizers result in more zero blocks, while for very fine quantizers it may not worth to try to predict zero blocks.

Table 6 lists the different values for the experimentally best $\alpha$, $\beta$ and $\delta$ parameters that are defined in Equation (7) for different quantizers. If the sum of absolute values is greater than the threshold, then the block is predicted to be non-zero. The threshold is determined statistically based on the characteristics of several video sequences so that it does not cause any quality degradation in the reconstructed picture. The threshold is found to vary only slightly as a function of the video content. As can be seen from the Table 6, zero block prediction becomes much more advantageous at low bit rates, i.e. when a coarse quantizer is employed. It is also possible to increase the parameter $\alpha$ by selecting higher thresholds. However, in that case, the reproduction quality may decrease.

4.2.1.3 Computation Time Gains

Since zero block prediction has to be done for every DCT calculation, the computation gain (in percentage) is given by

$$C_g = \frac{X - ((1-\alpha)*X + Y)}{X} \times 100 = \frac{\alpha*X - Y}{X} \times 100,$$

(8)
where $X$ is the DCT computation time, $Y$ is the computation time of prediction of zero blocks and $\alpha$ is the percentage of blocks which are predicted to have all zero coefficients. Even if a block is predicted to be zero, it may still be necessary to perform some additional operations, e.g. to fill in the coefficients array with zeros. In that case, Equation (7) becomes

$$C = X(1-\alpha) + Y + \alpha Z,$$

(9)

where $Z$ is the computation time require to process a zero block after its detected, and Equation (8) becomes

$$C_e = \frac{\alpha(X-Z)-Y}{X} \times 100$$

(10)

The values of $X$, $Y$ and $Z$ strongly depend on the implementation of each function. Also, the relative computation cost of individual operations, such as addition and multiplication, is processor dependent. For instance, if an Intel MMX processor is used, then with proper pipelining the cost of a multiplication can be the same as the cost of an addition, which is one clock cycle. On the other hand, other processors may have a multiplication cost that is higher than an addition cost. Therefore, it is not practical to compute the number of additions and multiplications necessary for an operation and place these in Equation (10) to determine precise value of $C_e$. Rather, it would be more meaningful to use the processor time needed to perform each operation as a parameter during the estimation process.

4.2.1.4 Simulation Results

In order to evaluate the computation advantage that zero block prediction provides, the computation times of individual functions and implementations should be known. The
computation times of different implementations of DCT functions running on a Pentium MMX 200 MHz computer are given in Table 7.

<table>
<thead>
<tr>
<th>Function</th>
<th>X: Time (for 100 frames: 59400 call)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast DCT (H.263 encoder)</td>
<td>2000 ms.</td>
</tr>
<tr>
<td>Fast DCT (Agui, et al.)</td>
<td>1900 ms.</td>
</tr>
<tr>
<td>Integer DCT (Agui et al.)</td>
<td>1150 ms.</td>
</tr>
<tr>
<td>MMX DCT (MMX assembly version of Integer DCT)</td>
<td>400 ms. (+80 ms. preprocessing ignored)</td>
</tr>
</tbody>
</table>

**Table 7. Computation times of DCT functions.**

Table 8 shows the computation costs of calculating the sum of absolute values of an 8x8 block.

<table>
<thead>
<tr>
<th>Function</th>
<th>Y: Time (for 100 frames: 59400 calls)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of absolute values (C imp.)</td>
<td>200 ms.</td>
</tr>
<tr>
<td>Sum of absolute values (MMX imp.)</td>
<td>110 ms.</td>
</tr>
</tbody>
</table>

**Table 8. Computation times of different Sum of absolute values functions.**

Also, the implementation dependent computation cost of setting an 8x8 array of integer coefficients to zero values is given in Table 9.

<table>
<thead>
<tr>
<th>Function</th>
<th>Z: Time (for 100 frames: 59400 call)</th>
</tr>
</thead>
<tbody>
<tr>
<td>memset (C imp.)</td>
<td>150 ms.</td>
</tr>
</tbody>
</table>

**Table 9. Computation time of memory set functions.**

Using Equation (9) the computation time for the MMX DCT implementation, with a quantizer of 20 and for a low activity video sequence is calculated as
Chapter 4: Efficient Low Bit Rate Video Coding Algorithms

\[ C = 400 \times (1 - 0.741) + 110 + 0.741 \times 150 = 103 + 110 + 111 = 324 \text{ ms.} \]

When the same conditions are simulated, a total computation time for the DCT of 340 ms. is obtained. If we take into consideration that MSVC++* profiling is not extremely accurate and some operations would change the cache content and thus affect the cache hit ratio, the estimated computation time appears to be quite accurate.

Using above zero block prediction method, a computation gain of 15% is achieved. This speed improvement is not very significant. This is due to the fact that the computation time of the sum of absolute values (Y) and the memset function (Z) are relatively large as compared to the computation time of the DCT (X) on an MMX processor. On the other hand, this is not the case for implementation of the DCT on a Pentium processor that does not support MMX. Therefore, greater speed improvements, in such a case, can be expected. In fact, using our “C” DCT implementation, with a quantizer of 20, and for low activity video sequences, the computation time becomes

\[ C = 1900 \times (1 - 0.741) + 200 + 0.741 \times 150 = 492 + 110 + 111 = 713 \text{ ms.} \]

The actual computation time for the function is 751 ms, resulting in a computation gain of 60%.

4.2.1.5 Implementation Complexity and Memory Requirements

The above method adds very little complexity and memory requirements to the software implementations of the DCT. However, in hardware implementations of the DCT units, the

* Microsoft Visual C++
residual data transfer to the DCT component is one of the main time consuming parts of the whole process. Since the described method does not reduce the amount of data transfer, hardware implementations may not benefit from this as much as software implementations.

### 4.3 IDCT

In video coding, the IDCT is performed both at the encoder and the decoder. While the DCT has to be computed for each block, the IDCT needs to be computed only for the non-zero blocks, which are indicated by the Coded Block Pattern (CBP) in the H.263+ bit stream. The CPB shows which 8x8 blocks are encoded in a macroblock structure. In a typical H.263+ low bit rate bit stream, approximately 70-80% of the prediction error blocks are not coded, i.e. they are zero blocks. Moreover, even the non-zero blocks have dominating zero sub-blocks. This makes it possible to further reduce the computation time. Figure 26 shows the number of computations necessary for a fast IDCT, if zero sub-blocks exist in the block.

