

THESIS

2

Prof

60362330

This is to certify that the
thesis entitled

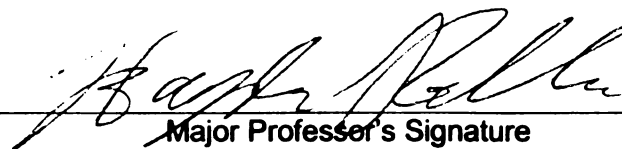
COMPLEXITY DISTORTION OPTIMIZATION FOR
VIDEO OVER WIRELESS DEVICES

presented by

Alan Paul Ray

has been accepted towards fulfillment
of the requirements for the

M.S. degree in Electrical and Computer Eng.


Major Professor's Signature

April 29, 2001

Date

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

**COMPLEXITY DISTORTION OPTIMIZATION FOR
VIDEO OVER WIRELESS DEVICES**

By

Alan Paul Ray

A THESIS

**Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of**

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

2004

ABSTRACT

COMPLEXITY DISTORTION OPTIMIZATION FOR VIDEO OVER WIRELESS DEVICES

By

Alan Paul Ray

The introduction of powerful compact wireless devices (such as handheld computers) raises new and unexplored aspects to optimizing complexity-distortion for video. Historically, the computational power of powerful platforms (e.g., computers) has exceeded the available communication bandwidth over emerging networks such as the Internet and wireless networks. Hence, improved performance was achieved by reducing the required transmission bitrate of video (through higher compression) and by increasing the computational complexity. Handheld devices do not follow the traditional balance of bandwidth and power; computational power is at a premium while bandwidth is comparatively more available. This thesis investigates the complexity-distortion options for a state-of-the-art video compression standard, H.264/JVT, which is an estimated three times more complex than the previous standards. The research focuses on analyzing the video standard's complexity for one of the most popular handheld devices, the iPAQ. Operational complexity-distortion and rate-complexity curves are presented for a number of Group of Picture (GoP) structures, entropy coding modes and compared against Pentium IV performance. The experiments include some of the earliest comparisons between JVT's entropy coding modes, context-adaptive binary arithmetic coding and content-adaptive variable-length coding.

TABLE OF CONTENTS

LIST OF TABLES	V
LIST OF FIGURES	VI
CHAPTER 1	
INTRODUCTION	1
Problem Background	1
Prior Work and Motivation.....	2
Objectives	4
Approach.....	5
Contributions.....	5
Thesis Overview	7
CHAPTER 2	
BACKGROUND	8
Standard Encoder Model.....	8
Input Frame	9
Motion Estimation & Prediction Error	10
Frequency Transform & Quantization	12
Decode Frame Prediction.....	13
Entropy Coding.....	13
JVT Decoder Implementation.....	14
Input Decoding.....	14
Dequantization and Hadamard Transform.....	17
Motion Prediction	18
Deblocking (Loop Filter)	21
CHAPTER 3	
IMPACT OF GROUP OF PICTURE TYPES ON REAL TIME PERFORMANCE	23
Hardware Selection.....	24
Handheld device - iPAQ:	25
PC Platform:.....	25
Software Selection	25
Operating System	26
Compiler and Software Development.....	27
JVT Decoder	28
JVT Encoder	31
JVT Player	32
Experimental Platform	33
Experimental JVT Video Sequence Settings	34
Experiment Variables:	34
Experiment Constants:	38
Primary (I,P,B,T) GoP Results by Complexity-Distortion:	39
Plot Specifications	41

Akiyo.....	42
Foreman	47
Mobile	52
Trends and Observations.....	56
 CHAPTER 4	
BITRATE COMPLEXITY ANALYSIS	59
Timing Probes.....	59
Actual Timing Probes	60
Calculated Timing Probes.....	63
Akiyo.....	64
Summary	66
Akiyo Tables and Figures	67
Foreman	74
Summary.....	76
Foreman Tables and Figures.....	77
Mobile.....	84
Summary	85
Mobile Tables and Figures.....	86
Conclusions.....	93
 CHAPTER 5	
CONCLUSION.....	96
Summary	96
Future Directions	97
 BIBLIOGRAPHY	100
General References	101

LIST OF TABLES

Table 1: Complexity Summary over 25-45 dB range.....	7
Table 2: Summary of Coding Variables	34
Table 3: Akiyo Complexity Deltas over 25-45 dB	44
Table 4: Foreman Complexity Deltas over 25-45 dB	49
Table 5: Mobile Complexity Deltas over 25-45 dB.....	54
Table 6: Akiyo Rate Invariant Probes (15fps, CE)	67
Table 7: Foreman Rate Invariant Probes (15fps, CE).....	77
Table 8: Mobile Rate Invariant Probes (15fps, CE)	86
Table 9: Complexity Summary over 25-45 dB range.....	94

LIST OF FIGURES

Figure 1: Basic Video Encoder Model.....	9
Figure 2: Group of Picture Illustration.....	20
Figure 3: First Frame of Video Sequences	35
Figure 4: Akiyo Complexity-Distortion (15 FPS, CE).....	42
Figure 5: Akiyo Complexity-Distortion (15 FPS, PC)	43
Figure 6: Foreman Complexity-Distortion (15 FPS, CE).....	47
Figure 7: Foreman Complexity-Distortion (15 FPS, PC).....	48
Figure 8: Mobile Complexity-Distortion (15 FPS, CE).....	52
Figure 9: Mobile Complexity-Distortion (15 FPS, PC).....	53
Figure 10: Akiyo Rate-Complexity (15 FPS, CE).....	68
Figure 11: Akiyo Deblock Rate-Complexity (15 FPS, CE)	69
Figure 12: Akiyo Deblock Rate-Complexity by Picture Type (15 FPS, CE).....	70
Figure 13: Akiyo ReadMB Rate-Complexity (15 FPS, CE, CABAC).....	71
Figure 14: Akiyo ReadMB Rate-Complexity (15 FPS, CE, CAVLC)	72
Figure 15: Akiyo ProcessMB exl. ReadMB Rate-Complexity (15 FPS, CE, CABAC) ..	73
Figure 16: Foreman Rate-Complexity (15 FPS, CE).....	78
Figure 17: Foreman Deblock Rate-Complexity (15 FPS, CE)	79
Figure 18: Foreman Deblock Rate-Complexity by Picture Type (15 FPS, CE).....	80
Figure 19: Foreman ReadMB Rate-Complexity (15 FPS, CE, CABAC).....	81
Figure 20: Foreman ReadMB Rate-Complexity (15 FPS, CE, CAVLC).....	82
Figure 21: Foreman ProcessMB exl. ReadMB Rate-Complexity (15 FPS, CE, CABAC)	83

Figure 22: Mobile Rate-Complexity (15 FPS, CE)	87
Figure 23: Mobile Deblock Rate-Complexity (15 FPS, CE).....	88
Figure 24: Mobile Deblock Rate-Complexity by Picture Type (15 FPS, CE)	89
Figure 25: Mobile ReadMB Rate-Complexity (15 FPS, CE, CABAC)	90
Figure 26: Mobile ReadMB Rate-Complexity (15 FPS, CE, CAVLC)	91
Figure 27: Mobile ProcessMB exl. ReadMB Rate-Complexity (15 FPS, CE, CABAC).	92

CHAPTER 1 INTRODUCTION

Problem Background

Compression is a crucial component of video coding. Uncompressed video is extremely space intensive, with an average resolution 320x240 pixel image taking up over 300 kilobytes, or over 27 megabits per second at the 30 frames per second standard. As video streaming grows and spreads, today's bandwidth and storage simply cannot support any significant level of uncompressed video. Previous standards such as MPEG-2, MPEG-4, and H.263 have successively introduced newer and improved approaches to video compression.

However, computer processing power has continued to rapidly grow, allowing the introduction of better compression standards to meet the rising use of video usage. High compression standards attempt to deal with difficulties such as increased network traffic, limited bandwidth, and storage space. The current state-of-the-art standard is the JVT¹/H.264 standard² developed by the Moving Picture Experts Group and Video Coding Experts Group (under ISO – International Standard Organization) and the Video Coding Experts Group (under ITU – the International Telecommunications Union) [1]. This standard has been developed as a response to “the need ... for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments.” [1]

¹ JVT is a Joint Video Team from the two aforementioned experts groups.

² Throughout this thesis, JVT, H.264, JVT/H.264, and H.264/JVT are used interchangeably.

Prior Work and Motivation

As discussed later, JVT provides significant advantages and improvements over previous standards in terms of higher compression ratios. One estimate put H.264 bit savings at 39% compared to MPEG-4, 59% over H.263 High Latency Profile, and 64% better than MPEG-2 [2]. Another study suggested around 50% coding gain over MPEG-2 and the baseline H.263, and 24% over the high profile H.263 model [3]. However, to obtain the improved compression rates, the JVT standard significantly increases the computational complexity. One study suggests that the baseline H.264 decoder increases complexity an average of 2.4 times compared to the previous H.263 baseline implementation [4]. Although much research has gone into reduced-complexity improvements to H.264, it remains a computationally complex standard.

While the increased complexity of the H.264 standard poses some problems for today's traditional desktop and laptop computers, these computers have grown rapidly in their processing power, and typically have specialized video cards and instruction sets. In addition, significant amounts of research are focused on reducing complexity for traditional computers. However, the spread of handheld computers and wireless networks introduces a new set of requirements to video decoding.

Handheld computers, with high-speed Internet and quality graphics, are clearly viable candidates for video decoding. Their wireless network access has the same bandwidth as any other wireless implementation, currently around one to five megabits per second, depending upon the technology being used, the environment interference, error handling

techniques, and the network implementation and usage. The typical handheld graphics, while adequate, are far less capable than a typical desktop. The handheld uses a 240x360 pixel screen, at least four times smaller than a desktop screen. A desktop's color range ranges in the millions of colors; a handheld device at sixty-five thousand or fewer (a maximum of 16 bits per pixel). More importantly, the handhelds typically lack specialized video cards, powerful processors, or extra instruction sets.

The handheld processors are slower, both in terms of clock rate as well as optimizations. Because of power, space, and heat issues, architectural speedups are typically not implemented to the same degree as for standard systems. Thus a 400Mhz processor for a handheld is not as powerful as a 400Mhz processor for a desktop. Additionally, handheld devices do not run the same operating systems as desktops. Thus, the needs and limitations for a video decoder over a handheld are likely to be significantly different from video decoder optimized for a desktop.

Specifically, bandwidth may be much less of an issue compared to the lack of computing power. Furthermore, due to the different operating system and architectural designs, handheld optimization may not particularly correspond to desktop optimizations. However, designing a new video standard for handheld devices is undesirable for several reasons. First, video will often be generated for multiple destinations, possibly without knowing the platform of the end-user. In addition, video is often encoded for multiple uses (handheld, storage, desktop) and encoding video multiple times for different applications is less than desirable. Furthermore, the development of additional standards

defeats one of the major purposes of having a standard: To have a universally accepted format for video encoding. JVT, designed and backed by both the MPEG and ITU standards organizations, is likely to become the next generation standard in many applications. However, little research has been done for addressing the questions involved in how JVT can be best implemented on handheld devices.

Since handheld devices, due to their low storage space, are better suited for real-time video than to storing video, this thesis focuses on the real-time performance of a handheld decoder with a variety of approaches to decoding JVT video sequences. Special attention is paid to the time involved in the subparts of video decoding. One previous study [5] breaks JVT decoding process into six main sections³: The deblocking filter, interpolation, inverse transform and quantization, variable length decoding (VLD), intra prediction, and miscellaneous. Based on using a Pentium III processor, this previous study estimated that the deblocking filter takes 20-35% of the time, interpolation (including motion prediction) another 35-50%, intra prediction less than 1%, inverse transform and quantization 7-11%, variable length decoding 4-9%, and miscellaneous around 6-21%. The wide range of results reflects the use of different video sequences.

Objectives

The purpose of this research was three-fold: To understand the impact of high level encoder options such as entropy coding and Group of Picture (GoP) structure on complexity-distortion; to develop a handheld specific complexity model to guide further

³ The following chapter elaborates on and provides some background on these parts of the video decoder.

handheld decoder optimizations; and to generate a real-time JVT decoder for a handheld device.

Approach

Since prior research, such as the study by Lappalainen et al. suggests interpolation accounts for a large portion of the decoding time, an initial research focus is the impact of the frame type composition upon the computational complexity. The frame type composition, or Group of Picture (GoP) types, is a significant factor in controlling the amount of motion prediction required by the video decoder.

Contributions

The major contributions of this research are broken down into four sections. The first contribution, developing the available reference software into a viable real time H.264 decoder, provided the underlying software used for performance analysis throughout the rest of this research. In addition, the real time H.264 decoder ensures the practical application of the research to actual decoding.

The second contribution, discussed in chapter three, provides a baseline complexity for the H.264 decoder under different parameters, especially the impact of motion prediction through controlling the picture frame types. The exploration of these parameters gives insight into the complexity control available at a high level through varying the encoder options. This contribution focused on the impact of I, P, and B-frames on the complexity-distortion curves generated by different levels of quantization in the encoder. This

contribution highlights the generally optimum performance of P-frames, even when no rate-limit is imposed on the sequence.

The third contribution, examined in chapter 4, breaks down the decoder into sub-components and presents the rate-distortion for each aspect. It identifies the portions of the decoder that are complexity invariant, regardless of rate, and the components that are rate-dependent. Additionally, this section breaks out the section that goes into parsing, which is highly linear based on rate, and also breaks out the decoding (non-parsing) complexity for I, P, and B frames. This section highlights the significance of parsing, especially with a sub-optimal implementation, as well as the large decoding time, especially with B frames.

The fourth contribution compares the iPAQ results to the same experiments run on a Pentium IV processor. The contribution highlights that the performance differences between the two processors are not simply scaled versions of each other. The Pentium speedup varies significantly based upon the component of the decoder. This variance strongly suggests that algorithms and optimizations from one processor will not necessarily be equally beneficial on the other processor and that separate handheld research is needed. This contribution is discussed in both chapter 3 and 4 after the relevant iPAQ discussion. Table 1 provides an overview of the complexity and parsing rates for a range of sequences.⁴

⁴ This table is discussed in more depth at the conclusion of chapter 4.

	Frame Type	iPAQ 400MhZ	Pentium IV, 1.5GhZ		PC Speedup (Times Faster)	
					<i>Low End</i>	<i>High End</i>
		<i>Decoding, excl. Parsing (ms/frame)</i>				
	<i>I</i>	11.5-13.5	2.75-3.10		4.18	4.35
	<i>P</i>	21.5-22.5	3.95-4.45		5.44	5.06
	<i>B</i>	33.0-48.5	5.40-7.10		6.11	6.83
		<i>Parsing (bytes/ms)</i>				
		<i>CABAC Coding</i>				
	<i>I</i>	250-280	1100-1300		4.40	4.60
	<i>P</i>	160-250	720-1100		4.50	4.40
	<i>B</i>	80-210	450-1000		5.63	4.76
		<i>CAVLC Coding</i>				
	<i>I</i>	135-245	600-1130		4.44	4.61
	<i>P</i>	50-100	380-520		7.60	5.20
	<i>B</i>	30-80	360-500		12.00	6.25

Table 1: Complexity Summary over 25-45 dB range

Thesis Overview

The remaining chapters are organized as follows: Chapter 2 discusses video coding standards in general and the important features of the JVT standard. Chapter 3 looks at the complexity-distortion of difference GoP sequences as well as contrasting the PC and iPAQ performance. Chapter 4 breaks down iPAQ decoder performance into its components and looks at the component performance based on compression. Chapter 5 concludes with summarizing the contributions in more detail and looking at future work.

CHAPTER 2 BACKGROUND

This chapter discusses some key basics and well-established concepts for video decoders, the critical portions of the H.264/JVT standard, and the previous studies into H.264 and wireless complexity. The first section of this chapter provides a general summary of how a video sequence is broken down for coding and provides a very brief synopsis of the generic encoding process. The next section steps through the JVT standard decoding process of a correctly coded video stream, starting with the incoming stream, then looking at coefficient dequantization, motion prediction, and finally image deblocking. These sections include exploration of previous research on JVT complexity issues as well as the few previous studies on implementing video decoding on handheld platforms.

Standard Encoder Model

The typical video compression model is based on an encoder, either real-time or offline, that is encoding a series of pictures, transmitting the encoded video sequence over some channel, possibly with additional channel coding, where the sequence is received by a decoder, which reverses the encoding process and generates a sequence similar to the original. The similarity to the uncorrupted original sequence, often referred to as the signal-to-noise ratio, depends upon the amount of information discarded by the encoder while compressing the sequence and any optimization in the decoder that neglected transmitted information. Many standards, including the H.264 standard, specify only the correct (“standard compliant”) input and output of the decoder; for the encoder, any algorithm that generates a correctly encoded (compliant) stream is acceptable. Likewise,

any decoding process that takes a valid input stream and produces the correct decoded video sequence is acceptable. In practice, there is a standard encoder model as shown in Figure 1. The rest of this section examines each of these seven components (input frame, motion estimation, prediction error, frequency transform, quantization, decode frame prediction, and entropy coding).

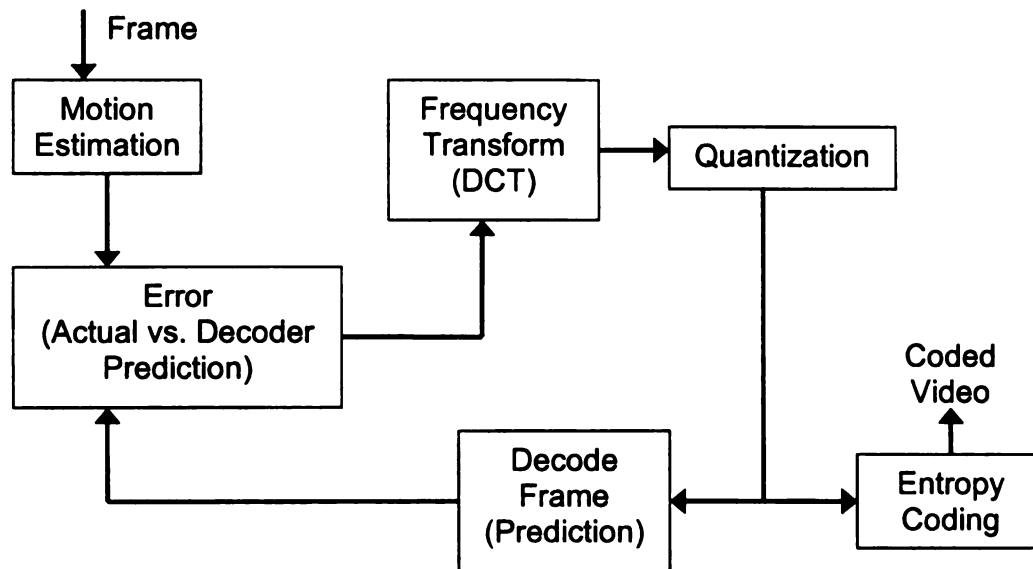


Figure 1: Basic Video Encoder Model

Input Frame

Video frames, typically in YUV format, are sequentially fed into the encoder. YUV is a color representation scheme where Y represents the intensity, or luminance (luma), of the color. The U (or Cb) and V (or Cr) components contain the chrominance, or color component. (Cb and Cr primarily contain the blue and red chrominance components of the original Red-Green-Blue (RGB) pixel, respectively; whereas the luminance Y primarily contains the green component.) Because the human eye is much more sensitivity to intensity than to color, the chrominance is often specified with fewer bits. One of the significant advantages of the YUV scheme is the ease of sending more detail

regarding the luminance by including more bits for the Y component. The second way of exploiting the YUV scheme, used by popular video compression standards such as JVT, is to use a single sample of the UV component for multiple samples of the Y component. One format that is used by the MPEG and ITU series of video standards (including JVT) is referred to as the YUV 4:2:0, which implies that for each pixel, the Y component is encoded. However, a Cb and Cr sample are only taken every other column and every other row, effectively one UV sample per four Y samples. Various schemes are used to properly average the UV components over all four Y samples. Currently the JVT standard supports 8-bits per Y, U, or V sample, although support for additional bits is expected [6].