<table>
<thead>
<tr>
<th>Non-Zero</th>
<th>Non-Zero</th>
<th>144 Multiplications and 464 Additions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Zero</td>
<td>Non-Zero</td>
<td>(a)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Zero</th>
<th>Non-Zero</th>
<th>92 Multiplications and 284 Additions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Zero</td>
<td>Zero</td>
<td>(b)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Zero</th>
<th>Zero</th>
<th>56 Multiplications and 252 Additions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>Zero</td>
<td>(c)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Zero</th>
<th>Non-Zero</th>
<th>128 Multiplications and 432 Additions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Zero</td>
<td>Zero</td>
<td>(d)</td>
</tr>
</tbody>
</table>

**Figure 26. Number of computations of a fast IDCT for different sub zero block patterns.**
Different fast DCT algorithms may produce a greater or smaller reduction in the number of computations. The computation saving for the block structure shown in Figure 26.c is approximately 50%. This structure is also the most typical in a low bit rate video coding application where approximately 80% or so of the blocks conform such a structure, of course depending on the input sequence and the quantizer. Hence, the other structures shown in Figure 26 can be ignored, and therefore, the savings in the computation time is

$$C_g = 80\% \times 0.5 = 40\%.$$  

There is also an overhead involved in finding which of the sub-blocks are zero. However, this overhead can be eliminated by finding the required information during variable length coding of coefficients at the encoder side and variable length decoding of the bit stream at the decoder side. After the residual blocks are DCT coded and quantized they are scanned in zigzag order. Variable length coding is then performed. After the last non-zero coefficient is VLC coded, a bit sequence indicates that the rest of the coefficients are zero valued. The position of the last non-zero coefficient makes transparent the zero sub block structure. However, since the coefficients are zigzag scanned, it is not possible to exactly identify the structure in Figure 26.c, instead the structure in Figure 27 can be identified. Approximately 70% of the blocks are in this structure in a low bit rate bit stream.

50 multiplications and 240 additions.

Figure 27. Zero Sub block detection via bit stream parsing.
4.3.1 Implementation Complexity and Memory Requirements

The implementation of zero sub-block detection is a fairly simple task for both hardware and software video coders. Hardware applications may benefit especially from the fact that the above method reduces the amount of data transfer to the IDCT unit. However, two different IDCT implementations may be necessary in hardware, one for non-zero blocks and the other for partially zero blocks. On the other hand, a software implementation of the IDCT module that will support both types of blocks is trivial.

4.4 Quantization and Dequantization

The quantization operation takes 8% and the dequantization operation takes 2% of the whole encoding time if the fast search algorithm is during for the motion estimation process. While dequantization is performed only for the non-zero blocks, quantization has to be performed for every block.

It is mentioned in Section 4.2 that the pre-scaled fast DCT algorithm and the post-scaled fast IDCT algorithm may remove the need to perform quantization and dequantization if the quantizer is embedded into the pre-scale and post-scale coefficients. While this is true for most video coding algorithms, because of the dead-zone quantization of H.263, it is not applicable to our encoder implementation unless the Advanced Intra Coding mode is turned on.

The zero block prediction method that is described in Section 4.2.1 can also be applied to the quantization process. The computation gain of both the DCT and the quantization can be calculated by including the quantization as follows:
\[ C_g = \frac{X + Q - ((1 - \alpha) \cdot (X + Q) + Y + \alpha Z)}{X + Q} \times 100 = \frac{\alpha \cdot (X + Q - Z) - Y}{X + Q} \times 100, \quad (11) \]

where \( Q \) is the computation time of quantization.

### 4.4.1 Quantization with Tables

Another method for faster quantization is to use look-up tables. After the residual block is DCT coded, the coefficients block is clipped into the range \([-2048, 2047]\). It is possible to create a 2D array by pre-calculating all quantization results for each of the 31 quantizers and to access this array by using the coefficient value and the quantizer as indices. Such an array would take 253,952 bytes of memory if 16-bit short integer registers are used.

If look-up tables are used, 64 irregular memory read operations are performed per 8x8 block. Since these memory accesses are irregular and thus cause cache-misses, this method is only three times faster than doing quantization using the CPU. If look-up tables are used in conjunction with the previously discussed zero block prediction method, then the advantage of using look-up tables becomes insignificant during the encoding process.

### 4.4.2 Implementation Complexity and Memory Requirements

The implementation of look-up tables and the memory needed for this is trivial in software applications. In hardware, however, using 250 Kbytes of memory for such look-up tables may not be suitable. But, if the memory is not a problem, the implementation of lookup tables becomes practical for hardware applications too, as it removes the need for a multiplication unit.
4.5 Partial SAD Computation Technique

If the full search algorithm is used, the motion estimation procedure takes approximately 75% of the encoding time. A number of fast motion vector search algorithms were developed, however, the full search algorithm is still the most commonly used one because of unavoidable quality degradation caused by the use of fast search algorithms.

A common computation reduction technique is to compare the accumulated SAD values to the minimum SAD after each row of the block and then terminate the computation if the accumulated SAD is greater than the minimum SAD. Statistics show that more than half of all SAD computations are terminated before calculating the 5th line in a 16x16 block. Using this technique, approximately 70 SAD computations are performed instead of the 256 SAD calculations normally needed for a 16x16 block.

The number of SAD computations can be further reduced by trying to predict whether the SAD will exceed the minimum SAD before it actually exceeds it. For example, during the SAD computation of a 16x16 block, after the computation is completed for the first row, if the accumulated SAD so far is more than the half of the minimum SAD, it can be predicted that the final SAD will exceed the minimum SAD. This prediction is clearly a function of the SAD value computed so far, the number of rows processed and the total number of rows. Considering these, a fairly effective prediction model can be given by

\[
\text{Predicted}_\text{SAD} = \text{SAD} + 1 \times \frac{\text{SAD}}{N-1} \times \phi,
\]

(12)
where \( I \) is the number of rows that are not processed so far, \( N \) is the dimension of the ME block and \( \phi \) is the accuracy coefficient. The number of SAD calculations per search and the PSNR decrease in the picture for different \( \phi \) values are presented in Table 10 for the FOREMAN and the AKIYO sequences at different bit rates based on 16x16 block motion estimation.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>w/o SAD prediction</th>
<th>( \phi = 0.2 )</th>
<th>( \phi = 0.5 )</th>
<th>( \phi = 0.7 )</th>
<th>( \phi = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREMAN (48 kbit/sec)</td>
<td>PSNR (dB)</td>
<td>30.74</td>
<td>-0.04</td>
<td>-0.07</td>
<td>-0.3</td>
</tr>
<tr>
<td></td>
<td>Number of SADs</td>
<td>89</td>
<td>61</td>
<td>40</td>
<td>33</td>
</tr>
<tr>
<td>AKIYO (24 kbit/sec)</td>
<td>PSNR (dB)</td>
<td>37.17</td>
<td>+0.03</td>
<td>0.0</td>
<td>-0.06</td>
</tr>
<tr>
<td></td>
<td>Number of SADs</td>
<td>54</td>
<td>33</td>
<td>24</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 10. Change in picture quality for different \( \phi \).

The SAD prediction is not as effective when it is used in conjunction with the fast search ME algorithms. The reason is that in many fast search algorithms, the minimum SAD and the calculated SAD are very close to each other, delaying the termination of the SAD computation until the late stages of the process. Thus, trying to predict the SAD would usually not be as successful and would not save as much computation time when fast search motion estimation is employed.

### 4.6 Fast Half Pixel Motion Estimation Based on Approximation

Half pixel motion estimation is one of the main enhancements of H.263 over H.261. It provides, in most cases, more than 2 dB PSNR increase in picture quality. The eight SAD calculations that are necessary for a half pixel ME form a rather insignificant computational load when compared to the over 256 SAD calculations needed for a full search ME. Yet, there have been efforts to
reduce or eliminate the computation time for the half pixel ME. The resulting computation reduction techniques have become especially beneficial when fast search algorithms are used.

4.6.1 A Simplified ME for MPEG-2 Encoder

\[
\begin{array}{ccc}
A & B & O \\
O & O & O \\
& a & c \\
& \times & \times \\
C & b & D \\
& \times & O & \times \\
& \times & \times \\
& O & O & O \\
\end{array}
\]

O Integer Pixel

\begin{align}
SAD_a &= \psi_{VH} \times (SAD_A + SAD_B + SAD_C + SAD_D + 2) / 4, \\
SAD_b &= \psi_H \times (SAD_C + SAD_D + 1) / 2, \text{ and} \\
SAD_c &= \psi_V \times (SAD_B + SAD_D + 1) / 2,
\end{align}

where \(\psi_{VH}, \psi_H\) and \(\psi_V\) are statistically optimized coefficients for high bit rate (>1 Mbit/sec) MPEG2 encoding. It is possible to adapt this technique to low bit rate video coding (<64 Kbit/sec) by re-optimizing these coefficients, as shown in Table 11.
Table 11. The change in PSNR using different approximation factors ($\psi$).

<table>
<thead>
<tr>
<th></th>
<th>Foreman 48 Kb/sec (Q&gt;13)</th>
<th>Akiyo 8 Kb/sec (Q&gt;13)</th>
<th>Foreman 64 Kb/sec (Q&lt;13)</th>
<th>Akiyo 28 Kb/sec (Q&lt;13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR with half pixel ME</td>
<td>30.47</td>
<td>32.63</td>
<td>31.56</td>
<td>37.92</td>
</tr>
<tr>
<td>PSNR with $\psi_{vh}=13/16$, $\psi_v=\psi_H=14/16$</td>
<td>-0.15</td>
<td>-0.61</td>
<td>-0.15</td>
<td>-0.34</td>
</tr>
<tr>
<td>PSNR with $\psi_{vh}=27/32$, $\psi_v=\psi_H=29/32$</td>
<td>-0.13</td>
<td>-0.21</td>
<td>-0.21</td>
<td>-0.47</td>
</tr>
</tbody>
</table>

4.6.2 Fast Search ME and Half Pixel ME Approximation

![Figure 29. Half pixel motion vector prediction using 5 precomputed integer pixel SADs](image)

To perform half pixel ME approximation, all the surrounding integer pixel SADs must be known. This is not a problem in the case of full search ME. However, in many fast search ME algorithms, not all the surrounding SADs are computed. One solution is to simply perform ME for missing locations. Another solution is to modify the approximation equations and the approximation coefficients, $\psi$, so that only the available integer SADs are used.
The H.263+ Test Model Near-term (TMN) fast integer pixel ME algorithm computes the SADs at locations that are the vertices of a diamond as illustrated in Figure 29. The modified half pixel ME equations and the optimized $\psi$'s for this algorithm are given by

\begin{align}
SAD_a &= \psi_{VH} \times (SAD_A + SAD_B + SAD_C + 1) / 3 \quad \psi_{VH} = 28/32, \\
SAD_b &= \psi_H \times (SAD_A + SAD_C + 1) / 2 \quad \psi_H = 29/32, \text{ and} \\
SAD_c &= \psi_V \times (SAD_B + SAD_C + 1) / 2 \quad \psi_V = 29/32.
\end{align}

Although this technique removes the need for 8 half pixel SAD computations, the reproduction quality may not be acceptable in some applications. Thus, we next suggest a new approach that yields better computation-performance tradeoffs. In this approach, three fast half pixel ME methods are considered. Our motivation is essentially the same for all three methods. If a half pixel block is estimated to have the minimum SAD by using the surrounding integer pixel blocks, it is most likely that it or a neighbor block has the minimum SAD. A limited search can then be performed to find the block correspond to the smallest SAD. Even if the described procedure fails to find the minimum SAD, a very close match is likely to be found. The three methods are described next:

1. **Method 1**: We find the smallest SAD out of the four surrounding integer pixel SADs. Then we perform three half pixel MEs accordingly. For example, in Figure 30, if B corresponds to the smallest SAD, then we compute the half pixel SADs for pixels 1, 2 and 3. If C corresponds to the smallest SAD, then we compute half pixel SADs for pixels 3, 5 and 8, and so on. We also compare these to the center pixel SAD and pick the smallest of two.
Chapter 4: Efficient Low Bit Rate Video Coding Algorithms

Figure 30. Half and integer pixel locations in a diamond shape area.

2. Method 2: We find the two smallest SADs out of the four surrounding integer pixel SADs. We then perform two or three half pixel ME SADs depending on their location. For instance, in Figure 30, if the SADs for pixels A and B are the smallest, we find the half pixel SADs for the pixels 1, 2 and 4. If SADs for B and D are the smallest then we find the half pixel SADs for the pixels 2 and 7. We then pick the smallest of these SADs and the center pixel SAD.

3. Method 3: We compute the half pixel SADs using the four surrounding integer pixels and the center pixel by approximation. We then perform a half pixel ME to find more accurate SADs for the N pixels that correspond to the smallest approximated SADs out of 8 pixels. Last, we find the smallest of the N SADs and the center SAD to determine the half pixel MV.

The change in PSNR associated with each of the above methods is given in Table 12. The method 3, by selecting N equal to 4, thus performing 4 ME operations per macroblock, gives the best results. The method 1, which uses 3 ME operations, and the method 2, which performs 2-3
ME operations, cause very small quality degradation. The only method that doesn’t need to perform any ME is the approximation method that uses 5 integer pixels. When compared to the PSNR decrease caused by not using half pixel ME at all, the 5 pixel approximation method gives very good results and can be used in applications where processing power is very limited.

<table>
<thead>
<tr>
<th></th>
<th>Foreman 64 Kb/sec</th>
<th>Foreman 48 Kb/sec</th>
<th>Akiyo 28 Kb/sec</th>
<th>Akiyo 8 Kb/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>with half-pixel ME</td>
<td>31.56</td>
<td>30.47</td>
<td>37.92</td>
<td>32.63</td>
</tr>
<tr>
<td>no half-pixel ME</td>
<td>-2.24</td>
<td>-2.16</td>
<td>-1.61</td>
<td>-1.33</td>
</tr>
<tr>
<td>Approximation using 9 integer pixels $\psi_{YH}=27/32$, $\psi_Y=\psi_{H}=29/32$</td>
<td>-0.21</td>
<td>-0.13</td>
<td>-0.47</td>
<td>-0.21</td>
</tr>
<tr>
<td>Approximation using 5 integer pixels $\psi_{YH}=28/32$, $\psi_Y=\psi_{H}=29/32$</td>
<td>-0.36</td>
<td>-0.31</td>
<td>-0.52</td>
<td>-0.21</td>
</tr>
<tr>
<td>Method 1</td>
<td>-0.07</td>
<td>-0.05</td>
<td>-0.04</td>
<td>0</td>
</tr>
<tr>
<td>Method 2</td>
<td>-0.12</td>
<td>-0.11</td>
<td>0</td>
<td>-0.02</td>
</tr>
<tr>
<td>Method 3 (N=4 pixels)</td>
<td>-0.04</td>
<td>-0.02</td>
<td>+0.02</td>
<td>+0.06</td>
</tr>
<tr>
<td>Method 3 (N=2 pixels)</td>
<td>-0.15</td>
<td>-0.14</td>
<td>-0.05</td>
<td>-0.02</td>
</tr>
</tbody>
</table>

Table 12. PSNR table for different fast half pixel ME methods.