Motion Estimation & Prediction Error

Motion estimating plays a major role in enabling modern video compression standards to achieve high coding efficiency. Motion estimation is traditionally achieved by dividing the image frame (under consideration by the encoder) of the video sequence (being compressed) into square (or rectangle) regions. The classic subdivision of a frame is into sixteen by sixteen (16x16) pixel blocks known as *macroblocks*. Each macroblock, therefore, contains a 16x16 array of the Y luminance information, and two 8x8 arrays, one for the Cb component and the other for the Cr component. (The 4:2:0 YUV format has four times the Y information compared to the U and V.) The macroblock is the fundamental block which is encoded, transmitted, and then decoded. Note that the standard video formats, such as QCIF (176x144 pixels) are multiples of 16x16. QCIF, for example, contains 99 macroblocks (11x9).

When the encoder receives a frame, it breaks the frame (*current frame*) down into macroblocks and for each macroblock in the current frame, searches for the most similar 16x16 area within the previously coded frame (*reference frame*). Once the most similar 16x16 area (“best match”) in the reference frame is identified by the motion search process, the encoder then takes the difference between the two: the macroblock in the current frame and the 16x16 best-match area in the reference frame. The 16x16 area (“best match”) in the previous frame is the same size as the macroblock of the current frame that is being coded; however this 16x16 area might consist of multiple macroblock subsections of the previous frame. The range of the search depends upon the type of picture being encoded and the encoder options. Some picture types only allow searching within the same frame for similar macroblocks, others allow looking at one or more previous reference frames. The number of reference frames depends upon the encoder options; more frames increases the encoder complexity significantly. The definition of “most similar” macroblock depends upon the encoder implementation. Numerous schemes for increasing the search efficiency and performance have been published. For example, Andy Chang et al. propose a fast estimation procedure that reduces computational complexity by forty percent compared to a standard full search [7]. The difference between the current block and the reference block is taken because usually the difference contains smaller values than the original macroblock and thus requires fewer bits to encode. In addition to the difference, a motion vector is sent specifying the reference 16x16 area (in the reference frame) for the current macroblock.

As discussed later, the encoder, after encoding each frame, immediately (internally) decodes the frame and stores it locally to serve as a reference decoded frame for the next (current) frame. This allows the encoder to be “in synch” with the decoder at the receiver (as explained further below). In other words, this enables the encoder to use the correct “reference” frame that will be available at the true decoder when computing the difference signal. (Recall that the receiver will not have access to the original lossless frames, and it will only have access to the decoded frames, distorted due to compression.)

Frequency Transform & Quantization

Once the Y, U, and V coefficients have been determined along with any motion vector, the coefficients of the macroblock undergo a frequency transform. This transform is usually a discrete cosine transform (DCT). Here, since the data is in arrays of sizes such as 4x4, 8x8, or 16x16, the transforms can easily be applied. The result is a set of coefficients that describe the samples and motion vectors from in the frequency domain rather than pixel by pixel. Many of these components tend to be zero or very small, further reducing the amount of information needing to be transmitted. The JVT standard uses a special Hadamard transform, discussed later, to avoid multiplications or floating point discrepancies by the decoder.

Once the frequency coefficients are determined, they are quantized to further reduce the amount of information. The simplest form of quantization is to simply divide each coefficient by some number and send the integer result, discarding any remainder. The decoder simply multiplies the remainder by the coefficient, which is also sent, and its

final coefficients are close to the original. More complicated schemes with scaled quantization parameters are actually used to improve quantization performance. Careful selection to the quantization equations in the standard enable multiplication and division to be avoided and replaced with shifting and additions. This quantization step is the primary lossy step where information is discarded by the encoder that cannot be recovered by the decoder. The level of quantization is determined by the encoder options.

Decode Frame Prediction

Because of the loss of information from quantization, and possibly from the frequency transform (mainly due to minor loss in the precision of computation), a decoded frame is not exactly the same frame that was earlier encoded. Since the decoder is using motion prediction based on past decoded frames, not past original frames, the encoder needs to select the motion prediction based on the actual decoder frames. Otherwise, the motion vectors for the macroblocks will point to different coefficients than expected, and the newly decoded frame will not be correct. Since future frames will be based on past frames, the error will propagate and grow, adding significant distortion. To avoid this problem, the encoder includes a decoder which decodes each encoded frame. These decoded frames are then used for motion estimation, and so the motion estimation is correct regardless of what loss the encoder introduces.

Entropy Coding

The quantized macroblock coefficients, motion vectors, and other overhead information are now ready for lossless *entropy encoding* to minimize the amount of transmitted data.

Channel coding and transmission protocols, which compensate for lossy channels, are irrelevant to the video standard although they clearly impact the overall video performance. Many standards, including JVT, include advanced features for handling lost data at the cost of additional complexity and bandwidth. The JVT standard supports two entropy coding standards, discussed shortly in more depth as part of the JVT decoder. Entropy coding represents the final encoding step and the output is the encoded video, which is then channel coded, and sent to the decoder.

JVT Decoder Implementation

This discussion of the key aspects of the JVT standard are subdivided into several parts. The first portion deals with the decoder input and entropy decoding. The next focuses on the coefficient dequantization and inverse transform. The final parts examine motion prediction and the deblocking filter. Schwarz and Wiegand provide a more in-depth discussion of many of these aspects in [2].

Input Decoding

The first segment of the decoder is processing the incoming video stream and then reversing the entropy coding. The video stream processing is based upon breaking the stream down into pre-defined packets. The packets then specify the protocol and data within each packet, and the data is extracted in the appropriate manner.

Network Abstraction Layer Unit (NALU)

Since the JVT standard is designed to work independent of any network or channel technology, JVT uses its own packet model, called a network abstraction layer unit (NALU), which is very similar to an Internet Protocol (IP) packet. The NALU encapsulates all the frame data, whether motion vectors, frequency coefficients, or flow control data. The incoming stream is searched for the unique NALU startcodes, then the decoder processes one NALU at a time. NALUs may contain either flow control data or picture data. (Flow control data involves information such as the reference frame buffer handling, slices, and macroblock ordering schemes.) Different NALU types contain different data packed in according to that NALU type.

Entropy Coding: CALVC & CABAC

Depending upon the encoder options and the NALU type, the data is unpacked appropriately. Two major types of entropy coding are used to pack the data. The first, a combination of universal variable length codes (UVLC) and context adaptive variable length codes (CAVLC). Throughout this thesis, this combination is generally referred to by the CAVLC abbreviation, although this method of entropy encoding uses both. However, previous coding standards generally relied on a pure UVLC entropy approach. The context adaptive VLC codes use a combination of different lookup tables, where the context is the nearby macroblocks. In addition to the lookup tables, additional syntax is used to specify the location of zeroes in between the non-zero coefficients. A more in-depth explanation can be found in [8], as well as the original standard proposal [9]. Horowitz et al. [4] briefly note in their decoder complexity analysis that their empirical

tests indicate CAVLC is about two times more complicated than the previous pure UVLC defined by JVT. However, they do not examine CAVLC complexity beyond that brief footnote.

The other entropy method that is supported by the JVT standard is context based adaptive binary arithmetic coding (CABAC). CABAC is an extension of arithmetic coding.

Arithmetic coding works by encoding a series of coefficients as a floating point number.

The more bits, the more specific the floating point number and the more coefficients can be encoded in a single number. Arithmetic coding can nearly achieve the source entropy because it uses the probability distribution to determine which floating-point numbers correspond to which series of coefficients. Since the probability distributions are variable for different video sequences, and even within the same sequence, the JVT model recalculates the probabilities so that the distributions are representative of the recent past. Studies such as [10, 11] suggest that the context based arithmetic coding represents a 10-25% bit improvement over the UVLC implementation, depending upon quantization parameter and video sequence. A study by Saponara et al. suggested that 16% was the upper limit [12].

In terms of prior research into coding complexity, both CABAC and CAVLC are relatively new coding schemes, and more focus has been given to the bitrate improvement of CABAC than to its complexity [10, 11, 13]. This research does, though, clearly note the bitrate advantages of context based coding. A 1987 study by Witten et al. [14] between arithmetic coding and Huffman coding (which is traditionally the core of

VLC type of coding) suggests that arithmetic coding was generally superior to Huffman coding using the computers of that time. However, the study did not examine context based coding, nor did it attempt to fully optimize the Huffman implementation. (Huffman codes, like VLC codes, generally rely on table based decoding.) The H.263 annex E first introduced CABAC to video standards. The JVT standard used an improved low-complexity table-based implemented, and a multiplication free coder was recently proposed in [15]. However the study by Saponara et al., using version 2 of the JVT reference software (pre-CAVLC), reports that CABAC had only a minor complexity increase but provided no experimental numbers for entropy coding [12]. It's important to remember that the context based usage of VLC tables differs from previous video standards which relied upon a single universal table.

Dequantization and Hadamard Transform

$$W = \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{array} \right] Z \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{array} \right] / 2$$

Z = Frequency Coefficients

W = Unscaled Spatial Coefficients

Equation 1: Luma DC Hadamard Inverse Transform

As mentioned earlier, many previous standards have used discrete cosine transforms (DCT) for the space-frequency domain transform. Because the DCT relies on floating point approximations irrational numbers, precision is platform and implementation dependent. This imprecision leads to “drift,” the same problem that occurred when

motion vectors were based off of the original frames rather than the decoded frames and led to a decoder instance in the encoder.

To prevent this drift problem, as well as to improve performance, the JVT standard implements a 4x4 integer variant of the DCT. All of the transforms, one of which is shown in Equation 1, can be implemented as a series of one dimension transforms which can be implemented using only addition, subtraction, and bit shifts for the multiplications and divisions. The JVT standard uses a 4x4 transform, where previous standards use a larger 8x8 transform. The smaller transform allows the JVT to use smaller macroblock divisions, as discussed below. A fuller treatment of the JVT Hadamard transform progression and implementation is given by Richardson [16]. The transform specifics vary by the type of sub-macroblock they are decoding; luma DC blocks and chroma DC blocks have their own variants of this process. Once the unscaled spatial coefficients are reconstructed by the inverse transform, they are rescaled by multiplying by a factor determined using lookup tables referenced by the quantization parameter.

Motion Prediction

Each image is encoded and decoded one macroblock at a time. The encoded information for each macroblock includes the motion vectors and the differential coefficients. For images with any significant amount of motion, simply using the difference with the previous image provides poor correlation compared to elsewhere, both within the same picture and previous pictures. In order to best exploit the sequence correlation, two types of motion are exploited. The first, *spatial correlation*, tries to use portions of the same frame as a base for encoding the remainder of the frame. The other, *temporal correlation*,

uses previously encoded frames as the basis for encoding. The difference between the current block and the selected match is the motion vector, which is generally encoded as the difference between that motion vector and previous motion vectors.

The encoder decides how best to encode the macroblock. It can encode the macroblock as one large 16x16 chunk, or subdivide it into 8x8, 16x8, or 8x16 blocks. Those blocks can further be divided into 4x4, 8x4, or 4x8 sub-blocks. The more blocks that are used, the better match the motion compensation can be to that particular block. Chang et al. simulations suggest as much as a 15% bitrate improvement by using the subblock sizes in addition to the whole macroblock [7]. However, more blocks increase the information needed by adding more motion vectors. Blocks may be *intra*-coded, taking advantage of spatial correlation, or *inter*-coded, pointing to previously coded frames. Inter-coded blocks may be *predictive*, using one motion vector, or *bi-predictive*, using two motion vectors. The *bi-predictive* mode allow a block to combine information from two previous blocks, saving on coefficient information but increasing the computation required on the decoding end.

Frame Types & Picture Buffering

The frame type, determined by the encoder options, determines which coding modes are used in coding that frame. An intra-frame, or *I-frame*, only uses intra-coded blocks. Thus it is completely self-decoding as it has no motion vectors pointing to previous information. However, *I-frames* tend to be the largest type of frame because they do not take advantage of any temporal correlation. *P-frames* are predictive frames, which can use either intra coding (intra-coded macroblocks) or a single motion vector per

macroblock. This motion vector points to the “best match” area (normally of size 16x16 pixels) with the previously coded I-frame or P-frame. And the bi-predictive frame (*B-frame*) use two motion vectors, which can point to different reference pictures. The pattern in which an encoder uses these frame types is known as the Group of Picture (GoP) type. The encoder uses this pattern to decide which picture type to encode next, resulting in the pattern repeating over and over. Figure 2 illustrates two classic GoPs. The top one, IPPP, consists of a single I-frame followed by P-frames. Each P-frame uses the previous P-frame as a reference frame. The second, IBBPBBP, consists of an I-frame, then a repeating pattern of two B-frames followed by a P-frame. Each P-frame uses the previous I or P-frame as a reference; the each B-frame uses the previous and next P-frames as reference. (The first two B-frames use the I-frame as a reference since there is no previous P-frame.)

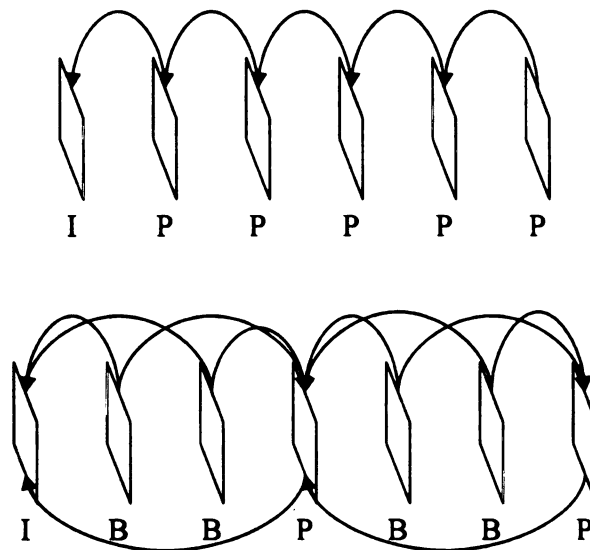


Figure 2: Group of Picture Illustration

One-quarter pixel prediction

The goal of the motion compensation is to find a vector that reduces the macroblock coefficients to as close to zero as possible. In theory, motion may not happen in pixel steps, and a higher degree of precision may find better reference matches. Previous video standards, such as MPEG-2 [17], have frequently used *half-pixel*, or half-pel, motion vector precision. Increased precision on the motion vector increases the search complexity for the encoder. Research into motion search techniques demonstrate that improved methods are possible. For example, Panusopone and Baylon present a half-pel 8x8 search method in [18] than can be generalized to different block sizes and resolutions. Previously, the added precision did not reduce bit usage for previous standards. Werner [19] proposed a filtering method to overcome signal noise in motion estimation. For JVT, Wedi and Musmann extended Werner's proposal into a functional method for using one-quarter and one-eighth pel precision [20]. The specific implementation in JVT, discussed briefly by Wieland et al. in [21], uses a finite input response filter for the half-pel estimation, and then averages the integer and half-pel values for the quarter-pel prediction. This process is clearly more complex than full-pel decoding.

Deblocking (Loop Filter)

Because the encoder and decoding is done in blocks, the edges of the blocks tend to have seams, which create visual artifacts and blockiness along the block boundaries. To help clear up this distortion, decoded macroblocks are run through a deblocking filter. The deblocking filter runs first on the vertical edges of the block, then on the horizontal

edges. The filter serves to average out values along the edges, thus giving a smoother look to the finished frame. This filter is an integer based filter, so it can be quickly implemented.

CHAPTER 3

IMPACT OF GROUP OF PICTURE TYPES ON REAL TIME PERFORMANCE

Research shows that the JVT standards provide a computationally-complex algorithm for video decoding. While significant research has been done into complexity reduction methods for the JVT standard[6, 7, 22], little research has been done into relating the problem to mobile devices. An exception is Yu et al. proposing a real-time mobile *encoder* in [23]. Lappalainen et al. did conclude that real time decoding for the *H.263 standard* should be possible for low frequency mobile processors and that 12-38Mhz of a Pentium III was required for 10fps QCIF decoding [5]. Horowitz et al. analyzed the JVT complexity from by looking at the number of average operations to decode a macroblock and then extrapolating that into complexity based on processor specifications [4] The previous study, however, largely ignored the entropy decoding complexity.

As highlighted in the previous chapter, the JVT standard provides a wide range of video coding tools and related options. By varying these options, the JVT codec can provide a wide range of video quality; at the same time, these coding options imply a wide range of complexity that is encountered by the JVT decoder. Hence, a key objective of this thesis is to identify high-level JVT coding options that provide the best quality (*minimum distortion*) while adhering to a maximum *complexity constraint*. Hence, our work establishes an initial baseline for the *complexity-distortion* performance of JVT decoding over handheld devices. Consequently, a variety of video sequences were encoded to explore the impact of frame rate, entropy coding, group of picture type, and video sequence upon the decoding performance. Equivalent tests were performed using a

Pentium IV to provide a comparative-performance baseline. The baseline tests (for both handhelds and standard computers) used a *lossless decoder* and rejected any optimizations which would represent a *lossy decoder*⁵.

The purpose of these experiments was three-fold: First, to answer the question “Given that a video sequence is encoded for a handheld device using the JVT standard, how should one proceed to encode that sequence to achieve the best possible quality under a given complexity constraint?” Second, to identify the broad over-arching complexity of the JVT decoder implementation chosen. Finally, to establish the performance of this decoding software compared to that used in previous research. Our experiments investigated the complete range of quantization choices, four different frame rates, both entropy encoding methods, four common Group of Picture (GoP) types, and Intra-coded (I-frame) frequency variance for two common GoPs.

This chapter is organized as follows. First, the hardware and software components of the experiments are described. Then the video encoding configurations used in the experiments are described. Subsequently, the complexity-distortion results are presented in graph form with corresponding analysis.