![Figure 31. PSNR - number of iterations tradeoff in method 3.](image-url)
Figure 31 shows the change in PSNR when different N values are used for different bit rates of the Foreman sequence. N defines the number of times the half pixel ME search is iterated. Selecting an N value of 3 seems to yield good compromise between computation time and picture quality.

4.6.3 Implementation Complexity and Memory Requirements

Fast Half Pixel ME not only reduces the number of SAD computations but also lowers the number of irregular memory accesses. Therefore it increases the performance of both hardware and software systems. There is some overhead resulting from the computation of half pixel SAD approximations and finding the minimum one, but it is rather small. While hardware implementation of the proposed methods may be nontrivial because of the need to have a multiplier unit, the complexity of the associated software implementations is negligible.
Chapter 5

MMX Implementation of the H.263+ Video Encoder

5.1 Introduction

In this chapter, we propose platform (and more specifically MMX) dependent mapping techniques for the computationally intensive SIMD structured components of video coding to improve the video coding speed performance. These video coding components include the DCT, SAD computations, interpolation, data interleaving for half pixel motion estimation and motion compensation functions. The mapping of these components to MMX provide an additional speed advantage over the platform independent optimizations proposed in Chapter 4 and make real time H.263+ video encoding possible on the Pentium MMX processor [20], [36], [37].

5.2 Video Coding Cores with SIMD Structure

Video coding systems, like many other multimedia systems, has very computationally intensive core functions that can be implemented efficiently using a Single Instruction Multiple Data (SIMD) structure. Table 13 shows a profile of the H.263+ video encoder for the FOREMAN sequence without any modes enabled, and with the fast ME search, fast IDCT, fast DCT and the fast Quantization enabled and rate control disabled. The relative computation time percentages may vary slightly as a function of the input video sequence and the quantizer.
Table 13. Relative computational costs of the most compute intensive H.263+ encoder functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Computational Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast ME</td>
<td>17%</td>
</tr>
<tr>
<td>DCT</td>
<td>15%</td>
</tr>
<tr>
<td>Half Pixel ME</td>
<td>12%</td>
</tr>
<tr>
<td>Interpolation</td>
<td>7%</td>
</tr>
<tr>
<td>IDCT</td>
<td>5%</td>
</tr>
<tr>
<td>Load ME Area</td>
<td>4%</td>
</tr>
<tr>
<td>Motion Compensation</td>
<td>3.5%</td>
</tr>
<tr>
<td>Quantization</td>
<td>3%</td>
</tr>
</tbody>
</table>

Except for the quantization function, which is implemented using a look-up table, all of the functions in Table 13 employ SIMD structured algorithms. Thus, a significant performance increase can be expected if the MMX architecture is properly exploited.

5.3 SAD Computations for Motion Estimation

The motion vectors are found through performing a number of SAD operations. In H.263+, the SAD is performed on either 16x16, in baseline coding, or on 8x8 unsigned byte blocks, when Advanced Prediction or Deblocking Filter modes enabled. Since the size of each data element is 8 bits, 8 data elements can be processed at the same time on MMX, giving a significant performance improvement.
Subtraction of two unsigned bytes produces a 9-bit signed number. Therefore, to compute the absolute value of the difference it may seem that a conversion to 16-bit format is needed. However, this conversion may not be necessary if the saturated arithmetic feature of the MMX architecture is used. Consider the two MMX registers in Figure 32. One of the registers contains 8 bytes of data from the target block and the other register contains 8 bytes of data from the current block. When register $B$ is subtracted from register $A$ with unsigned saturation, the result is set to the difference between $a_i$ and $b_i$ if $a_i$ is greater than $b_i$ or zero if $a_i$ is smaller than or equal to $b_i$, simply because the negative result saturates to zero. When the situation is reversed, i.e. $A$ is subtracted from $B$, the result saturates to zero when $b_i$ is smaller than $a_i$. If the OR operation is applied to these two subtraction results, the outcome will be the absolute value of the difference between $A$ and $B$ registers. Then the data can be unpacked and accumulated in 16-bit precision registers. Computing one SAD for two 16x16 blocks would require that 64 memory load, 64 packed subtraction, 32 packed-OR, 32 unpack and 63 addition operations be performed. The basic core of this algorithm is published by Intel [20]. We adopted their technique to work with 16x16 and 8x8 blocks.

\[
\begin{align*}
\text{a8} & | \text{a7} & | \text{a6} & | \text{a5} & | \text{a4} & | \text{a3} & | \text{a2} & | \text{a1} \\
\text{b8} & | \text{b7} & | \text{b6} & | \text{b5} & | \text{b4} & | \text{b3} & | \text{b2} & | \text{b1} \\
- \text{(psubusb)} & & & & & & & \\
\text{b8} & | \text{b7} & | \text{b6} & | \text{b5} & | \text{b4} & | \text{b3} & | \text{b2} & | \text{b1} \\
= & & & & & & & \\
x8 & | x7 & | x6 & | 0 & | x4 & | x3 & | x2 & | 0 \\
\text{a8} & | \text{a7} & | \text{a6} & | \text{a5} & | \text{a4} & | \text{a3} & | \text{a2} & | \text{a1} \\
= & & & & & & & \\
0 & | 0 & | 0 & | y5 & | 0 & | 0 & | y1 \\
\text{por} & & & & & & & \\
x8 & | x7 & | x6 & y5 & | x4 & | x3 & | x2 & | y1
\end{align*}
\]

Figure 32. Data flow to compute absolute value of differences of two unsigned data.
One disadvantage of using this technique is that we cannot use any kind of partial SAD computation technique. Since 4 of the additions are saved in the same register, after computing each line, i.e. 16 bytes, unpacking these values and adding them together to find the SAD at the end of each row would cause an unacceptable overhead. Moreover, if fast search is used, not using the partial SAD computation technique would not slow ME since most of the time the SAD computation is not terminated until the later stages. The MMX implementation of the SAD function runs approximately 4-5 times faster than the optimized C implementation. For example, when encoding the FOREMAN sequence with a constant quantizer 13 on a Pentium MMX 200 MHz computer, 221901 SAD computations were performed in 2349 ms. in the optimized C version of the encoder and the same number of operations were performed only in 522 ms. in the MMX version of the encoder. This implementation doesn't have any effect on the rate-distortion performance of the encoder.

5.4 DCT

Computing the DCT takes 15% of the encoding time, and, unlike SAD, its high computational cost is caused mainly by its complexity. We implemented the fast floating point DCT algorithm from Arai et al. [1] to use MMX instructions because of its scalability properties, small number of operations it requires and its suitable structure to implement in SIMD structure. Since MMX instructions can perform only integer arithmetic, the floating point algorithm was converted to fixed-point algorithm. The input to the DCT is an array of signed 8-bit data and the output is an array of 16-bit signed data. In order to keep the precision as high as possible, in the intermediate
stages of the computations, some downscaling and upscaling operations are applied to the data. These operations are given by

\[
\text{Upscale}(X, \text{precision}): X << \text{precision}, \quad (19) \\
\text{Downscale}(X, \text{precision}): (X + 2^{\text{precision}-1}) >> \text{precision}, \quad (20)
\]

where "\(<<\)" denotes the left shift operation and "\(>>\)" denotes the right shift operation.

In the intermediate stages of the computations, it is possible to use either 16-bit or 32-bit registers. Using 32-bit registers would increase the accuracy of the DCT but then only 2 intermediate data elements will fit into one MMX register, causing performance loss in terms of speed. On the other hand, by using 16-bit intermediate data it is possible to process 4 data elements at a time. Since H.263+ is a low bit rate video coding standard and DCT coefficients are thus usually quantized very coarsely, DCT errors are mostly insignificant. Therefore, 16-bit intermediate registers are here used.

It is possible to implement the DCT function using MMX instruction in three ways:

1. Computing the DCT for a 1x8 vector one at a time: Here, the DCT is first performed on the first row, then the second row, and so on. This is also how our C implementation works.

2. Performing the DCT on four 1x8 vectors at a time. In this method, the DCT is first applied to the first four rows, and then the other four rows are processed. The reasoning behind processing four rows at a time, instead of 2 or 3 for example, is that the four 16-bit data elements fits into one 64-bit MMX register.
Chapter 5: MMX Implementation of the H.263+ Encoder

3. Performing the DCT on four 8x8 blocks simultaneously. Here, each block is processed as described in the first method but the computations for four blocks are carried out at the same time.

The disadvantages of the first method are the implementation complexity and the different extra precision lost due to fitting many data elements in the same MMX register. That may cause extra precision loss. Implementing the third method appears to be simple, but the four 8x8 matrix data has to be interleaved before applying the DCT and this may cause a major overhead. In order to implement the second method, the input array should be transposed before applying the DCT. Because of its simplicity, the second method is selected in our implementation.

To perform the DCT of four 8x1-vectors, one 16-bit data word from each row is read into the same MMX register. In order to obtain the highest efficiency, 16-bit words should be placed next to each other in memory making transposition of the input matrix necessary. After transposition, the DCT for the upper four rows of the array is computed, followed by the computations for the remaining four rows. After that, the array is transposed. Then, the DCT is applied again four rows at a time.

Operations other than multiplications are performed using four 16-bit data. The multiplications are performed using 32-bit data, and the corresponding results are immediately downscaled and compressed into 16 bits. As illustrated in Figure 33, first, two multiplication operations, one for the low significant part and the other for the high significant part are performed. Then, the results were interleaved so that the higher significant part and the lower significant part of the same multiplication will be in the same MMX register. Then, 32-bit
additions and arithmetic right shift operations are executed for downscaling. Last, the four 32-bit data is packed into one 64-bit MMX register with signed saturation.

\[
Z = \text{Downscale} \ (Y \times A, P)
\]

**Y**: An intermediate variable  
**A**: Coefficient to multiply  
**P**: Downscaling precision

\[
\begin{align*}
Y_3 & \quad Y_2 & \quad Y_1 & \quad Y_0 \\
A & \quad A & \quad A & \quad A
\end{align*}
\]

![Diagram](image)

Figure 33. Flowgraph of multiplication and downscaling operations performed with MMX instructions.

The MMX DCT implementation runs approximately four times faster than the floating point implementation and three times faster than the integer optimized C implementation. In our encoder implementation in C, the residual block is stored in a 32-bit integer array. However, the
Chapter 5: MMX Implementation of the H.263+ Encoder

MMX DCT implementation takes a 16-bit array as input. Therefore, the 32-bit integer array needs to be converted to a 16-bit integer array adding a significant overhead. Hence, the overall speed up drops to three times faster when compared to the optimized C floating point algorithm.

Since an integer DCT algorithm is used in the MMX implementation, the rate-distortion performance of the encoder is affected as well. Table 14 shows the resulting picture quality in terms of PSNR for different bit rates and sequences of both DCT implementations. As seen from the table, using an integer DCT implementation does not cause a significant decrease in PSNR.

<table>
<thead>
<tr>
<th></th>
<th>Foreman 48 Kb/sec</th>
<th>Akiyo 8 Kb/sec</th>
<th>Foreman 64 Kb/sec</th>
<th>Akiyo 28 Kb/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR with floating point DCT</td>
<td>30.47</td>
<td>32.63</td>
<td>31.56</td>
<td>37.92</td>
</tr>
<tr>
<td>Variation in PSNR with integer or MMX DCT</td>
<td>-0.01</td>
<td>+0.02</td>
<td>-0.01</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 14. Performance degradation caused by using integer DCT implementation.

5.5 IDCT

The H.263+ standard is based on inter picture prediction. Thus, any errors caused due to IDCT’s low accuracy propagate throughout the video coding process. ITU-T already has an IDCT accuracy measure procedure defined within the H.263+ standard. Unless very high fixed point accuracy is used, it is very difficult, if not impossible, to implement a standard compliant IDCT function MMX-wise using 32-bit or less precision. Moreover, when the IDCT is implemented with 32-bit precision, and when the computational overhead is considered, a large speed up is not expected. Intel has already implemented and made publicly available an MMX IDCT routine for
MPEG decoding. Their implementation employs 32-bit precision for multiplication and 16-bit precision for accumulation operations, and is approximately 4 times faster than the optimized C implementation. However it does not meet the specifications of the ITU-T standard. The ITU-T IDCT accuracy specifications require that, the DCT and the IDCT, with 64-bit floating point accuracy, applied to a certain number of blocks that are generated by a pseudo random number generator routine, and then the peak error, mean square and mean errors (pixel-wise and overall) should be less than predefined thresholds. Also, if the reference IDCT produces a zero output, the IDCT under test should also produce a zero output. More details on these requirements can be found in [25]. Even if an implementation does not meet the standard, it may be beneficial to compare its computation accuracy to such thresholds. Table 15 shows the partial* accuracy results for 32-bit floating point, 64-bit integer and 32-bit integer IDCT implementations along with Intel’s MMX IDCT implementation.

<table>
<thead>
<tr>
<th>IDCT Implementation</th>
<th>Peak error</th>
<th>Zero output rule violation</th>
<th>Maximum mean error for any pixel</th>
<th>Overall mean error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thresholds</td>
<td>1</td>
<td>No</td>
<td>0.015</td>
<td>0.0015</td>
</tr>
<tr>
<td>32-bit floating point fast IDCT</td>
<td>1</td>
<td>No</td>
<td>0.0002</td>
<td>0.0000156</td>
</tr>
<tr>
<td>64-bit integer IDCT</td>
<td>1</td>
<td>No</td>
<td>0.0001</td>
<td>0.00000625</td>
</tr>
<tr>
<td>32-bit integer IDCT</td>
<td>1</td>
<td>Yes</td>
<td>0.0539</td>
<td>0.0289</td>
</tr>
<tr>
<td>Intel’s MMX IDCT</td>
<td>3</td>
<td>Yes</td>
<td>1.866</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table 15. ITU-T IDCT accuracy test results for various IDCT implementations.

* The test results are given only for numbers that are in the range of [-256, +255].

87
Chapter 5: MMX Implementation of the H.263+ Encoder

The IDCT accuracy is more important for H.263+ than it is for MPEG because in a typical H.263+ application only one frame (out of 100+ frames) is intra coded, whereas in an MPEG application a frame is intra coded every 10-15 frames. Nevertheless, Intel’s IDCT implementation can still be used in our H.263+ encoder provided that the encoder/decoder software runs on exactly the same type of processor. Only that, the codec would not be truly standard compliant.