Hardware Selection

The significant hardware choices were first the type of handheld, and then the standard computer used as a reference comparison for the handheld performance.

⁵ Here, a *lossy decoder* implies a decoder that discards some information/data that is already sent by the encoder for the purpose of reducing complexity.

Handheld device - iPAQ:

The iPAQ series, built by Compaq and later by HP after the Compaq / HP merger, is a series of handheld devices that are representative of the capabilities of the generic handheld devices. They are designed as multipurpose consumer devices. They possess simple graphics, using 16 bits per pixel – 6 bits for green and 5 for red and blue. (Many of the early mobile devices used this 5-6-5 RGB format but did not support 65K colors.) The processors are simple but fast 32-bit processors, capable of speeds between 200 and 600 Mhz. The selected iPAQ used in this research is the h5550 model. Designed by Intel, the 5550 model employs a 32-bit processor targeted for mobile environments. The iPAQ represents a commercial integrated product typical of a generic mobile computing device.

PC Platform:

To provide the PC baseline, a system running Windows 2000 with a Pentium 4 1500 Mhz chip and 256 megabytes of RAM was used. Aside from the normal system services and background tasks (e.g. virus detection, firewall) no other loads were running during simulation testing.

Software Selection

In order to prepare the experiments, the following software was used. The general principles behind the selection were to use the commercially available state-of-the-art, to use pre-developed solutions as a baseline and then improve the bottlenecks, and to initially avoid any degrading in video-quality or decoder-speed due to software changes.

Improvements which sacrificed video quality for improved complexity were not considered at this point (i.e., we focused on *lossless decoding*).

Operating System

The handheld devices are developed and designed with Windows Compact Edition (WinCE) in mind. Thus hardware interface and control problems are minimized. WinCE is a light version of the traditional Windows system. It supports most of the basic Windows features, but it uses relatively few of the advanced features. Graphically, for example, WinCE uses its own high speed graphics library (GAPI) for direct access to the screen rather than the standard DirectX used by other systems. The WinCE software development kit (SDK) supports few of the libraries that are normally available. The operating system, however, supports minimal configuration of its memory management, thread priorities, or file handling. However, because of WinCE's simplicity, availability on a wide variety of handhelds, and support by Microsoft, it makes an excellent platform for examining the actual issues involved in real-time video decoding over low-complexity devices. The software development kit supports all standard C/C++ syntax as well as wireless internet, file I/O, display writing, basic timing, and 32-bit arithmetic, giving the basic building blocks for video decoding. Additionally, WinCE is designed to interface with the desktop, facilitating desktop controlled program execution, video transfer, and data collection for experiments. Given that gigabytes of video sequences would eventually be generated, this automation is a valuable component. For those interested, others have written Linux versions to replace the default WinCE operating system. The open source aspects of Linux would facilitate further research into operating

system improvements to speed up decoder performance. However, significant work in terms of a reasonable coding environment (see below) would be needed for such a change.

Compiler and Software Development

Since the current iPAQs are based upon the PXA255 or related processors, they use the ARM instruction set. Microsoft's freely released Embedded Visual C++ 3.0 (EVC3) and 4.0 (EVC4) development kits were used to facilitate development of handheld executables. Version 3.0 represents an early version of WinCE; version 4.0 represents the current version. Early design and development was done with version 3.0 although the results presented throughout this thesis are based on a more updated iPAQ which used the Visual C++ 4.0 environment. While EVC allows a handful of Microsoft/ARM specific optimizations, and some of these were used at specific bottlenecks in the program, the majority of the code remains standard C/C++ and can be ported to any platform.

All linking was done with the included Microsoft linker, while compilation was done with Intel's Windows Compiler for Embedded Visual C++ (later combined into Intel's Window's Compiler program). Initial trials used the Microsoft compiler but the Microsoft compiler incorrectly compiled code for a single array resulting in incorrect array values in the decoder. The code compiles correctly with both Visual Studio .NET (discussed later) and the Intel compiler. Speed comparisons between the two compilers showed no statistically significant variance in performance based upon the compiler.

A WinCE software development kit called Port SDK [24] was used to enable the use of standard console I/O in the WinCE environment. Port SDK also provided a simpler method to execute the compiled decoder since the decoder did not need to create its own windows environment. A generic console application [25] actually served as a DOS console under WinCE. (WinCE has no native console.). Preliminary tests showed no significant overhead to using the PortSDK/console when compared to the standard windows interface. All timings ignore program initialization and cleanup, including thread and window initialization. The console interface allowed for standard I/O methods to be used instead of specializing for the WinCE, increasing portability and reducing complexity.

Microsoft Visual Studio .NET 2003 was used for compiling and coding the Pentium executable version of the code, along with the necessary data gathering environment. Two different profilers, Intel's VTune and CompuWare's DevPartner Profiler were used to profile the video decoding complexity on the Pentium platform. In the handful of places where the operating system calls differ significantly, separate coding sections are included to properly interact with the code. Methods for accessing the system time were the most common difference. Threads also showed the potential to require different interfaces. These areas in the code are the most likely to cause difficulty in the event of porting to a separate system but should be easily identifiable.

JVT Decoder

The JVT team, as part of their standard, is developing algorithmically correct implementations of portions of the H.264 standard. One of these is version 6.1e [26].

Version 6.1e contains a fully functional encoder and decoder for the standard features, including all of those used in this research. Some of the advanced features, such as slices, flexible macroblock ordering, and error correction were not implemented as of this release. (Later versions, in an attempt to improve the encoder and decoder and add additional features, leave the encoder and/or decoder inoperable.)

As mentioned previously, the JVT technical specification for the decoder is an ideal decoder. Numerous ideas have been proposed for reducing complexity in exchange for a degraded video signal. For the purpose of complexity overview, I implemented generic software improvements which did not reduce the video signal quality. Some additional changes were made to make monitoring software performance easier since the available profiling tools for WinCE are impractical on non-custom hardware. The significant changes were:

Removal of system calls: All extraneous file inputs and outputs, such as output storage and verification were removed. Thus the decoded frames were not written to a file, nor compared against the “ideal” video. Selected sequences were viewed to ensure that the decoder still functioned properly after code changes. Additionally, the majority of the memory allocation and de-allocation calls were moved to the program initialization or clean-up stage rather than the timed portion of the decoding. My experiments showed that repeated memory cleanup and reallocation was shown to be using as much as 30-35% of the decoding time based on the JVT software version that was initially considered. As mentioned elsewhere, this heavy system load is one of the significant differences from

the desktop, where the same memory tasks were taking an estimated two or three percent of the run time.

Addition of circular frame buffer: The decoder handling of the decoded frame storage was changed to a circular buffer implemented through linked lists. Instead of copying decoded frames into a second memory location for storage, the pointer to the newly decoded frame was added to the list of decoded frames and a new pointer to an unused frame was selected. When a frame was displayed, the decoder returned the frame space to the list of unused frames for future use. This change also facilitated video decoding by allowing the decoder to pass a memory pointer to the video player, by naturally reordering out of sequence frames⁶, and by adding a decoded frame buffer to smooth decoding irregularities. The addition of the buffer provided a nominal 5% improvement or so.

Addition of timers: Due to the severe lack of profiling software for the WinCE environment, a number of timers were inserted into the software to obtain timing information for key sections of the code. As of this research, the author is unaware of any quality profiling approaches to complicated WinCE tasks. A handful of private non-commercial solutions are available but these are incredibly overhead intensive, seem susceptible to misprofiling functions, and interface poorly with other software. Function counts can be found through profiling the decoder on the Pentium platforms. However, as

⁶ The out of sequence frames occur because a B frame, because of prediction, may be dependent on a frame that occurs after the B frame. When this is the case, the frames that the B frame requires for prediction are sent first, which means that the frames are not sent in order.

noted by the difference in memory allocation complexity, the handheld environment response does not necessary correlate to the PC.

These inserted timers provide an overview of how the decoding time is being spent within the decoder by partitioning the decoding time into roughly six different areas such as overhead, frame decoding, macroblock decoding, and macroblock reading. Tests indicated that timers inserted were sufficiently few in number that there was negligible impact upon the average run time.

Modification of frequently called functions:

Frequently called functions, such as the NALU locator, the start code search routine, and portions of the symbol decoding routines were rewritten to handle the most frequent case more quickly. These changes involved techniques such as grabbing larger chunks of files at a time, storing decode bits rather than recalculating each time, and an advanced search approach. An estimated 15-25% improvement was seen as a result of these improvements.

JVT Encoder

The original 30fps YUV video test sequences were encoded using version 6.1e of the JVT encoder [26]. This encoder, provided by the JVT team, correctly handled the options involving entropy coding, frame rate, and GoP structure. Advanced options such as those involving error control, flexible macroblock ordering, and search restrictions were not fully implemented as of this release. This release is one of the earlier implementations of

the CAVLC entropy standard in the JVT source code⁷. Later releases began implementing these features but also broke other components of the encoder. The encoder was used “as is” with the exception of changes to its input and output routines to provide the information in a format more conducive to interaction with the experimental platform (described below).

JVT Player

A separate Windows video player was co-developed with another colleague in the WAVES lab to display the decoded frames from the JVT decoder on the screen. Separate CE and PC versions were developed to take advantage of the different display technologies. The CE version, the portion that I worked and developed, operated as a separate thread from the decoder. The player thread calculated an interval i milliseconds that it should sleep and then woke up, asked the decoder for a pointer to a frame to display, displayed the frame, and went back to sleep for i milliseconds. Mutexes, a thread synchronization method, prevented threads from interfering with each other and delay conditions handled the buffer underflow and overflow cases.

Experimentation suggested that the best case display time for a QCIF image using GAPI interface was about six (6) milliseconds, and a fullscreen image would require about seventeen (17) milliseconds. This additional overhead meant that to display a 5fps sequence real time requires a decoding speed of 5.2fps, a 10fps sequence requires a 10.6fps rate, and a 15fps sequence about 16.5 rate. The actual required rate will vary depending on the actual performance of the video driver.

⁷ The CABAC and CAVLC entropy coding modes are discussed more fully in Chapter 2.

Experimental Platform

In order to efficiently encode, decode, and record experiment information, an automatic experiment executor was built. Its functionality included several main components:

Automatic Encoder: Given a list of experiments, the executor generated the appropriate encoder configuration file, ran the JVT encoder, waited for the encoder to finish, and recorded the bit counts and PSNR values. It also then stored the results in the appropriate result file and moved the necessary input files for the decoder into the appropriate directory.

Automatic Decoder: Given a list of experiments and the platform, the executor copied the decoder input and configuration files into the proper location, ran the appropriate decoder, and read in the resulting timings from the decoder run. The results were stored within the result files. The generator was capable of tracking different configurations and comparing configurations against a specified optimum run.

Plotter: Given a list of experiments and the desired data calculation, the executor would parse the result data and generate a file which specified how to plot the requested information. This file was specially formatted to be fed into Matlab function which then interpreted the text into a plot.

Experimental JVT Video Sequence Settings

The first subsection discusses the encoding and decoding settings that were varied for the experimental results. The second subsection explains settings that were fixed for this research.

Experiment Variables:

Video Sequence: Three different video sequences were used; all three are standard 10-second long video coding test sequences. Figure 3 shows a sample frame from each of the sequences. On the left, the Akiyo with the talking anchorwoman in a news studio; in the center, Foreman with a

Parameter	Value
Video Sequences:	Akiyo, Foreman, Mobile
GoP Structures:	I, P, B, T
I Frame Frequencies:	4, 8, 12, 16, 20
QP Values:	0-51
Frame Rate:	5, 10, 15, 30
Entropy Coding:	CABAC & CAVLC
Video Format:	QCIF (176x144)
Hadamard Transform:	On
Max. Search Range:	16
Num. Reference Frames:	1
Forced Intra-Macroblocks:	None
Slices:	Unused
SP Frames:	Unused
Loop Filter Parameters:	Default

Table 2: Summary of Coding Variables

foreman talking about the construction site, which is shown at the final frames; and on the right, Mobile with the moving train and calendar. The first, Akiyo, is a shot of talking news anchor woman. It features minimum motion, high temporal and spatial correlation, and is highly compressible. Akiyo represents an ideal video sequence. The second, Foreman, is a higher motion sequence of a construction foreman talking and then showing the surrounding landscape. It features higher motion and lower temporal correspondence compared to Akiyo and is more representative of a typical video sequence. The third sequence was Mobile, which is a high motion video with lots of detail. It is very difficult to compress and it loses rather significant quality with compression.



Figure 3: First Frame of Video Sequences
Akiyo (Left), Foreman (Center), Mobile (Right)

Group of Picture (GoP) structure: Four main GoP structures were used, and then sub-variants of two of those were further investigated. The four main GoP structures are referred to here as I, P, B, and T. The I-GoP pattern is an all I-frame sequence. This GoP ignores temporal correlation and is a sequence of separate images, like stringing together a series of JPEG images (i.e., similar to *motion JPEG*). The I-GoP has a high bitrate due to minimal compression but also has no motion prediction (i.e., low complexity, in general). The P-GoP is a mostly P-frame sequence with an I frame every 12 frames. That is, an I frame, 11 P frames, then a second I-frame and 11 more P-frames, and so on. The repeated I frame gives the decoder a periodic non-dependent frame which prevents long term signal degradation from occurring as well as providing a moderate level of error control. Even if a frame is corrupted in transmission, a complete picture will arrive within a second or two, depending upon the frame rate.

The B-sequence replaces every other frame with a B-frame, so the pattern becomes I-B-P-B-P-...I-B-P-B-P-... The frequent use of B-frames takes advantage of temporal correlation in both directions and thus increases the compression. However, bi-directional motion prediction increases the complexity during decoding. The final GoP, T, uses two

B frames instead of one: I-B-B-P-B-B-P. This pattern further optimizes any correlation between frames. In both the B and T GoPs, an I frame repeats every 12 frames, regardless of whether the intermediate frames are P or B-frames.

The two sub-variants profiled involved changing the frequency of the I-frames for the P and B sequences. The P-GoP variants are I4X, I8X, I16X, and I20X where the number is the I-frame frequency. Thus the P-GoP is equivalent to I12X and the I-GoP equivalent to I1X. The B-GoP variants are B4X, B8X, B16X, and B20X, again with the number indicating I-frame frequency and B-GoP being B12X. B frames repeat themselves every other frame, so B4X has the pattern I-B-P-B-I-B-P-... The complexity response to changing the I-frame frequency gives insight into how frequent image refreshing impacts performance compared against longer sequences of motion prediction. These results were remarkably similar to the primary four GoPs examine (I, P, B, T) and specific analysis is not presented for them. In general, their complexity variance is directly related to I-frame frequency and follows the data shown in Table 9.

Frame Rate: The original 10 second long video sequences, were taken at thirty frames per second for a total of 300 frames. The sequences were then encoded at four different frame rates: Five, ten, fifteen, and thirty frames per second (FPS). 15 FPS corresponds to skipping every other frame. 10 FPS to skipping two of every three frames, and 5 FPS to skipping five of every six frames. In theory, skipping fewer frames should increase the temporal correlation between frames, enabling higher compression and a lower bitrate.

However, the since the complexity increase verses the bitrate reduction is unknown, the increased correlation may not aid computational complexity.

Quantization Parameter (QP): Video sequences were generated for all valid QP values, which range from 0 to 51. 0 is almost uncompressed video, thus having extremely high bitrates and excellent picture quality. 51 is the lowest quality, with minimum bitrate and typically unacceptable picture quality. For most of the analysis, the working subrange of 25-45dB PSNR was selected; 25dB is at the low end of what is considered acceptable video, and 45dB consists of an excellent picture. Thus 25-45dB represents a working range within which to evaluate performance. Different QP were selected for this range for each video sequence as the QP that generated the range varied based on frame rate, GoP structure and video sequence.

Entropy Coding: The JVT standard allows for two types of entropy coding, context based adaptive arithmetic coding (CABAC) and context-adaptive variable length coding (CAVLC). Previous research indicates that CABAC decreases the bitrate by 10-15% compared the equivalently coded UVLC sequence [11]. However, CABAC is traditionally more computationally complex and the bit reduction is less likely to be useful with the complexity limitations of the iPAQ. When comparing CAVLC with CABAC, Wiegand et al. estimate CABAC gains 5-15% in bitrate savings [21], but with the highest savings in interlaced television signals (which are not being investigated in this thesis). Within the test sequence, our experiments suggested that for a fixed rate, CABAC coding offered a distortion improvement ranging between zero and one decibels.

Experiment Constants:

Video Format: Due to the screen-size of most handheld devices (320x240 pixels or smaller), the Quarter Common Intermediate Format (QCIF) (176x144 pixels) standard was used. The other common standard is Common Intermediate Format (CIF) which is four times larger (352x288 pixels) and far larger than most handheld devices can accommodate without additional scaling. The standard video test clips come in standard formats and the choice of QCIF reduced any degradation from issues not related to the encoding process.

Hadamard Transform: The encoder used the standard Hadamard transform to provide better encoding results. A less accurate approximation is also available which does not use the Hadamard Transform.

Forced Intra-Macroblocks: Forced intra-macroblocks, blocks that are compressed without motion prediction, are normally used to refresh the signal quality and prevent error propagation. Since the encoded video sequences included I-frames at regular intervals, no additional intra-macroblocks were added.

Maximum Search Range: The search range determines how far from the block center the encoder will search when looking for the most similar block. Here a standard value of 16 pixels, the size of one full macroblock, was used.

Number of Reference Frames: The number of reference frames determines how many previous (or future) frames are searched when looking for the most similar block. In my research, a single reference frame provided the minimal performance improvement when compared with multiple reference frames. However, multiple reference frames lead to a significant increase in encoder complexity. Several recent JVT studies has used five reference frames [4, 5], but provide no rationale for this choice. We believe that any level of prediction that is higher the one used in B-frames (i.e., the usage of more-than two reference frames) corresponds to a significant level of unreasonable complexity that should not (cannot) be used for handheld devices.

Block Search Restrictions: No block search restrictions were enabled. The encoder was allowed to choose 16x16, 16x8, 8x8, 8x4, or 4x4 as it found the best matches.

Slices, SP Frames, Loop Filter Parameters: None of these special features were used. Only a single slice was used, no switching frames were used, and the default loop parameters were used.

Primary (I,P,B,T) GoP Results by Complexity-Distortion:

For each of the three sequences (Akiyo, Foreman, and Mobile), the results and analysis are broken into sections. The first section contains the results and an overview of the important points. The next two sections are a performance comparison between the CABAC and CAVLC modes and then a comparison between the CE and PC results. For

Foreman and Mobile, a brief synopsis is then presented between the current sequence and the previous discussed sequences.