5.6 Interleaved Data Transfers for Half Pixel ME

Interpolated and Reconstructed Picture Data

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p11</td>
<td>p12</td>
<td>p13</td>
<td>p14</td>
<td>p15</td>
<td>p16</td>
<td></td>
</tr>
<tr>
<td>p21</td>
<td>p22</td>
<td>p23</td>
<td>p24</td>
<td>p25</td>
<td>p26</td>
<td></td>
</tr>
<tr>
<td>p31</td>
<td>p32</td>
<td>p33</td>
<td>p34</td>
<td>p35</td>
<td>p36</td>
<td></td>
</tr>
<tr>
<td>p41</td>
<td>p42</td>
<td>p43</td>
<td>p44</td>
<td>p45</td>
<td>p46</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 34. Half pixel search block locations in interpolated reconstructed data.](image)

Half pixel ME consists of two computationally intensive parts: SAD computation and extraction of the search block from the interpolated reference picture. The MMX SAD function
implemented for integer pixel ME can also be used for half pixel ME. However, this will not speed up the other computationally expensive tasks that represent approximately 25% of all computations.

Figure 34 shows the interpolated picture that is used for prediction and the interleaved locations of the half pixel search blocks. In order to perform SAD computations using the interleaved data, the data should first be collected into an array. This operation requires irregular memory accesses and takes considerable time. Since the blocks consists of 8-bit data arrays, by using data pack instructions, it is possible to obtain a 700% speed increase for this operation.

Figure 35 shows the flowgraph of the MMX implementation. First, 16 bytes are read into two MMX registers, then an AND operation is performed with the mask “00FF00FF00FF00FF” to eliminate the data that will not be placed into the half pixel search block. Then, the remaining bytes are packed into one MMX register with either signed or unsigned saturation. Last, the MMX register is written back to the memory as the search block.

Figure 35. Flowgraph of the data interleaving for MMX Half Pixel ME function.
Chapter 5: MMX Implementation of the H.263+ Encoder

5.7 Interpolation

Interpolation is performed on every reconstructed I and P picture in order to predict the current picture in half pixel accuracy. Figure 36 explains the bilinear interpolation performed as specified in the H.263+ standard. Rtype is an optional parameter that improves the picture quality and can be signaled in the bit stream header.

\[
\begin{align*}
\text{p11} & \times a \quad \text{p12} \\
\times c & \quad \times d \\
\text{p21} & \\
\text{p22} &
\end{align*}
\]

\[\begin{align*}
a &= p11 \\
b &= (p11+p12+1-rtype)/2 \\
c &= (p11+p21+1-rtype)/2 \\
d &= (p11+p12+p21+p22+2-rtype)/4 \\
\text{rtype} &= \text{rounding type}. \text{When it is on, rtype is 1 for every other picture and 0 for the rest.}
\]

Figure 36. H.263+ interpolation.

Since the interpolation is done on 8-bit data and the result is also stored in 8 bits, it is possible to achieve a large speed up by processing 8 units of data simultaneously. Figure 37 shows the inner core of the interpolation function implemented using MMX instructions. First, the four 1x8 pixel vectors are loaded into the MMX registers and the data that will be processed is separated by using unpacking instructions. Next, the packed additions are performed and the results are written into 16-bit registers. Also, rtype is subtracted from each data and the outcome is shifted to the right by one or two depending on the position of the pixel. Last, the 8-bit results are packed together again and written back to the memory. In the outer loop, this operation is repeated for the one pixel below until the bottom line of the picture is reached. Then, starting from the top, the same procedure is repeated for the next 8 pixels in the horizontal direction.
Figure 37. The inner core of MMX implementation of interpolation.

The implemented MMX interpolation algorithm is more than three times faster than the optimized C implementation. This optimization does not change the rate-distortion performance of the encoder.
5.8 Motion Compensation

Motion compensation consists of simple addition operations of reconstructed residual blocks and prediction blocks. Since it is performed for all the coded macroblocks, this simple function takes a considerable computation time. In our software implementation, the difference and the prediction blocks are stored in 32-bit integer arrays. Thus, our MMX implementation gave less than 100% speed up. Obviously, implementations that use 16-bit arrays for motion compensation benefit more from use of MMX.

5.9 Load ME Area

MMX instructions can be used to speed up memory copy operations. A good example is the search block loading function for motion estimation. This function is called more than once for each macroblock. Using MMX instructions for simple read and write operations, an approximately 70% gain in performance is achieved.

5.10 Optimization of the MMX Functions

All of the MMX implementations that are explained above were manually optimized to fully utilize the MMX processor's scalar architecture. Currently, there are not any software optimizers written for MMX and probably there never will be such since Intel believes that programmers would do a much better job of optimizing the code than any of the optimization software. Following are some of the issues that were considered when optimizing the MMX source code.
5.10.1 Data Alignment

A misaligned access in the data cache or in the bus costs at least three extra clock cycles on the Pentium processor. A misaligned access in the data cache, which crosses a cache line boundary, costs nine to twelve clock cycles on Pentium processors. Intel recommends that 64-bit data be aligned on an 8-byte boundary. Using C it is not possible to guarantee this alignment for dynamically allocated memory, but a small adjustment to the memory pointer can provide alignment after the memory allocation procedure. Figure 38 shows the C routine ensures an \( n \) byte aligned array. If this routine is used, all the memory access should be done using \( \text{aligned}_\text{ptr} \). However the memory should be freed using \( \text{ptr} \).

```c
n\_byte *ptr;
n\_byte *aligned\_ptr;

int sizeofarray;
sizeof(n)

ptr = malloc(sizeofarray x sizeof(n) + sizeof(n));
aligned_ptr = (n\_byte *) ((unsigned int)ptr - ptr % sizeof(n));
```

Figure 38. C routine to create an \( n \) byte aligned array.

5.10.2 Optimization of Pipeline Usage

Optimization of the pipeline usage of MMX instructions provides most of the gain in terms of execution time. The MMX pipeline structure has been already discussed in Chapter 4. In this section, some techniques that enhance the pairing of instructions will be discussed.

As indicated before, MMX uses the two existing integer pipelines of the Pentium processor, namely the U pipe and the V pipe. MMX has only one multiplier and one shifter unit. Thus,
multiplication and shifting instructions, including pack and unpack, are not pairable with other multiplication and shifting instructions. The shift operation takes only one cycle. Therefore, it is necessary to insert at least one instruction between two shift instructions. On the other hand, since a multiplication takes three cycles to complete, at least two instructions should be put between two multiplication instructions. Also, multiplication results can only be used after two instructions. Instruction operand dependencies also effect instruction pairing. For instance, two instructions that one of them writes to a register and the other one reads from the same register can not be paired.

5.10.3 Memory Read and Write

The MMX Pentium processor has four write buffers. Therefore, at most four 64-bit data elements should be written to memory at a time. Unlike many other processors, in Pentium, reading from memory that is in the cache takes the same time as reading from a register. Hence, when optimizing the source code, using instructions that reads from the memory should be preferred to reduce the load on the registers so that they can be used in other instructions that can take only register operands.

5.10.4 Other Optimizations

Full optimization of the MMX source code involves many more techniques than explained in the preceding sections. For instance, taking into account that some instructions can be assigned only to the U pipe or the V pipe or understanding the branch prediction algorithm of the Pentium, it is possible to achieve more gains. Moreover, the Pentium has two instructions, RDMSR and

94
Chapter 5: MMX Implementation of the H.263+ Encoder

WRMSR, that give the exact utilization of pipelines, cache and other resources. These instructions can be used towards further optimization of the MMX source code.

5.11 Overall Performance Improvement of the Application

Our MMX implementation of a real-time H.263+ encoder is capable of encoding 10 frames per second in QCIF (176 x 144) resolution on a Pentium MMX 200 MHz processor. It is approximately 25% faster than the optimized C implementation. Table 16 shows a profile of the H.263+ baseline encoder for the FOREMAN sequence after MMX optimization. As seen from the table, since some of the functions were optimized more than others, the computational loads were re-distributed. Also, computation times of some of the functions which were insignificant before, such as Scan, have become important.

<table>
<thead>
<tr>
<th>Function</th>
<th>Computational Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT</td>
<td>9%</td>
</tr>
<tr>
<td>IDCT</td>
<td>9%</td>
</tr>
<tr>
<td>Quantization</td>
<td>8%</td>
</tr>
<tr>
<td>Load ME Area</td>
<td>7%</td>
</tr>
<tr>
<td>Scan</td>
<td>6%</td>
</tr>
<tr>
<td>Interpolation</td>
<td>5.5%</td>
</tr>
<tr>
<td>Fast ME</td>
<td>5%</td>
</tr>
<tr>
<td>Reconstruct Full Image</td>
<td>5%</td>
</tr>
<tr>
<td>Half Pixel ME</td>
<td>4%</td>
</tr>
<tr>
<td>Motion Compensation</td>
<td>4%</td>
</tr>
</tbody>
</table>

Table 16. Relative computational costs of the most computationally intensive H.263+ encoder functions after incorporation of the MMX functions.
Chapter 6

Conclusions and Future Research

In this thesis, we propose efficient coding algorithms and techniques that improve the performance of low bit rate (<64 Kbps) video coding significantly. These algorithms and techniques are implemented and tested using our public-domain software implementation of an H.263+ codec. The contribution of our research consists of two parts. In the first part, we develop platform independent algorithms, mostly using the global statistical properties of low resolution and slowly varying video sequences, for the most computationally intensive components of video coding. These proposed algorithms are zero block prediction prior to DCT, partial IDCT computations, fast half pixel motion estimation, quantization with look-up tables and partial SAD computations. These algorithms are discussed in detail in Chapter 4. In the second part, we propose MMX mapping strategies for the SIMD oriented video coding components. Through processing two, four and even eight data simultaneously we achieve major speed ups for some of the functions, particularly the DCT, SAD, interpolation, data interleaving for half pixel motion estimation and motion compensation functions. After employing the platform independent algorithmic optimizations and MMX mapping, we achieve a much faster encoder that can encode 15 frames per second (fps) on a Pentium MMX 200 MHz processor. The unoptimized version of
Chapter 6: Conclusions and Future Work

our encoder can only encode 5 fps on the same processor. More detailed performance results are presented in Appendix A. Also, two video/telephony applications that are developed to demonstrate the capabilities of the H.263+ video coding standard and our optimized encoder are presented in Appendix B.

Although this thesis focuses mainly on video encoding, some of the developed algorithms, such as the more efficient IDCT and the motion compensation algorithms, can be also used to increase the speed of video decoding. Moreover, the performance of the software implemented H.263+ encoder can be further improved by further optimization of the MMX optimized functions. This can be achieved by using advanced debuggers, such as Intel's VTune software package [22], that provides detailed information on the pipeline usage and cache misses of the MMX instructions.

Although some effort has been spent to improve the speed performance of the optional coding modes, such as the implementation of the 8x8 SAD function using MMX instructions, we focused mostly on optimizing the baseline components of the H.263+. It is possible to optimize some of the optional coding modes of H.263+, such as the Deblocking Filter Mode and the Improved PB frames mode. However, since the H.263+ optional modes usually require relatively complex functions that are performed on large data types, the hardware dependent speed improvements are expected to be not very significant.

Another interesting research direction is to extend the hardware dependent techniques, which were presented in Chapter 5, to similar SIMD architectures such as the VIS architecture of Sun's Ultra Sparc [49], the VLIW architecture of TI's C6X, and Philips' Tri+media chips.
Bibliography


[22] Intel VTune software web site at http://developer.intel.com/design/perftool/vtcd/


[33] MTTY Serial Communication software web site at

[34] W. B. Pennebaker, J. L. Mitchell “JPEG: Still Image Compression Standard”, Van

Architecture Software Developer's Manual Volume 2: Instruction Set Reference at
http://developer.intel.com/design/PentiumII/manuals/

pp. 42-50, August 1996.


Prentice-Hall, 1996.

[39] J. Ribas-Corbera and S. Lei, “Optimal quantizer control in DCT video coding for low-
delay video communications”, in Picture Coding Symposium, Berlin, Germany, Sept.
1997.

[40] K. Rijkse, “H.263:Video Coding for Low-Bit-Rate Communication”, IEEE


[42] Y. Senda, H. Harasaki, M. Yamo, “A Simplified Motion Estimation Using An

[43] A sample implementation of the 2D-IDCT we site at http://rnvs.informatik.tu-
chemnitz.de/~ja/MPEG/HTML/IDCT.html

[44] Sample MMX audio, communications and video routines on the Intel web site at
http://developer.intel.com/drg/mmx/appnotes/
Bibliography


List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC</td>
<td>Advanced Intra Coding</td>
</tr>
<tr>
<td>AIV</td>
<td>Alternative Inter VLC</td>
</tr>
<tr>
<td>AP</td>
<td>Advanced Prediction</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>bps</td>
<td>Bits Per Second</td>
</tr>
<tr>
<td>CBP</td>
<td>Coded Block Pattern</td>
</tr>
<tr>
<td>CIF</td>
<td>Common Intermediate Format</td>
</tr>
<tr>
<td>COD</td>
<td>Coded Macroblock Indication</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DF</td>
<td>Deblocking Filter</td>
</tr>
<tr>
<td>DPCM</td>
<td>Differential Pulse Code Modulator</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>EOB</td>
<td>End Of Block</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>fps</td>
<td>Frames Per Second</td>
</tr>
<tr>
<td>GOB</td>
<td>Group of Block</td>
</tr>
<tr>
<td>GSTN</td>
<td>General Switched Telephone Network</td>
</tr>
<tr>
<td>IA</td>
<td>Intel Architecture</td>
</tr>
<tr>
<td>IDCT</td>
<td>Inverse Discrete Cosine Transform</td>
</tr>
<tr>
<td>IPB</td>
<td>Improved PB-frames</td>
</tr>
<tr>
<td>ISD</td>
<td>Independently Segmented Decoding</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union Telecommunication Standardization Sector</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Picture Expert Group</td>
</tr>
<tr>
<td>kbps</td>
<td>Kilo Bits Per second</td>
</tr>
<tr>
<td>KLH</td>
<td>Karhunen-Loeve-Hotelling</td>
</tr>
<tr>
<td>MAE</td>
<td>Minimum Absolute Error</td>
</tr>
<tr>
<td>Acronyms</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>MB</td>
<td>Macroblock</td>
</tr>
<tr>
<td>ME</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>MV</td>
<td>Motion Vector</td>
</tr>
<tr>
<td>MVD</td>
<td>Motion Vector Data</td>
</tr>
<tr>
<td>MPEG</td>
<td>Moving Picture Expert Group</td>
</tr>
<tr>
<td>MQ</td>
<td>Modified Quantization</td>
</tr>
<tr>
<td>MSE</td>
<td>Minimum Squared Error</td>
</tr>
<tr>
<td>MSVC</td>
<td>Microsoft Visual C</td>
</tr>
<tr>
<td>NAN</td>
<td>Not A Number</td>
</tr>
<tr>
<td>PB</td>
<td>PB-frames</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal to Noise Ratio</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>RPR</td>
<td>Reference Picture Resampling</td>
</tr>
<tr>
<td>RPS</td>
<td>Reference Picture Selection</td>
</tr>
<tr>
<td>RRU</td>
<td>Reduced Resolution Update</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>SAC</td>
<td>Syntax-based Arithmetic Coding</td>
</tr>
<tr>
<td>SAD</td>
<td>Sum of Absolute Differences</td>
</tr>
<tr>
<td>SEI</td>
<td>Supplemental Enhancement Information</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>SS</td>
<td>Slice Structured</td>
</tr>
<tr>
<td>TMN</td>
<td>Test Model Near-term</td>
</tr>
<tr>
<td>UMV</td>
<td>Unrestricted Motion Vector</td>
</tr>
<tr>
<td>VIC</td>
<td>Video Conferencing Tool</td>
</tr>
<tr>
<td>VLC</td>
<td>Variable Length Code</td>
</tr>
<tr>
<td>VLD</td>
<td>Variable Length Decode</td>
</tr>
</tbody>
</table>
Appendix A