The results shown and discussed below is the decoding complexity in terms of frames per second (FPS) plotted against the signal to noise ratio (PSNR) in decibels. The higher the signal to noise ratio, the better the quality is. At 0 QP, the PSNR is around 60-70dB, depending on the sequence. Anything around 45dB or higher is considered excellent signal quality. The distortion shown, 25-45dB represents the range between a rather highly distorted signal (25dB) and a high quality signal (45dB). The run times are measured in milliseconds or clock cycles, depending on which was most appropriate for the measurement scale and then converted to FPS. FPS provides normalization since the sequences differ slightly in their coded length based on their GoP pattern. The data is shown for a single test, as repeated tests on the same sequence indicated slight (less than 2%) deviation between runs. Furthermore, since each test only slightly different from the tests before and after it, the entire run from QP 0 to 51 serves as verification to the accuracy of individual runs. Certain tests, even when repeated, seem to be unusually quick or complicated, and this presumably is related to a combination of encoding choices and architectural caches. Because of its significantly faster microprocessor, the PC is more vulnerable to fluctuations than the handheld.

It is important to note that the data and analysis presented in this chapter assume that *adequate bandwidth is available* for receiving any of the available sequences. All bitrate constraints are ignored. Also, the time to quickly read the data from a file is assumed to

be the same as the time needed to receive the video stream over a wireless connection. However, the actual complexity between file reading and network reception may differ significantly.

Plot Specifications

In the plots presented in this thesis, the constant variables for a plot are specified in the title. The video sequence is abbreviated to the first three letters. The GoP structure has an additional specification: ‘-x’. *x* specifies the exact frame type from a sequence that is being plotted. ‘-A’ is the average of all frames, ‘-i’ is the I-frames, ‘-p’ the P-frames, and ‘-b’ the B-frames. While only an average SNR number is available, frame-type specific data is available for time and bitrate results. Clearly, frame-specific availability is dependent upon the sequence GoP: A P-GoP has no data for the B-frames, for example. Experiment variables that are constant for all experiments in a plot are printed in the title; the plot variables are labeled in the legend. The title also specifies the limits of the data shown in the plot. There are three limits used: An SNR limit, a kilobytes per frame (KBPF) limit, and a millisecond per frame (MSPF) limit. The default (“no limit”) setting is a PSNR of between 25 and 45dB, and any complexity and bitrate. All three limits are for the overall performance of a sequence, not the particular frame types. For example, the average size of all frames in a T-GoP will be much higher than the average B-frame size. However, the limit filter uses the average of all frames, even when only plotting the B-frames.

Akiyo

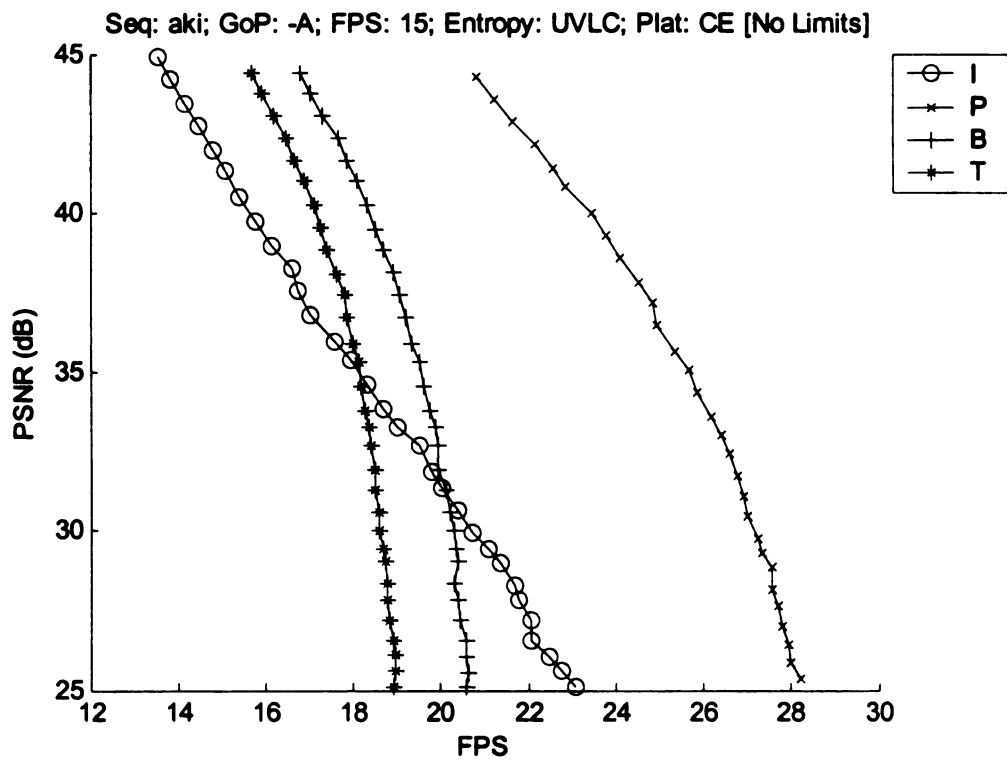
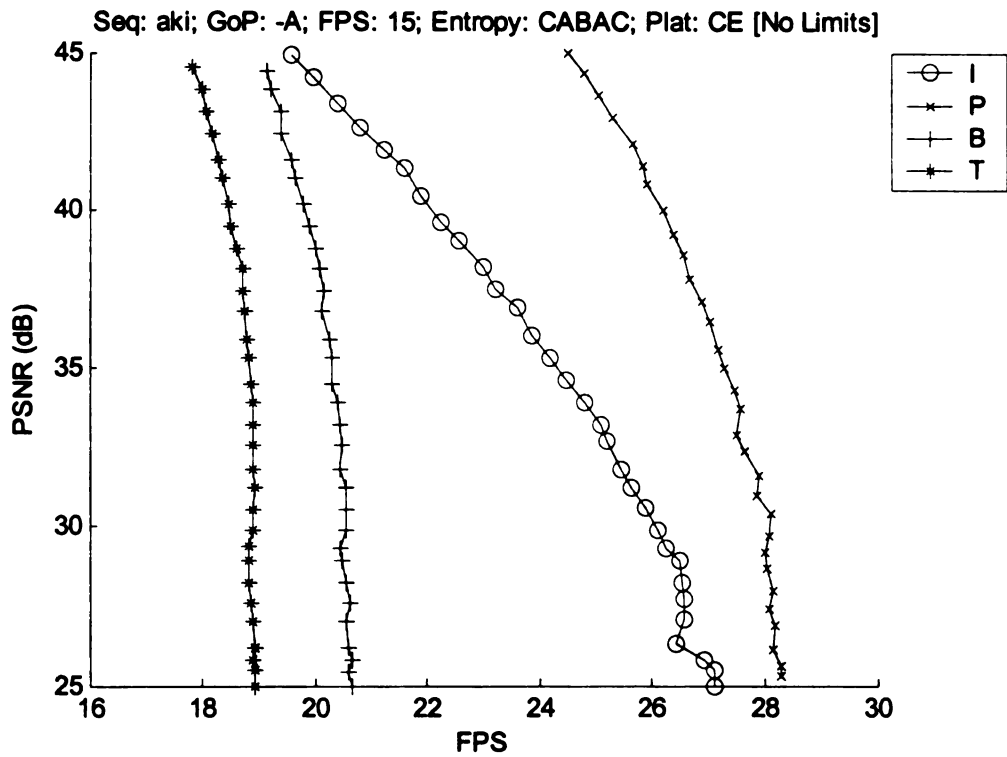


Figure 4: Akiyo Complexity-Distortion (15 FPS, CE)

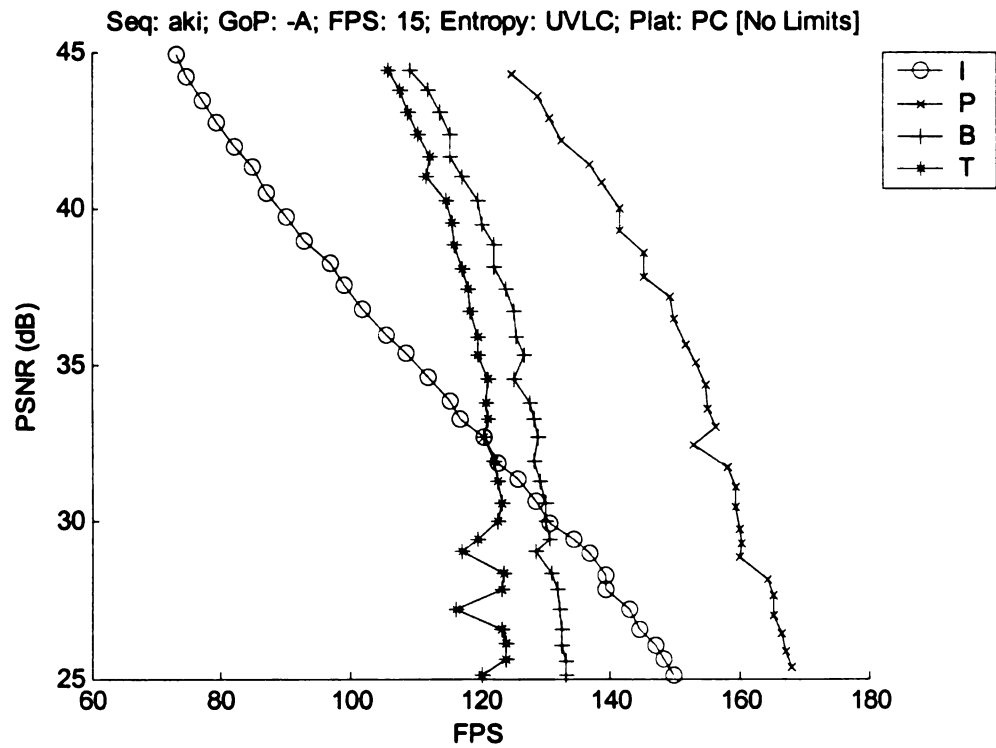
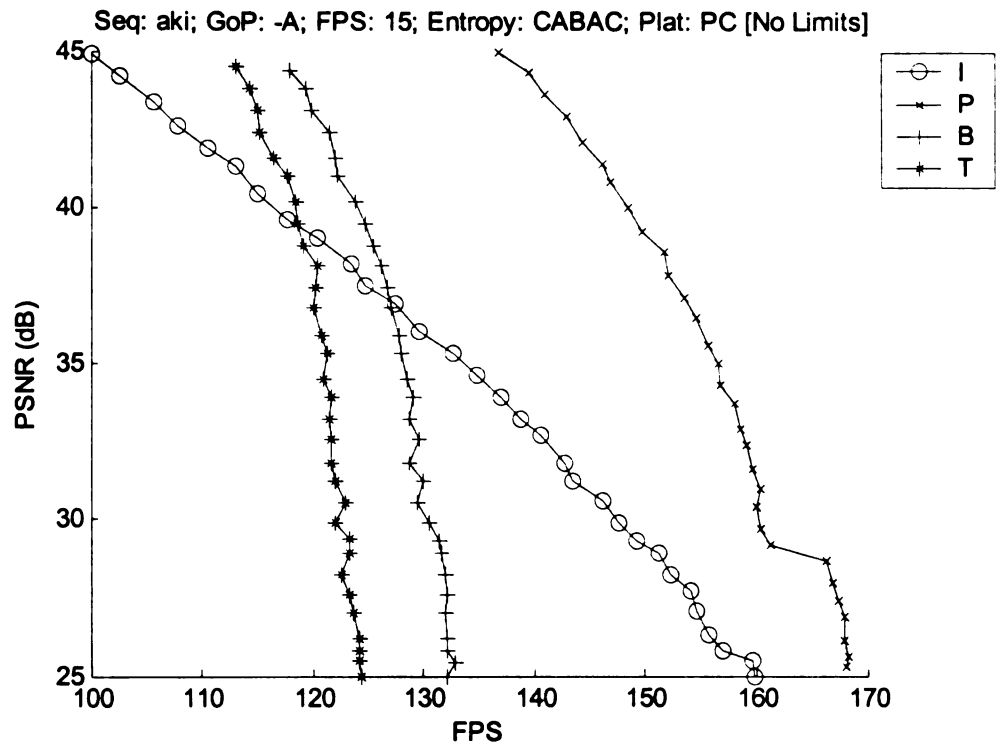


Figure 5: Akiyo Complexity-Distortion (15 FPS, PC)

Overview

Akiyo Seq:	Time (ms): slowest minus fastest		Percent Slowdown		% Entropy Diff.	
	<i>CABAC</i>	<i>CAVLC</i>	<i>CABAC</i>	<i>CAVLC</i>	<i>25dB</i>	<i>45dB</i>
<i>CE:</i>						
<i>I</i>	13.9	28.5	37.6	65.5	17.8	41.7
<i>P</i>	5.2	12.7	14.7	35.8	0.0	18.4
<i>B</i>	4.1	9.7	8.4	19.9	1.0	11.7
<i>T</i>	3.2	9.9	6.2	18.8	0.0	11.9
PC:						
<i>I</i>	3.8	6.3	60.0	94.8	6.7	29.9
<i>P</i>	1.3	2.1	21.7	36.0	-1.2	10.4
<i>B</i>	0.9	1.4	11.9	18.8	-0.8	5.4
<i>T</i>	0.9	0.8	10.6	9.1	4.2	2.7

Table 3: Akiyo Complexity Deltas over 25-45 dB

Figure 4 and Figure 5 show the handheld and PC performance for the primary GoP types (I, P, B, T) when run at 15 frames per second for both entropy types. The 15 fps sequence is shown for two reasons. First, it is the fastest framerate that the decoder can playback real time on the tested iPAQ platform. As suggested by Figure 4 and confirmed by tests on 30fps sequences, the decoder stops short of reaching 30fps even for a very low quality Akiyo sequence. Secondly, and among framerates that are lower than 30 fps, 15 fps maximizes the temporal correlation since only every other frame is removed. (Lower frame rates, such as 10 or 5 fps, ignore more frames within the sequence.) While this similarity matters less to Akiyo since every frame is extremely similar, it does mean that the P and B frames achieve better compression for the Foreman and Mobile sequences discussed later.

Clearly for the handheld, the P-GoP is the best compression approach for the easily compressed Akiyo sequence. For any quality level, it has the lowest complexity (highest framerate). Table 3 summarizes the overall complexity change for the 25-45 dB. The time is the number of milliseconds longer that decoding a frame at 45dB takes than

decoding at 25dB. The percent slowdown is the time delta as a percent of the fastest time to decode a frame, that is the time at 25dB. The percent difference is the percent slower that a CAVLC sequence is compared to the same decibel CABAC signal. Thus for the handheld P-GoP, moving from a low quality signal to a high quality signal is only a 15% (5.2ms) complexity increase using CABAC, while using 45dB CAVLC is 18% slower than using a 45dB CABAC signal. Note the PC and CE percentages can be compared against one another because the percentages are being taken against an individual baseline performance for each platform.

Several key conclusions stand out from these experiments. First, there can be significant differences in how handheld and desktop computers respond in complexity to distortion changes. Between architectural, operating system, and compilation differences, the same performance gains tested and developed for personal computers cannot be assumed to have equivalent efficiency on a handheld device. Second, the B and T-GoPs, with their high motion prediction and higher compression, are relatively stable in terms of complexity regardless of the distortion, entropy coding, and platform *for highly correlated sequences*. Finally, the I-GoP complexity varies drastically depending upon distortion, entropy coding, and platform.

Handheld: CABAC & CAVLC

Surprisingly, CAVLC entropy coding is not a significant improvement over the CABAC coded sequences. The decoder software source notes mention that the CAVLC decoding procedures have not been fully optimized and even the additional optimization described earlier has not fully caught up to the CABAC implementation. It is worth noting that at

high distortion and excluding the I-GoP, there is relatively little difference between the two entropy coding methods. Since the I-GoP uses the least compression (and thus a much higher bitrate compared to the other GoPs) and has the highest complexity difference at high distortion levels, *these results suggests that the amount of input data that must be parsed may be as significant as the computation it takes to parse the data.* This suggests that memory and file I/O bottlenecks may be an important factor in addition to available processor time. (This aspect will be further discussed and analyzed in the next chapter.)

Handheld vs. PC

Interestingly, the PC has a much harder time with the I-GoP as distortion increases compared to the iPAQ: The standard computer takes ninety-five percent longer to decode a 45dB CAVLC sequence compared to a 25dB one. It also took sixty percent longer to decode a CABAC sequence 45dB compared to the 25dB. The iPAQ, comparatively, takes only 66% longer for CAVLC and 38% longer for CABAC. While the iPAQ certainly slows down with lower distortion, its I-GoP performance is significantly less hindered: In Figure 4, the CABAC I-Gop is always faster than the B or T-GoPs; not so for the PC (see Figure 5). Correspondingly, the CE CAVLC I-Gop stays much closer to the T-GoP performance than the PC I-Gop stays to its corresponding T-GoP.