Overall Performance Improvements

Besides developing a number of hardware independent and dependent techniques to achieve an improvement in encoding speed, we also restructured our public domain H.263+ video encoding software performing faster function calls, re-arranging memory allocation operations and optimizing some of the computations. The overall performance improvements in encoding speed are summarized in Table 17. The table shows the encoding times for 100 frames of the high motion, FOREMAN, and low motion, AKIYO, video sequences. The simulations were performed on a PC with a Pentium MMX 200 MHz processor and 64 MB of RAM. The PC runs under the Windows95 operating system and our software does not use any of the multitasking features of the operating system. Each simulation was repeated several times and the numerical results were averaged. As can be seen from the table, the encoding speed improves almost 3 times in baseline coding when the fast MV search is enabled and twice as much in the other cases. Only a 100% speed improvement achieved for the full MV search case might seem to be surprising since 75% of all the operations correspond to SAD computations and the MMX-optimized SAD implementation is 3-4 times faster than that of unoptimized SAD one. However, this should be expected because, as indicated in Chapter 5, the partial SAD computation technique, which is very useful in full search ME, cannot be employed efficiently in the MMX implementation. Therefore, in the MMX implementation all the SAD computations are
Appendix A: Overall Performance Improvements

performed until the last row is processed, where in the C implementation, many of the SAD computations are terminated after processing, on the average, the first couple of rows.

<table>
<thead>
<tr>
<th></th>
<th>Encoding times before optimization (sec)</th>
<th>Encoding times after optimization (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FOREMAN</td>
<td>AKIYO</td>
</tr>
<tr>
<td>H.263+ Baseline coding (Full ME)</td>
<td>57.2</td>
<td>42.8</td>
</tr>
<tr>
<td>H.263+ Baseline coding (Fast ME)</td>
<td>23.5</td>
<td>17.3</td>
</tr>
<tr>
<td>H.263+: MQ, DF, AIC, AIV, IPB, UMV, AP modes turned on (Fast ME)</td>
<td>33.7</td>
<td>26.1</td>
</tr>
</tbody>
</table>

Table 17. The overall performance improvements in video encoding speed.
Appendix B

Example Applications for the H.263+ Video Coding Standard

The H.263+ video coding standard specifically targets low bit rate applications such as video telephony and video conferencing and low bit rate networks such as PSTN (Public Switched Telephone Networks) and wireless networks. In order to demonstrate the capabilities of this new video coding standard two applications were developed with the sponsorship of Avtel and the collaboration of the Technical University of Berlin. This appendix provides an overview of these applications.

6.1 Video Telephony/Conferencing over a Serial Link

A video telephony/conferencing application that allows communication of two PCs over a serial link was developed with the collaboration of Guy Cote and Michael Gallant. The project was sponsored by Avtel. The purpose of the application was to demonstrate that, using our H.263+ video encoding algorithms, it is possible to achieve video communication with acceptable quality levels over a link with as low as a 7200 kbits/sec bit rate. The target platform of such application is wireless networks. The application currently allows only one way video communication and speech transmission is not supported. In the future, the coded speech bit stream will be multiplexed with the video bit stream by using the ITU-T H.324 terminal standard.
Appendix B: Example Applications for the H.263+ Low Bit Rate Video Coding

In order to provide serial connection capabilities, our H.263+ encoder and decoder software is merged with a publicly available serial communication software: Microsoft MTTTY [33]. This Win32 application runs PCs with Windows 95/NT operating systems. Two wireless modems, each having a 9600 kbits/sec bandwidth, are connected to the serial port. The remaining 2400 kbits/sec bandwidth is reserved for transmitting speech. Figure 39 shows the platform and the simple block diagram of the application.

![Diagram of Video Telephony Application](image)

**Figure 39. Video telephony application over serial link using wireless modems.**

Since the bit rate is quite low, the smallest predefined picture format, sub-QCIF (128x96) is used. It is also possible to use larger picture formats, such as QCIF, but then the picture quality would be degraded. Also, a rate control algorithm, which is defined in [26], and its performance discussed in [7], is employed to keep the bit rate constant at 7200 kbit/sec. No error resilience
Appendix B: Example Applications for the H.263+ Low Bit Rate Video Coding

modes are employed because they are not supported in our H.263+ implementation. At very low bit rates, blocking artifacts increase and the color of the video becomes less visible because of the 2:1 downsampling and the coarse quantization of the color components. In this case, enabling the Deblocking Filter and Modified Quantization modes helps improve the subjective quality significantly without adding much computation time. Even when the processor power needed for frame grabbing and serial communication is included, we still manage to encode and decode approximately at the 10 fps rate.

6.2 Internet Video Conferencing

Figure 40. VIC application using the H.263+ codec.
The UCB*/LBNL** video tool, VIC, provides a real-time multimedia application for video conferencing over the Internet [48]. VIC uses the Draft Internet Standard Real-time Transport Protocol (RTP) for communication over internet. Even though it comes with some video coding standards embedded in the application, such as motion JPEG and the H.261, it is possible to use it for other video coding algorithms, such as ours. Gerd Knorr and Stephan Wenger from the Technical University of Berlin, Germany, developed an internet video conferencing application by merging the VIC tool and our publicly available H.263+ codec. The platform and the simple block diagram of the application is shown in Figure 40.

They also implemented the Slice structure mode and the Reference picture selection mode of the H.263+ for enhanced error resilience capabilities. Also, in order to provide synchronization points one frame every 5 seconds was intra coded. We recently combined our optimized H.263+ encoder with their application and achieved encoding performance of approximately 8-10 fps in QCIF resolution on an SGI O2 computer that has a175 MHz processor and 126 MB of RAM. In a T1 connection established between Germany and Canada at a non peak time, the packet rate loss was approximately 3-10% and there was almost no quality degradation.

* University of California, Berkley

** Lawrence Berkley National Laboratory