Foreman

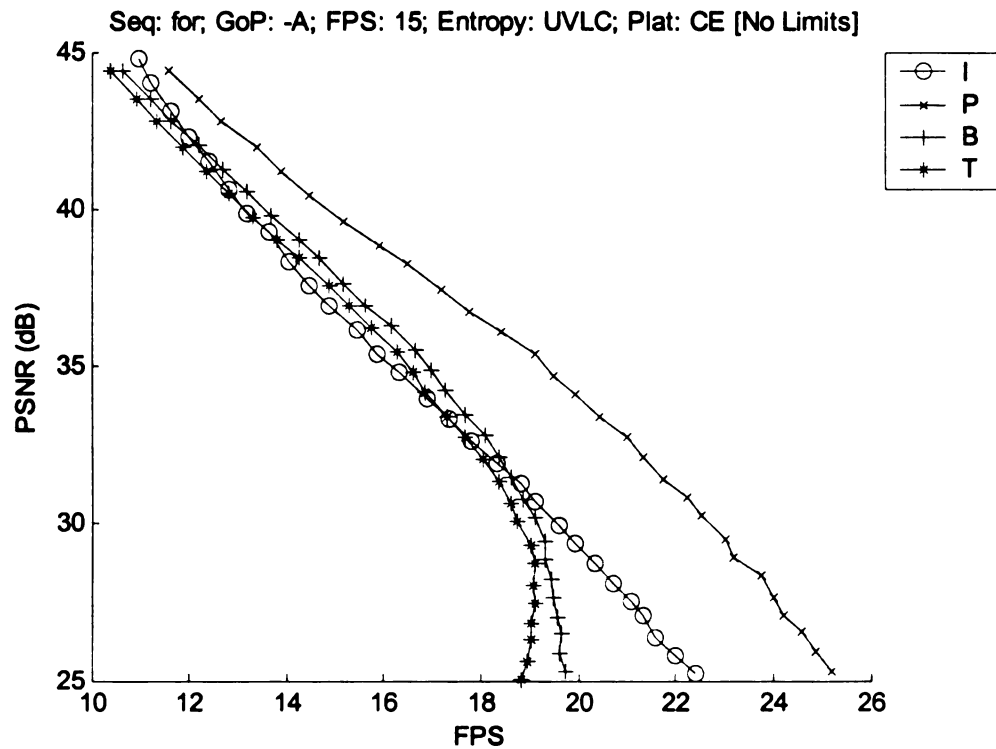
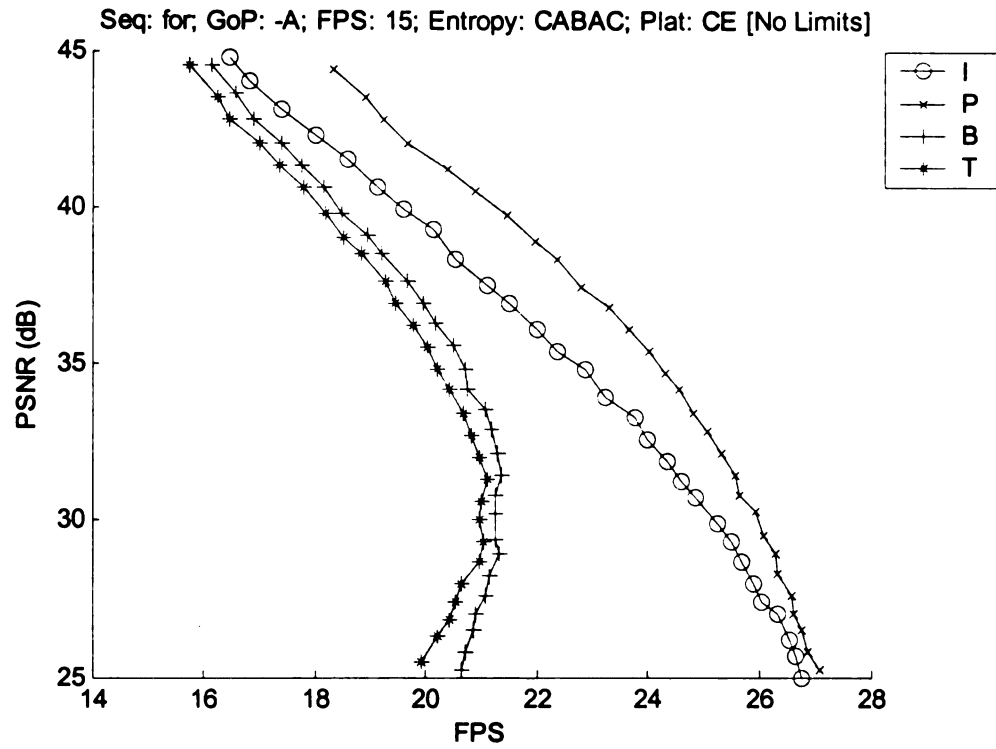


Figure 6: Foreman Complexity-Distortion (15 FPS, CE)

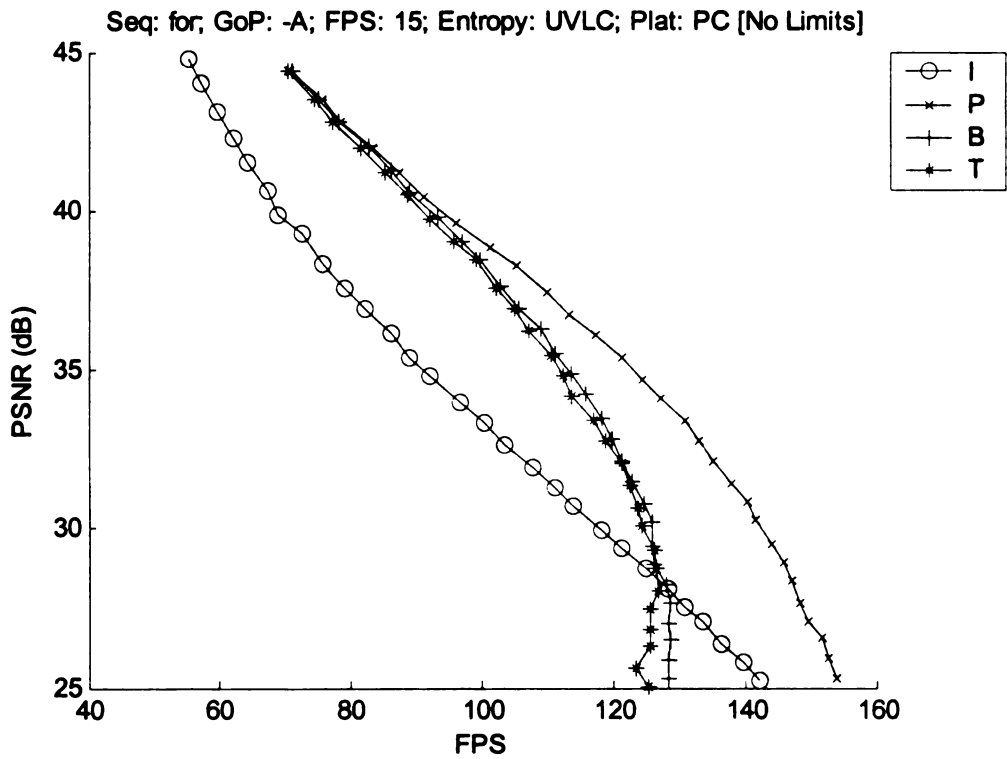
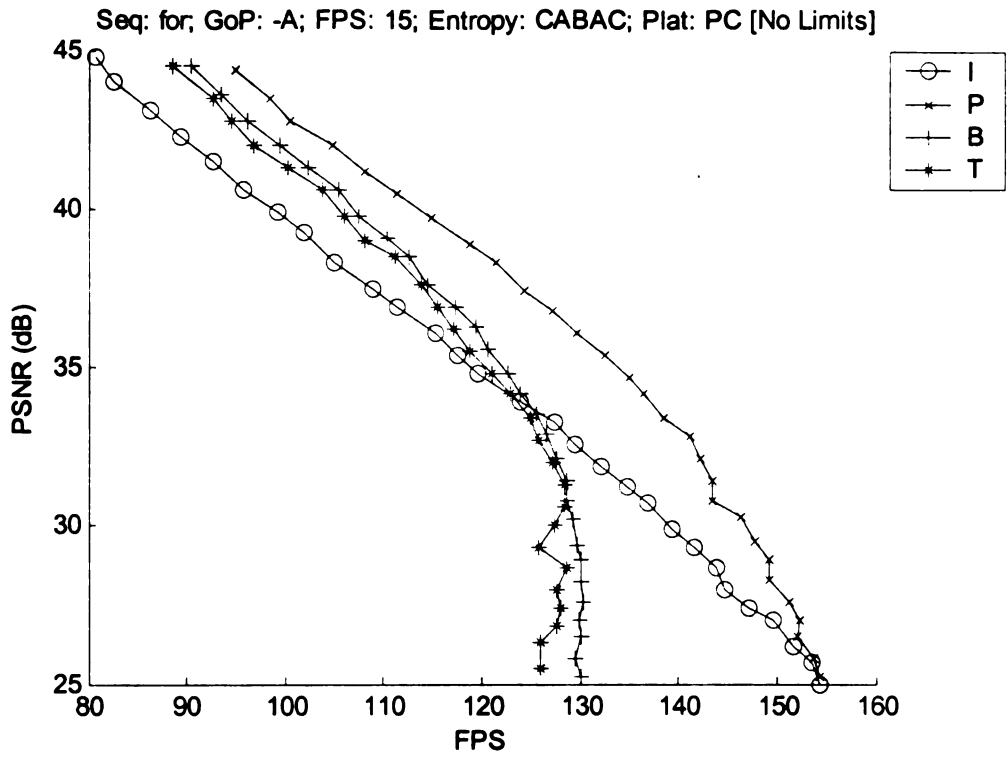


Figure 7: Foreman Complexity-Distortion (15 FPS, PC)

Overview

Foreman Seq:	Time (ms): slowest minus fastest		Percent Slowdown		% Entropy Diff.	
	<i>CABAC</i>	<i>CAVLC</i>	<i>CABAC</i>	<i>CAVLC</i>	<i>25dB</i>	<i>45dB</i>
CE:						
<i>I</i>	24.0	45.9	64.4	101.8	20.7	48.2
<i>P</i>	18.2	44.9	49.2	112.7	7.6	53.4
<i>B</i>	13.1	45.6	26.7	90.4	3.0	54.8
<i>T</i>	12.9	45.8	25.8	86.1	6.4	57.4
PC:						
<i>I</i>	6.0	11.4	92.5	176.8	-0.6	42.9
<i>P</i>	4.1	7.7	63.8	117.1	1.3	34.3
<i>B</i>	3.2	6.5	41.3	82.9	1.6	31.4
<i>T</i>	3.2	6.2	40.0	77.1	1.6	28.6

Table 4: Foreman Complexity Deltas over 25-45 dB

The Foreman sequence results for 15 frames per second are shown in Figure 6, Figure 7, and Table 4. The Foreman sequence was included the test sequences because of its similarity to a “typical” (more-less on average) sequence with reasonable temporal correlation but also containing a moderate-to-significant level of motion. Overall, the Foreman sequence reflects greater complexity compared to the Akiyo sequence, especially as the distortion decreases. Again in the results, like Akiyo, the P-GoP provides the best lowest complexity for a given distortion.

The handheld performance reflects the general knowledge that Foreman is a more complicated sequence than Akiyo was. While both sequences have roughly the same complexity at 25dB, the Foreman complexity increases more rapidly. In addition, the B and T-GoP complexity changes while it remained almost constant for the Akiyo sequence. With reduced compression, the difference between the GoP structures shrinks although a clear difference is still visible. Additionally with low distortion, real time performance can barely be achieved in some instances: The CABAC low distortion I, B,

and T GoPs are near the cutoff point for realtime functionality, and all four GoPs are well below realtime levels using CAVLC.

Handheld: CABAC & CAVLC

Overall when comparing the Akiyo with Foreman GoPs, all four GoPs are similar in speed at the 25dB level. All four also are significantly slower than their Akiyo counterparts at by the 45dB level. By comparing Figure 4 with Figure 6, it is clear that the starting complexity for a high distortion Akiyo sequence are similar to those of the high distortion Foreman sequences, although the Foreman is slightly more complicated even with high distortion. For the Foreman sequences at lower distortion rates, the different GoP patterns come much closer to merging.

Handheld vs. PC

As with Akiyo, the I-GoP once again performs much better compared to other GoPs when using the iPAQ rather than the PC. (The PC response is shown in Figure 5 and Figure 7.) Under CE, the I-GoP outperformed the B and T-GoPs with CABAC and matched them with CAVLC. However, the PC I-GoP has significantly greater complexity compared to the B and T sequences. With Foreman as a less compressible sequence than Akiyo, the complexity increases more sharply for the B and T-GoPs as distortion drops. However, the 25-35dB Foreman range shows the same frozen trend that characterized these GoPs in Akiyo: As the distortion drops, complexity remains almost constant. In fact for 25-31dB portions of the CABAC complexity-distortion graph, a reducing distortion also reduces complexity. It's not immediately clear what is causing this complexity

response and the question will be investigated more deeply with some of the detailed timing data.

Handheld: Akiyo vs. Foreman

Looking at the handheld device, reducing distortion on the Foreman sequence produces a very different result than for Akiyo. The Akiyo sequences begin with substantial time gaps between the P and B GoPs. Under CABAC, the difference is about 13ms at 25dB and is still around 12-13ms at 45dB. For CAVLC, the gap is similar, starting at about 14ms and ending at 12-13ms. However for Foreman, the gap starts around 10ms and shrinking to around 7ms for CABAC. CAVLC, it also shrinks from about 10ms to 7ms. In both sequences, the B and T GoPs mirror each other, with the T GoP being about 1ms slower with CABAC and about 5ms slower with CAVLC coding. Additionally, the complexity increase is significantly greater with the Foreman sequence. For the Akiyo P-GoP, the complexity increases 5ms per frame for CABAC and 13ms per frame for CAVLC. For the Foreman sequence, those numbers are 18ms and 47ms respectively. Thus the Foreman sequence, while having similar performance to Akiyo at 25dB, grows in complexity much more rapidly. This greater overall complexity seems to minimize the difference between I, P, B, and T frames, especially for the CAVLC coding approach.

Mobile

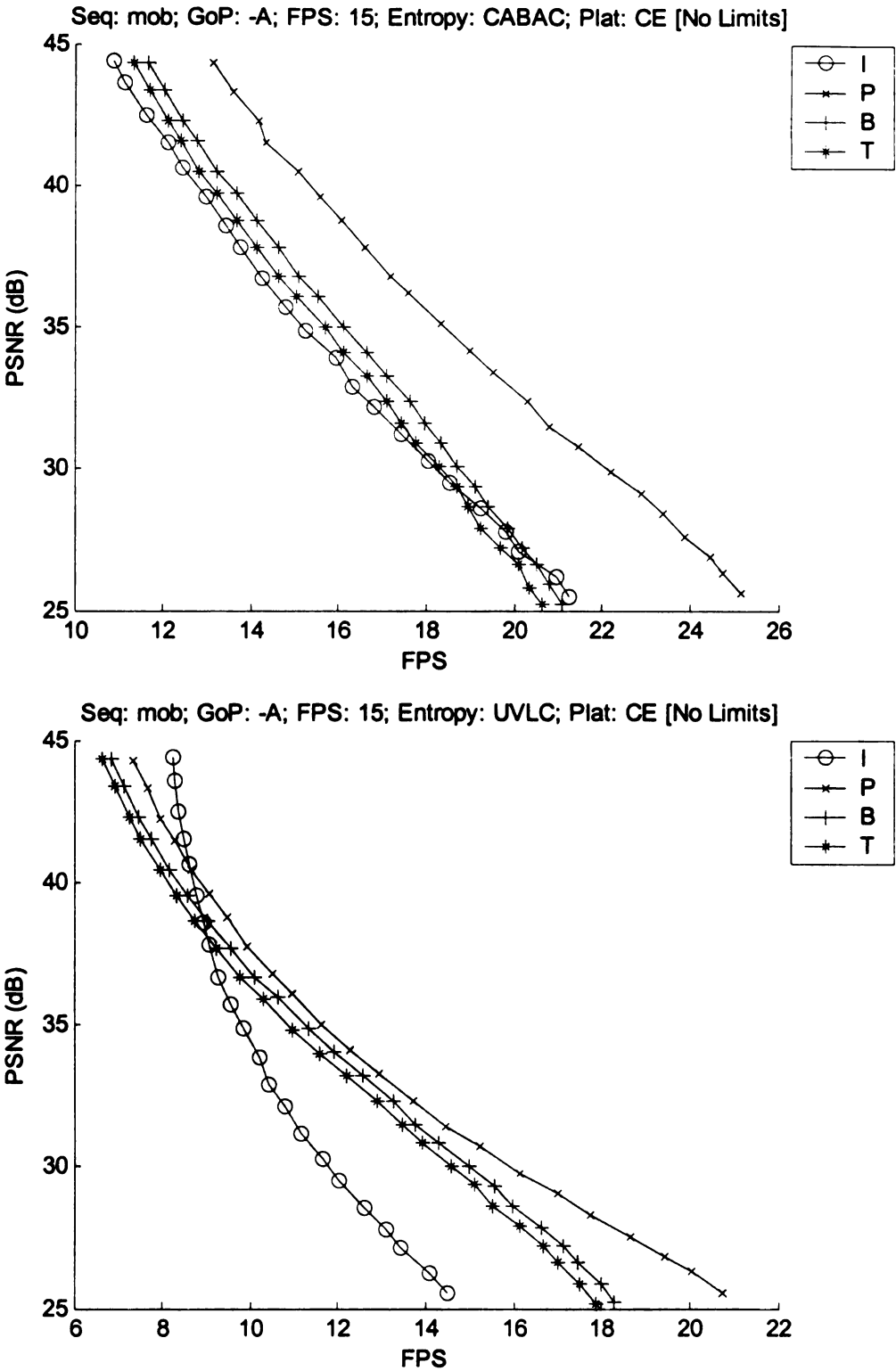


Figure 8: Mobile Complexity-Distortion (15 FPS, CE)

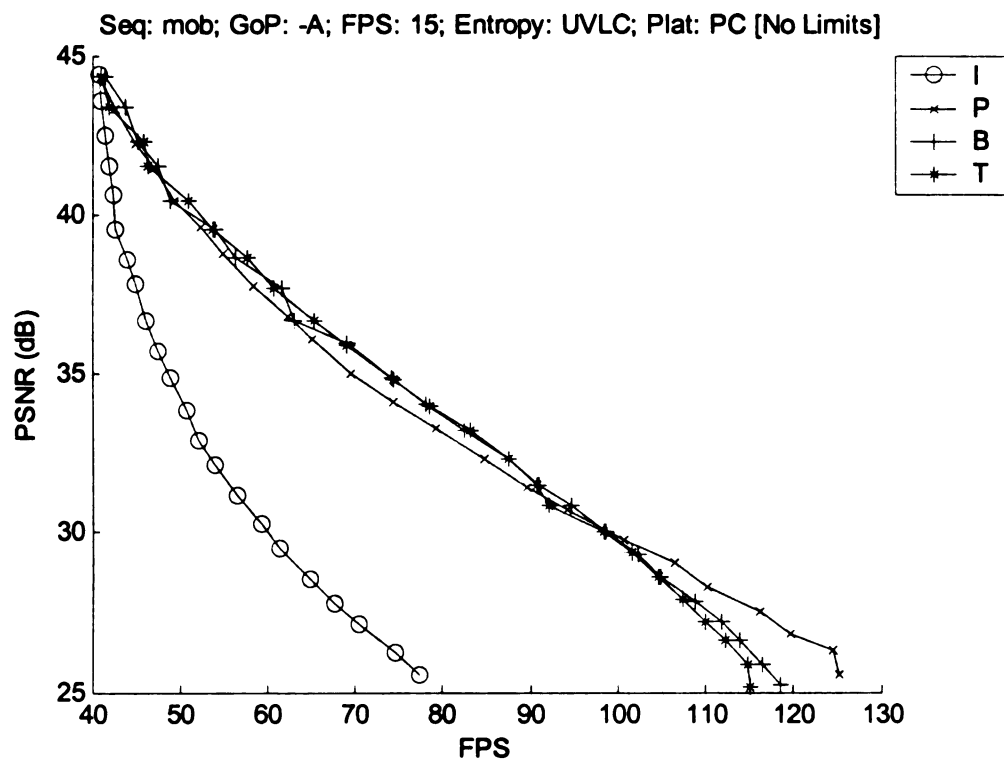
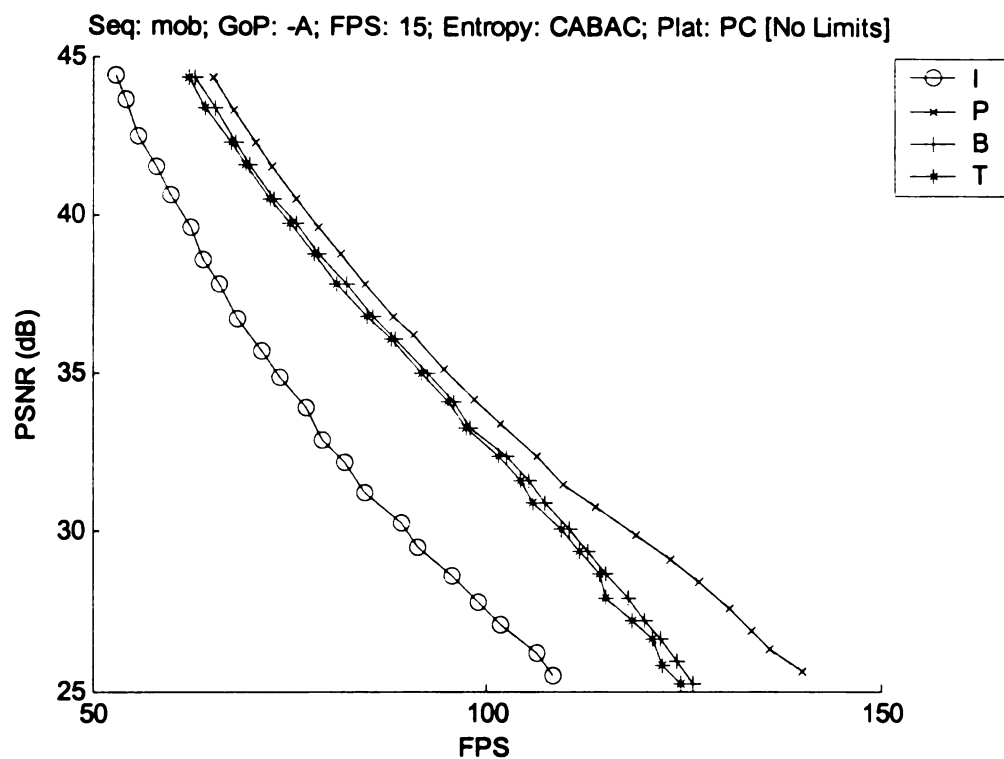


Figure 9: Mobile Complexity-Distortion (15 FPS, PC)

Overview

Mobile Seq:	Time (ms): slowest minus fastest		Percent Slowdown		% Entropy Diff.	
	<i>CABAC</i>	<i>CAVLC</i>	<i>CABAC</i>	<i>CAVLC</i>	<i>25dB</i>	<i>45dB</i>
<i>I</i>	51.2	51.5	129.1	74.7	73.8	32.5
<i>P</i>	30.0	85.0	63.8	176.0	2.9	73.3
<i>B</i>	37.1	87.0	78.0	155.7	17.3	68.6
<i>T</i>	34.6	98.6	70.8	178.5	13.3	84.6
PC:						
<i>I</i>	9.8	11.6	107.5	90.2	41.0	29.3
<i>P</i>	8.7	16.4	122.2	204.9	12.0	53.7
<i>B</i>	8.3	16.0	103.3	190.2	4.2	48.8
<i>T</i>	8.7	15.7	110.0	180.5	9.6	46.3

Table 5: Mobile Complexity Deltas over 25-45 dB

The plots and a tabular summary of results for the Mobile sequence are presented in Figure 8, Figure 9 and Table 5. Again, Mobile is a highly detailed sequence which is difficult to compress and contains a high degree of motion. It is by far the most complicated of the three test sequences used. In fact, real time performance is possible only in select cases: Below about 30dB for CAVLC and below 36 dB for CABAC. This represents a significant complexity increase over Akiyo, where all sequences were real time, and Foreman, where most sequences were real time.

Handheld: CABAC & CAVLC

A key observation is that for the first time, the I-GoP provides the lowest complexity solution for 40-45dB distortion sequences using the CAVLC entropy coding. Although the CAVLC sequences are still slower than their CABAC counterparts, there is no clear

explanation for the stronger performance by the I-GoP. This question will be revisited later when the detailed profiling and rate aspects of these tests are examined.

The 25dB CABAC baseline behavior is also unexpected. The high motion prediction GoPs (B and T) are actually less complicated than their Akiyo and Foreman counterparts. In fact, Akiyo, the most compressible sequence, has the highest complexity of the three at 25dB. However, the I and P GoPs are more complex for Mobile than for the previous sequences. This suggests that with high distortion, the B and T GoPs are fairly insensitive to correlation, while the P and especially the I GoP, without temporal correlation, are more sensitive to sequence complexity.

Handheld vs. PC

Surprisingly, the overall pattern is that as distortion decreases, the handheld complexity increases more slowly than the PC complexity. For example, the net CABAC handheld complexity increase for the P-GoP is 64%, but a shocking 122% for the PC. The exception to this pattern is the CABAC I-GoP where the PC increase is 108%, the handheld 129%. As noted before, the I-GoP is the least predictable GoP structure. The overall lower complexity increase for the handheld, however, suggests that the iPAQ is less sensitive to some algorithmic aspect related to distortion changes than the PC.

Mobile Compared Against Previous Sequences

The difficult Mobile sequence also produces a more linear performance from the CABAC sequences than has been seen before: The relative performance between the four GoPs is similar regardless of the distortion, although the T and I GoPs gradually slip further

behind the B GoPs performance. (This linear nature is not seen as strongly in the desktop, where the I-GoP is significantly slower than the others.) The CAVLC sequences have similar response for the P, B, and T GoPs while the I-GoP starts out much more complex but grows in complexity more slowly than the other GoPs. This is a reversal of the Foreman pattern where the I, B, and T-GoPs clustered and the P-GoP was less complex

The Mobile sequence also reflects a much greater complexity increase as the distortion decreases. Focusing on the handheld as the primary experiments of interest, the 45dB sequences grow in complexity roughly twice what the Foreman sequences did. For Foreman, the CABAC slowdown percentages for the I, P, B, and T GoPs were 64%, 50%, 27%, and 26% respectively, and 102%, 112%, 90%, and 86% for CAVLC. For the Mobile sequences, the equivalent CABAC numbers are 130%, 64%, 78%, and 81%; for CAVLC 75%, 176%, 156%, and 179%. While the CAVLC I-Gop and the CABAC P-GoP do not reflect the same slowdown rates, the overall pattern is that Mobile slows down significantly more compared to the baseline than Foreman (as well as Akiyo, which experienced minimal complexity increase).

Trends and Observations

Several distinct patterns show themselves through all three sequences.

- 1. First, the different GoPs do show repeatable patterns. The P-GoP consistently provides the lowest complexity for almost any level of distortion.**

The B and T GoPs mirror each other's performance with the double B frame always being a bit slower than the single B frame. Presumably if motion prediction and thus B frames were significantly faster, the increase in T frames

would result in slightly faster performance compared to the B-GoP. The I-GoP is the most unpredictable of the four GoPs, ranging from much slower than any other GoP to being the least complex choice for high quality CAVLC coded Mobile sequences. However, significant improvement in the performance of B frames is clearly needed before the B or T-GoPs become a viable option. The I-GoP, with special attention to parsing, may begin to outperform the P-GoP for specific sequence choices.

2. **Secondly, the results repeatedly show that Pentium simulations do not provide accurate predictions for handheld behavior.** At times, the results are very similar (e.g. CAVLC Akiyo) while at other points the slowest PC option is the best handheld option (e.g. CAVLC Mobile). In particular, this unpredictability is demonstrated by the I-GoP, which varies drastically in size and complexity based on the sequence and distortion level. From the high level results, the B and T-GoPs are very similar, with the T-GoP being just a bit slower than the B-GoP on both platforms. Beyond that, while similarities exist, such as the strong P-GoP performance, the similarities differ for each sequence and entropy coding selection. These differences suggest that algorithmic optimization at a language level may be inadequate to fully address the complexity issues.
3. The lack of correlation between experiments on the PC and handheld platforms, despite compiler optimization for both, suggests that non-algorithmic aspects such as chip architecture, operating system, and compiler techniques play a significant

role in the overall complexity of a given sequence. This disconnect emphasizes the need for multiple platform complexity studies as well as better analysis tools for non-traditional platforms.⁸

4. There is strong correlation between the compressibility of a sequence and its general complexity, both on the handheld and standard computer. Akiyo is the fastest sequence, followed by Foreman and finally by Mobile. This correlation is reflected regardless of platform, entropy technique, or Group of Picture structure. The complexity-compression correlation is most visible in the lower distortion sequences. Overall, the high distortion sequences of the same GoP structure have similar complexities; as distortion drops, the correlation becomes more distinct.
5. The complexity increase (as distortion drops) is significantly higher with CAVLC entropy coding than with CABAC. This suggests the parsing/entropy decoding method significantly influences the overall complexity. This theme will be explored more in depth in Chapter 4.

⁸ As of this writing (April 2003), Intel has just released a profiling tool for the PXA2xx processors (the processor series in the iPAQ); however the tool requires that the processor be used with Intel's development hardware and does not work with other hardware.

CHAPTER 4

BITRATE COMPLEXITY ANALYSIS

In analyzing the distortion-complexity plots shown in Chapter 3, the complexity variation is strikingly different based upon sequence choice as well as GoP type. In order to better understand the factors controlling the complexity result, a number of high level timing probes were inserted into the code. While profiling programs are available to profile code built for traditional computers, equivalent software is not available for most mobile platforms, including the iPAQ. Thus a creative and non-intrusive approach to timing is necessary for understanding the program performance. As mentioned earlier, the iPAQ timings do not act as a miniature PC results and thus profiling the PC is not an entirely accurate substitute.

The remainder of the chapter is organized as follows. First, the timing probes used to measure the program structure are discussed. Then the timing results for the primary four GoP-structures are presented by sequence. Finally, the chapter concludes with an overall summary. For normalization purposes, results are generally presented by milliseconds per frame. For the fifteen frames per second sequences, 60ms per frame or better is required to achieve realtime. The 60ms to decode leaves the video player with roughly 7ms to display each frame, which is the approximate display rate.

Timing Probes

A total of seven high precision *timing probes* were inserted into the decoding software. These probes were intentionally placed in the infrequently called functions in order to minimize the impact of the actual complexity impact of the probes. Initial tests indicated

that that any complexity change was well within the decoder's standard variance. Special attention has been given to the measurement descriptions to note "what is" and "what is not" included in each probe. Probes are called a total of roughly 200 times per frame.

Three estimates were derived for a probe time. The first, by measuring the time before and after a single probe, generated a worst case number. The second, by timing many probes and then averaging for a per probe time, generated a more likely estimate but ignored any advantage being gained from cache coherency. Neither estimate included the benefits that interleaving non-probe instructions could have on the CPU pipeline and memory. The final estimate was achieved by comparing pre-timing decoder runs with timer implemented runs. The estimated worst case time for a probe in this iPAQ environment is 40 ticks, with 3.69 million ticks occurring per second. For the fifteen per second frames, that translates into an addition of roughly one-third of a second to the total time. The averaged estimate, closer to 13 ticks, adds one-tenth of a second. Actual experimentation noticed no significant difference based on timing implementation beyond the standard variance. This observation suggests that the actual time change was under one-tenth of a second for most runs.

Actual Timing Probes

Equation 2 shows the relationship between each of the probes used, as well as the four additional calculated times. Each of the specific probes is discussed at greater depth below.

$$\begin{aligned}
T_{\text{Tot}} &= T_{\text{Over}} + \sum_{\text{I,P,B Frames}} T_{\text{DecFr}} \\
T_{\text{DecFr}} &= T_{\text{ReadNewFr}} + T_{\text{FrInit}} + T_{\text{ProcMB}} + T_{\text{PostFrDec}} + T_{\text{DeblockFr}} + T_{\text{Unk}} \\
T_{\text{ProcMB}} &= \sum_{\text{All MB}} (T_{\text{ReadMB}} + T_{\text{DecodeMB}}) + T_{\text{MBOth}} \\
T_{\text{Non-ReadMB}} &= T_{\text{ProcMB}} - \sum_{\text{All MB}} (T_{\text{ReadMB}})
\end{aligned}$$

Calculated, not timed, probes: $T_{\text{Unk}}, T_{\text{Over}}, T_{\text{MBOth}}, T_{\text{Non-ReadMB}}$

Equation 2: Timing Probe Interactions

Total Time (Tot): This is the total time spent decoding a complete sequence (e.g. approximately 150 frames for 15fps⁹), excluding decoder initialization. Any comparison to the total decoder time is against this number. There is one call to this timer for the whole program.

Decode Frame (DecFr): The time to decode a frame, ignoring a slight amount of memory manipulation and generic cleanup (calculated below as Overhead). The frame times are totaled by frame type (I, P, or B), and are used for any calculation involving the picture type total time calculations. The sum of all three of these picture type totals is slightly less than the total time, due to the overhead.

Process Macroblocks (ProcMB)¹⁰: This is the total time it takes to decode a frame, including reading the pre-macroblock information, reading macroblocks, and decoding them. It ignores deblocking time (measured elsewhere) as well as some frame cleanup.

⁹ All original sequences have 300 frames so a perfect 15fps sequence is 150 frames. However due to the GoP structure, some sequences end at 148 or 149 frames because the next frame would be a B-frame but it has no future I or P-frame to use as a reference frame. The data presented is normalized by frames to compensate.

The probe is called once per frame and this section is expected to occupy a significant portion of the total run time.

Pre-Decode Frame Init (FrInit): This probe measures the time to prepare for decoding a frame. It includes initializing frame flags, entropy settings, and error correction as well as ordering the frame buffers. This timing section, called once per frame, is expected to take very small percentage of the program.

Read New Frame (ReadNewFr): This probe measures the time to read and process the pre-macroblock flow control information for each frame. This time is expected to be minimal and the probe is called once per frame.

Read Macroblock (ReadMB): This probe, called once per macroblock (ninety-nine times per QCIF frame) measures the amount of time it takes to read a macroblock, primarily entropy decoding. However, the decoder also uses an inverse transform on intra-coded (“I”) 16x16 macroblocks to extrapolate the DC coefficients for the new intra prediction modes. On selected sequences that were profiled for function counts, this rare transform was called less than a thousand times (compared to over 300,000 calls to the transform in DecodeMB).

Decode Macroblock (DecodeMB): The time spent taking the differential coefficients and motion coefficients read during the Read Macroblock phase to find motion estimate and calculate the new macroblock coefficients using the motion vectors. Like read macroblock, the probe is called once per macroblock.

Post-Decode Frame (PostFrDec): This probe measures the clean-up time after successfully reading and decoding the macroblocks in a frame. The time here includes

¹⁰ This probe section corresponds to the “slice” breakdown within the reference code. Thus some of the

resetting various flags, motion information copying, and motion vector storage. This time is expected to be negligible but is included for completeness. The probe is called once per frame.

Deblock Frame (DeblockFr): This probe measures the time taken to deblock the frame (apply the deblocking filters to the decode image and remove some of the artifacts and blockiness). This probe is called once per frame, and deblocking is expected to be a significant task.

Calculated Timing Probes

The following additional measurements were calculated from the previous probes to provide additional information about the timing within decoder. All of the timings below use the previous nine timings as their foundation.

ProcessMB excluding Read Macroblock (Non-ReadMB): This is the time it took to decode the macroblocks excluding the time spent reading macroblocks. This time includes ProcessMB Other (see below) as well as Decode Macroblock.

Process Macroblocks Other (MBOth): The difference between the total time to decode all macroblocks and the measured subparts of macroblock decoding. The two subparts are read macroblock and decode macroblock. Those two times were added together and then subtracted from the total macroblock decoding time.

Overhead (Over): The difference between the total time spent decoding all of the frames and the total time spent decoding, ignoring decoder initialization. Unlike other measures,

graphs reference this as a slice rather than ProcessMB. The discussion uses ProcessMB for clarity.

overhead is calculated as picture type independent because only a single total time is available. Thus there is no measure of overhead for different picture types.

Unknown (Unk): The time spent decoding a frame but not spent in any known subsection.

This time is calculating by summing the following parts: frame deblocking, process macroblocks, read new frame, post frame decoding, and pre-decode frame init. The sum is then subtracted from the total time spent decoding frames. (Thus overhead is not included as unknown time.)

Akiyo

This section presents the timing figures for the timing sections that show significant variation in complexity based upon bitrate for the Akiyo sequence. The overall rate-complexity plots are shown in Figure 10. The bottom plot is the magnified section containing the non-I-GoP from the top plot¹¹.

Many of the probes, shown in Table 6 indicate that bitrate has an insignificant impact upon their performance. The slight variances shown are neither linear nor particularly related to time. In all cases except for post frame decode, they include the timings for all four GoP structures (I, P, B, and T), and the variance for an individual GoP type is much smaller. Over a range of 0 to 35 kilobits per frame, the total variance is only 2.3ms for CABAC and 3.4ms for CAVLC (due to read new frame having a higher variance).

¹¹ As mentioned before, the rate limit in the title is for the average size of all frames, not for the average size of specific frame type. The rate axis for the plots corresponds to either the average size of all frames ('-A') or of a specific frame ('-i', '-p', '-b') as determined by the plot type.

In comparison, the deblocking function is clearly rate related¹². The deblocking rate-complexity plots are shown in Figure 11, and then broken down by picture type in Figure 12. The bottom section of Figure 11 shows the magnified view of the non-I GoPs. The top half of Figure 12 looks at the I frames from all four GoPs, the bottom half looks at the P frames from the non-I-GoPs. Figure 11 suggests the linear performance clearly illustrated by Figure 12. The key point for the Figure 12 I frames is how similar the results are for all four GoPs: Regardless of how many I frames were used in a sequence, the deblocking time per kilobit remains constant. The B frames, although not shown, also follow the P pattern of increasing complexity with rate. CAVLC plots match, also not shown, form mirror images for the figures shown. This result is expected since entropy coding should not have any effect on the decoder outside of entropy reading.

Presented in Figure 13 are the complexity-rate results for read macroblock. Comparing Figure 13 with Figure 10, it is clear that much of the decoding complexity gain comes from the ReadMB, or parsing, portion of the decoder. The decoder increase is 14ms, the parsing increase 15ms. The P-GoP is 6ms vs. 4ms; the B-GoP 3.5ms vs. 4ms. The difference is even more noticeable in Figure 14, which shows the CAVLC ReadMB timings over the 0-7 kilobits per frame range. For the four GoPs (I, P, B, T), the total time increases are 12ms, 6ms, 7ms, and 7ms. The correlated ReadMB times are 6ms, 10ms, 11ms, and 11ms. Since the deblock frame probe decreases with increased rate for the I-GoP, the I-GoP total difference is somewhat less than the ReadMB increase. Likewise, since Deblock Frame increases with complexity for P, B, and T, the difference is

¹² The relative complexity change compared to time is rather small, and mostly likely is related to how many special cases the deblocker is dealing with, which allow the deblocker to do fewer calculations.

somewhat larger than purely the parsing increase. However in each case, the parsing accounts for over half of the net increase in complexity.

Summary

The timing analysis of the Akiyo decoding rate-complexity suggests several trends. First, that many sections of the decoding process are rate invariant: Frame reading, initialization, other MB decoding, post frame decode, overhead, and unknown time. (Post frame decoding, does however, have a very specific GoP based complexity.) Secondly, parsing and deblocking complexity are both linked to rate, but that parsing has almost a linear correlation to bitrate. The parsing complexity increase corresponds strongly to the net complexity increase in the decoder. Finally, the analysis suggests, as expected, that entropy coding has no effect on the decoding complexity outside of the parsing process.

Akiyo Tables and Figures

Akiyo CE Seq.	Time Range (ms)		Time Variance (ms)	Average Time (ms)
	<i>Fastest</i>	<i>Slowest</i>		
CABAC:				
<i>Read New Frame</i>	1.1	2	0.9	1.6
<i>Overhead</i>	0.25	0.63	0.38	0.45
<i>Unknown</i>	0.01	0.18	0.17	0.1
<i>Decode Frame Init</i>	3.87	4.1	0.23	3.97
<i>Process MB Other</i>	2.9	3.4	0.5	3.1
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.4; P: 2.5 B: 1.3; T: 0.8
CAVLC:				
<i>Read New Frame</i>	1.0	2.7	1.7	1.5
<i>Overhead</i>	0.2	0.6	0.4	0.4
<i>Unknown</i>	0.01	0.21	0.2	0.125
<i>Decode Frame Init</i>	3.67	3.87	0.2	3.75
<i>Process MB Other</i>	2.88	3.55	0.77	3.30
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.4; P: 2.5 B: 1.3; T: 0.8

Table 6: Akiyo Rate Invariant Probes (15fps, CE)

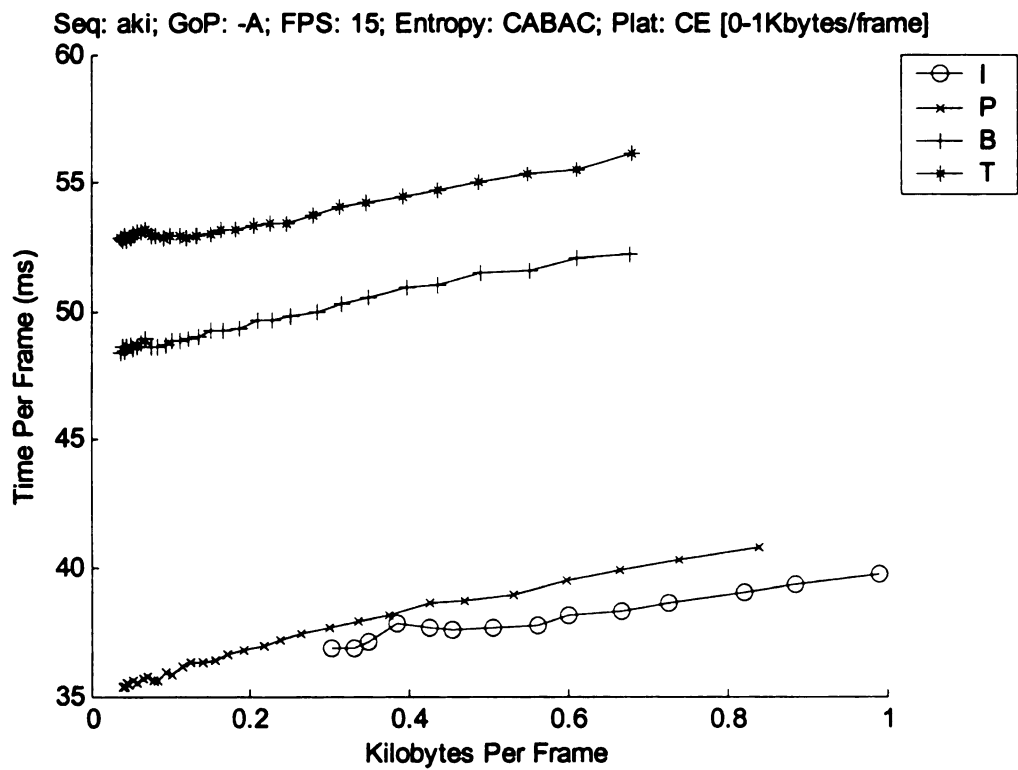
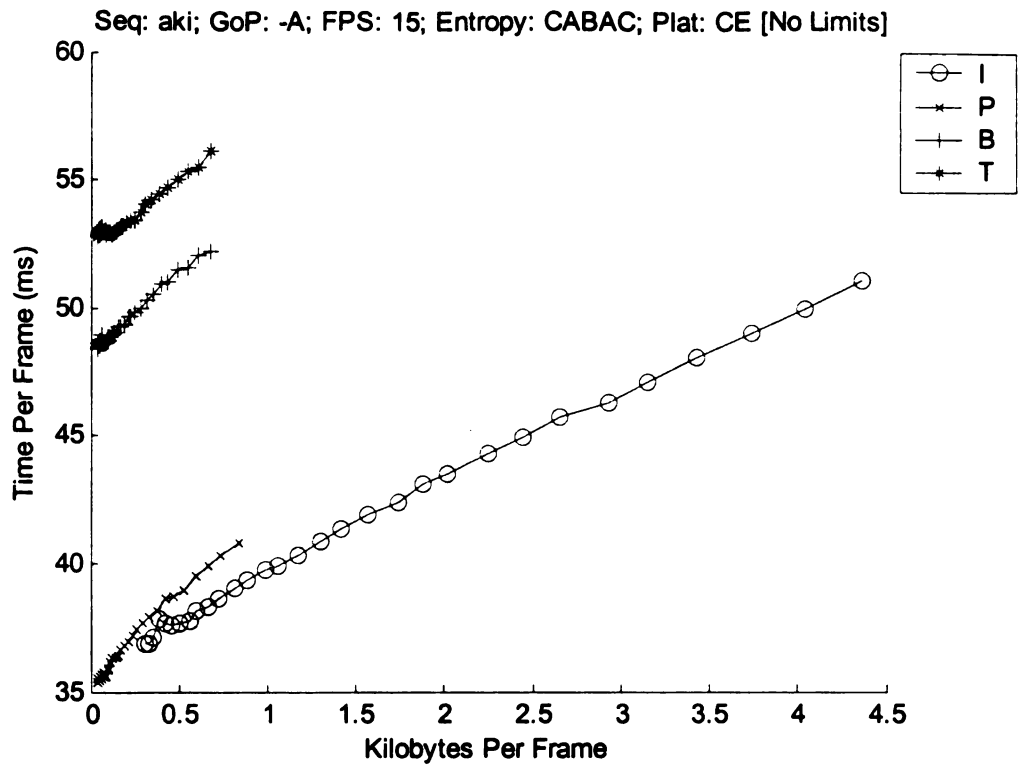


Figure 10: Akiyo Rate-Complexity (15 FPS, CE)

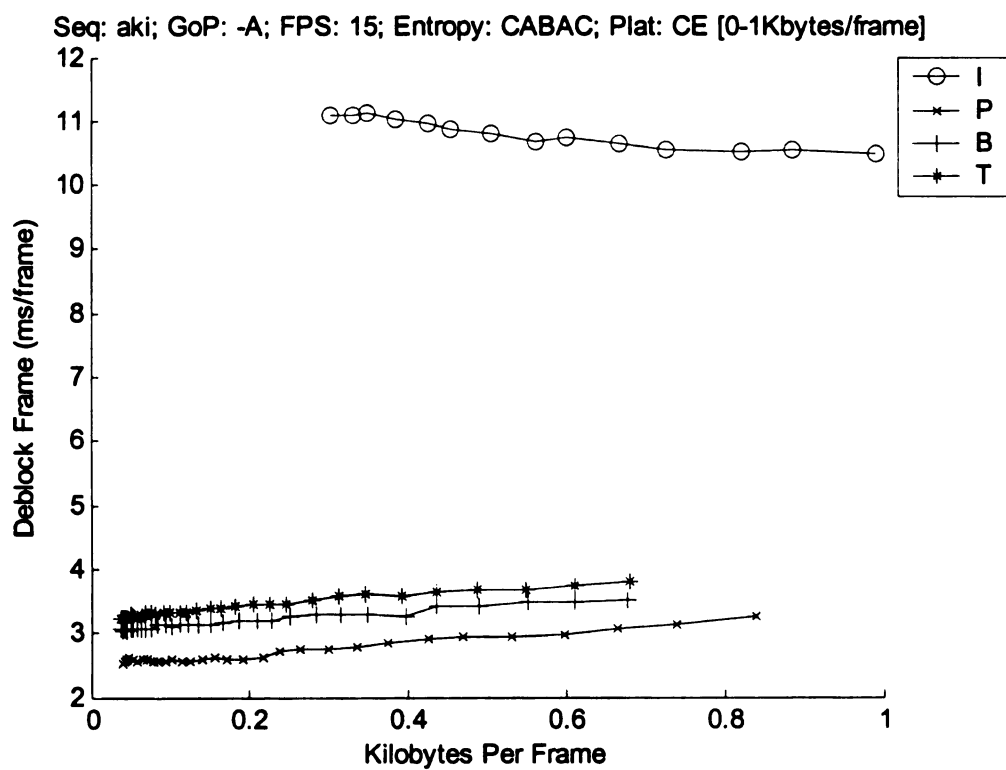
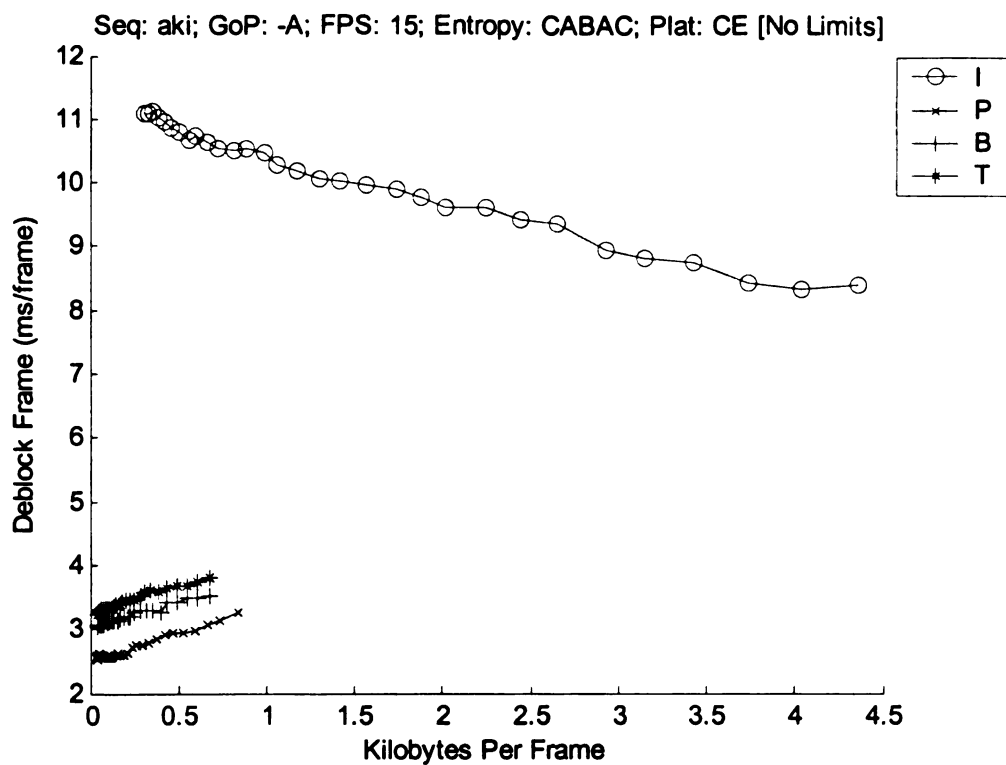


Figure 11: Akiyo Deblock Rate-Complexity (15 FPS, CE)

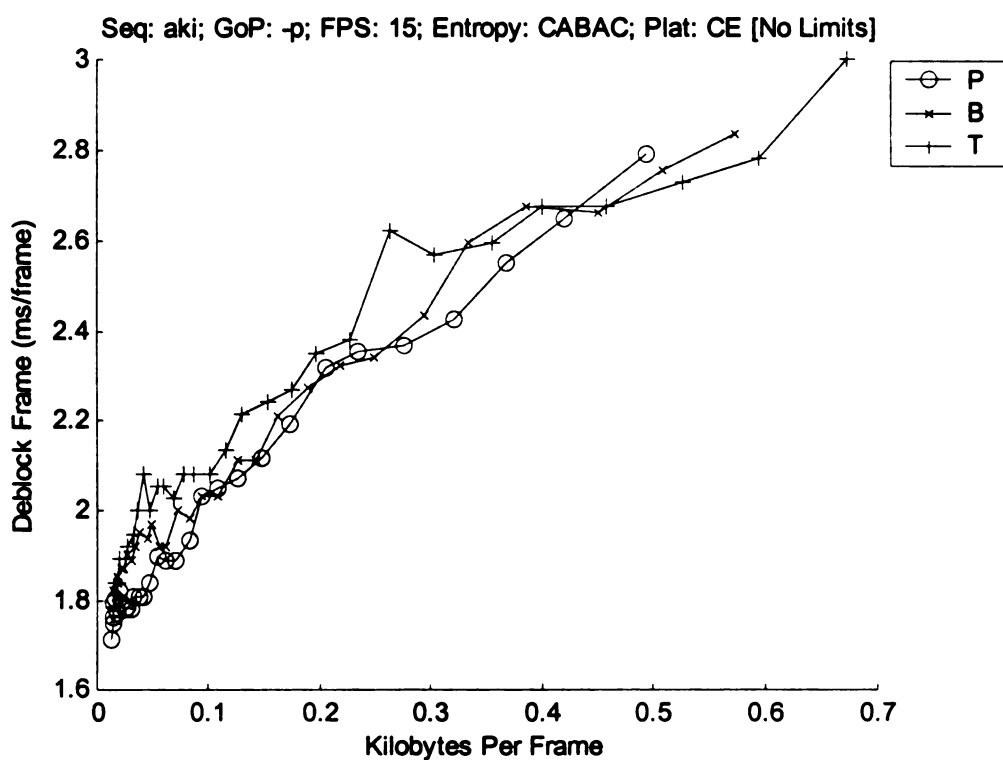
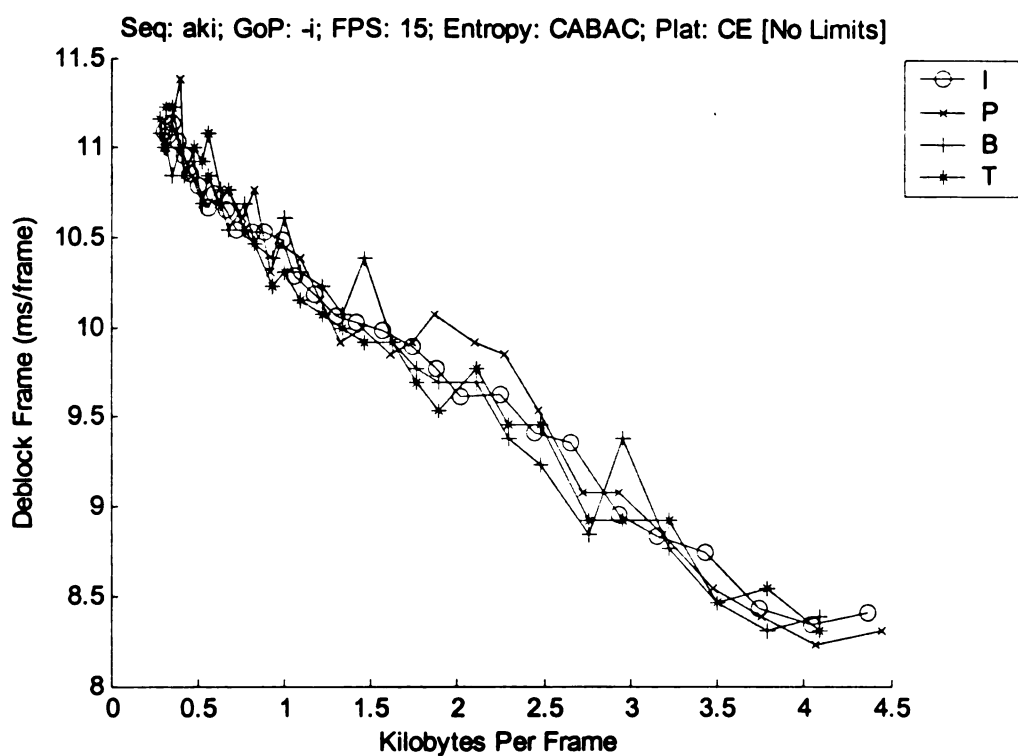


Figure 12: Akiyo Deblock Rate-Complexity by Picture Type (15 FPS, CE)

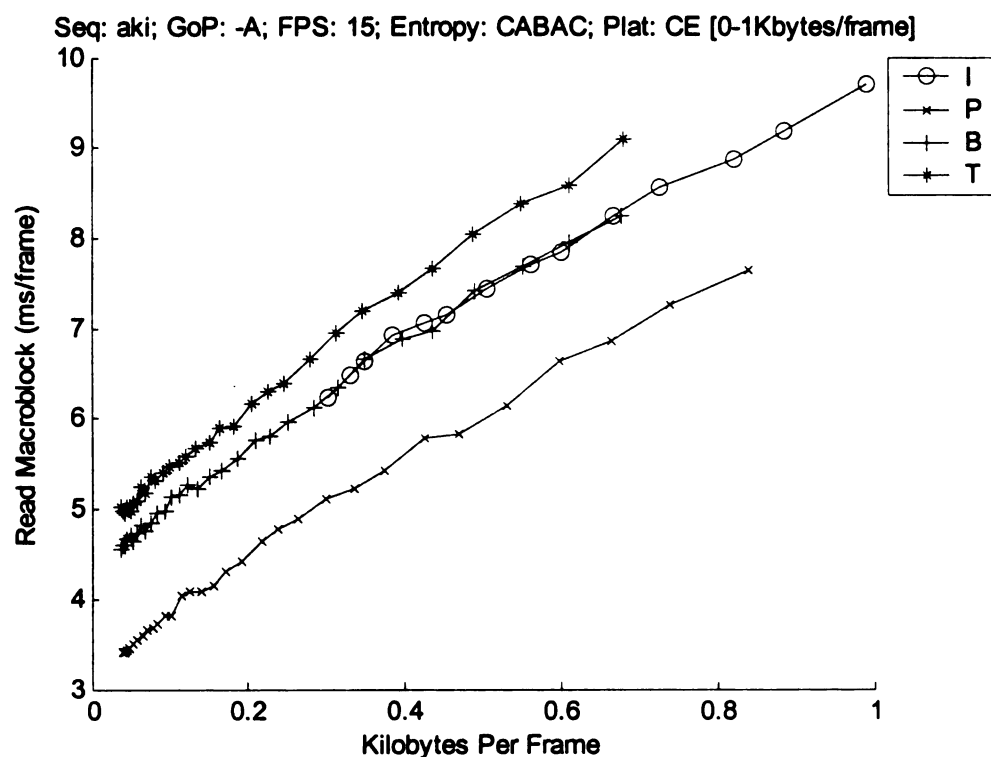
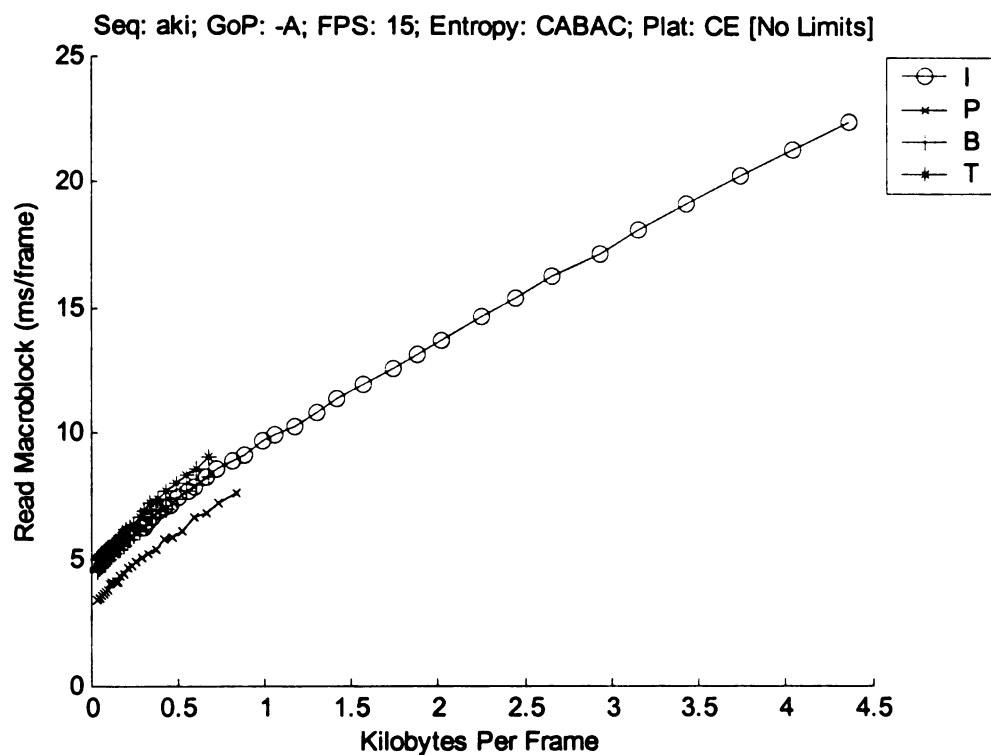


Figure 13: Akiyo ReadMB Rate-Complexity (15 FPS, CE, CABAC)

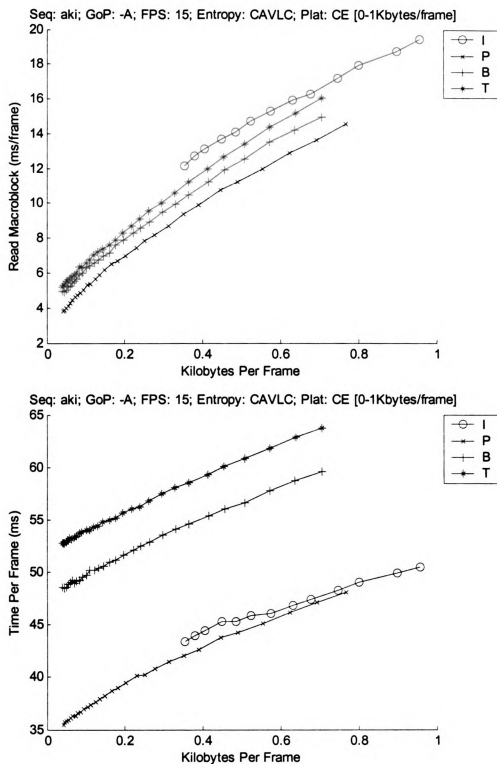


Figure 14: Akiyo ReadMB Rate-Complexity (15 FPS, CE, CAVLC)

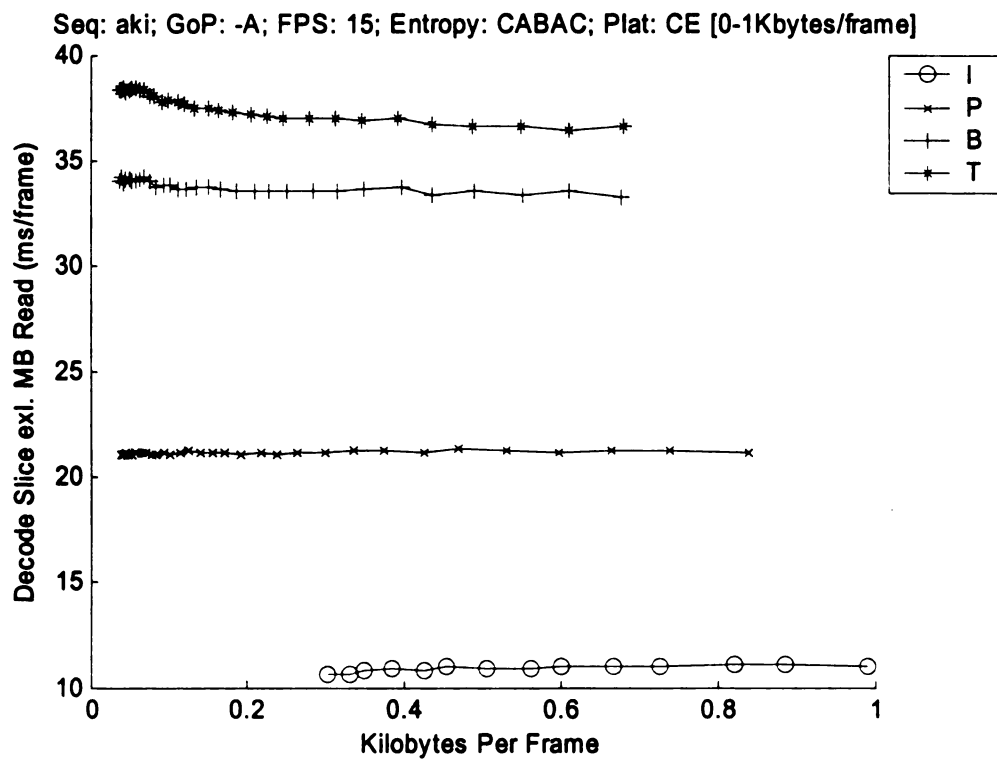
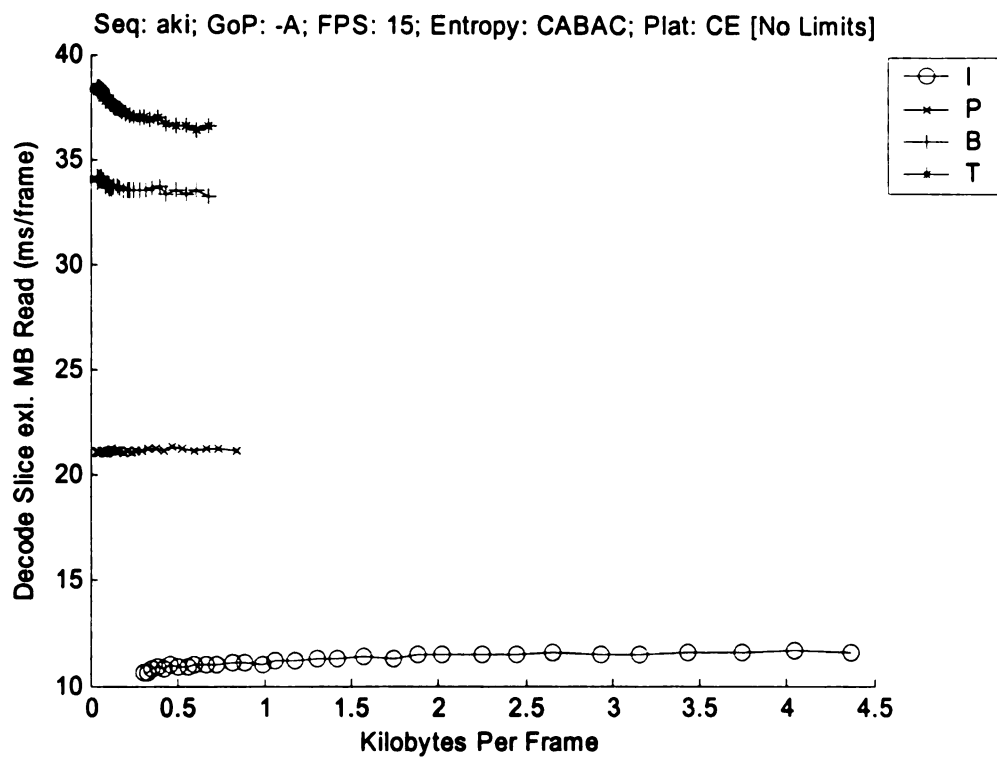


Figure 15: Akiyo ProcessMB excl. ReadMB Rate-Complexity (15 FPS, CE, CABAC)

Foreman

Foreman, despite being a more complex sequence, exhibits the same probe invariants.

Table 7 shows the same probe invariants noted for the Akiyo sequence. The range, variance, and average times are almost unchanged when compared to the Akiyo table.

This table further suggests that deblocking, decoding, and parsing contribute most of the complexity change to the decoder.

Figure 16 presents the rate-complexity plots for the Foreman sequence, once again with a magnified look at P, B, and T GoPs. For Akiyo with these three GoPs, the 45dB distortion was about 7Kb per frame. With Foreman, it is between 25 and 30Kb. The figure also emphasizes the sharp drop in complexity for the B and T GoPs at low bitrates. This drop will be discussed more when the ProcessMB decoding probe is discussed.

Figure 17 shows the blocking rate complexity for the CABAC sequences. The CAVLC are almost identical to those in the figure, as was true for the Akiyo sequence.

Deblocking remains a fairly rate independent task, although clearly it is rate dependent in to a small degree. Compared to the Akiyo sequence, the I-GoP rates are within a quarter of a second of each other. However, for the same rate, the other three GoPs are roughly 1ms more complex for the Foreman sequence. This difference suggests that deblocking is slightly sequence dependent. Figure 18 presents the deblocking complexity by picture type for the I and P frames. Like the Akiyo, the I frames take the same time to deblock, regardless of the sequence type they are used in. However, the P frames vary in

deblocking time by the GoP, especially at lower rates. In addition, while the Akiyo I frame deblock rate is roughly equivalent to the Foreman one, the P frames more complex it deblock, regardless of the GoP.

Figure 19 presents the parsing complexities for the Foreman sequence. Again, the parsing complexity is roughly linear, although the slope depends on the GoP structure. The I frame parsing rates are exactly the same as the Akiyo rate: All of the points from both sequences fall on a single line for each GoP. The P and B frames reflect the same linear trend, although the Foreman sequence is about 0.5ms (10%) more complex for both frame types. As with Akiyo, Figure 19 clearly shows the linear relationship between rate and complexity. Figure 20 shows the same linear pattern for the CAVLC parsing speed.

As with Akiyo, the parsing complexity increase represents a significant contribution to the overall complexity increase. For example, the CABAC I-GoP increases in complexity by 24ms, the parsing increase contributes 28ms to that time. Other decreases contribute the -4ms. For the CAVLC P-GoP, the complexity increase is 35ms, and the parsing contribution is over 30ms. With the more complicated Foreman sequence, less compression, and longer frames, the proportion of time spent parsing grows. With Akiyo, the largest parsing change was on the order of 10ms, with Foreman it is four times that number.

Figure 21 shows that the non-parsing complexity is steady, with the exception of a drastic complexity shift for the B and T GoPs between 0 and 5 Kb per frame. This shift explains

the unusual drop in overall complexity in the Foreman results, although the coding mechanism change responsible for the drop is unknown. It's worth noting that the actual decoding times, excluding parsing, are the same between the Foreman and Akiyo I and P-GoPs. The Foreman B and T-GoPs actually have lower non-parsing decode time than Akiyo. One possibility for this difference is that Akiyo, with higher compressibility, can make better use of the motion vectors which then require longer to decode; the weaker Foreman compression may simply decode more quickly. However, even excluding parsing, the I GoP still decodes twice as fast as the P-GoP, and 2.5 times faster than the B and T GoPs

Summary

The Foreman sequence, with a broader bitrate range than the Akiyo sequence, maintains very similar baseline decoding values aside from parsing. Despite being a more motion intensive sequence, the Foreman B and T GoPs actually decoded slightly faster than Akiyo when parsing time was ignored. The rate invariant sections carried over from the Akiyo sequence. Parsing contributed to most of the complexity gain, and more clearly than the Akiyo sequence because of the wider bitrate range.

Foreman Tables and Figures

Foreman CE Seq.	Time Range (ms)		Time Variance (ms)	Average Time (ms)
CABAC:	<i>Fastest</i>	<i>Slowest</i>		
<i>Read New Frame</i>	1.1	2.1	1.0	1.6
<i>Overhead</i>	0.24	0.62	0.38	0.45
<i>Unknown</i>	0.01	0.23	0.22	0.1
<i>Decode Frame Init</i>	3.87	4.15	0.28	3.97
<i>Process MB Other</i>	2.9	3.75	0.85	3.2
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.4; P: 2.5 B: 1.3; T: 0.8
CAVLC:				
<i>Read New Frame</i>	1.2	2.1	1.7	0.9
<i>Overhead</i>	0.2	0.63	0.43	0.45
<i>Unknown</i>	0.02	0.2	0.18	0.12
<i>Decode Frame Init</i>	3.67	3.85	0.18	3.75
<i>Process MB Other</i>	3.1	4.9	1.8	3.80
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.3; P: 2.4 B: 1.2; T: 0.8

Table 7: Foreman Rate Invariant Probes (15fps, CE)

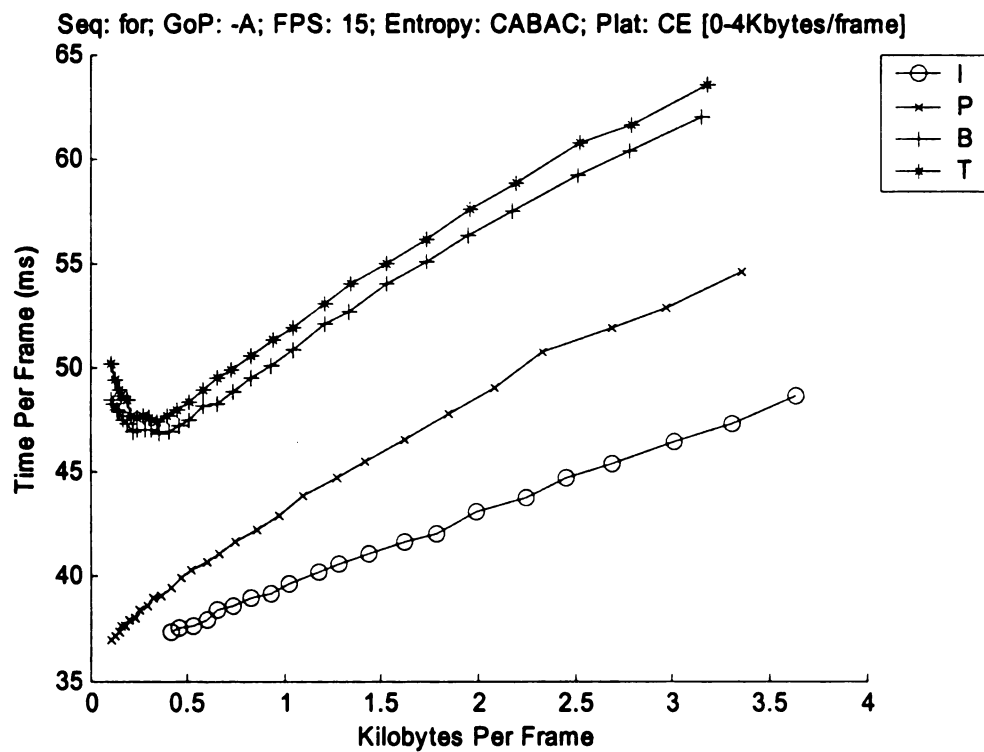
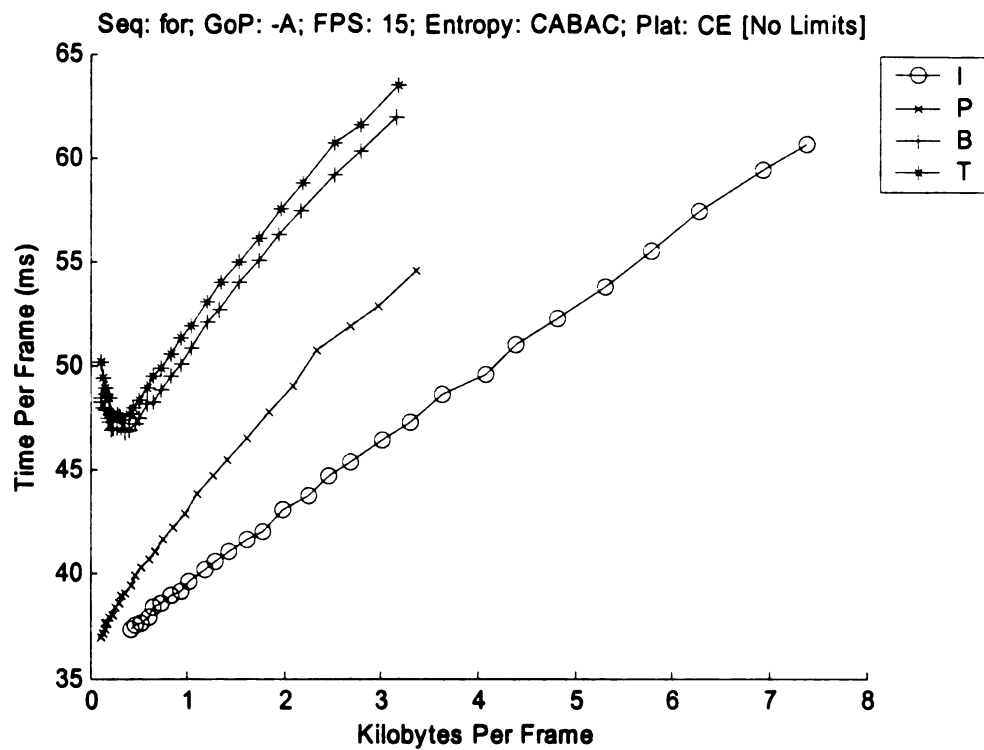


Figure 16: Foreman Rate-Complexity (15 FPS, CE)

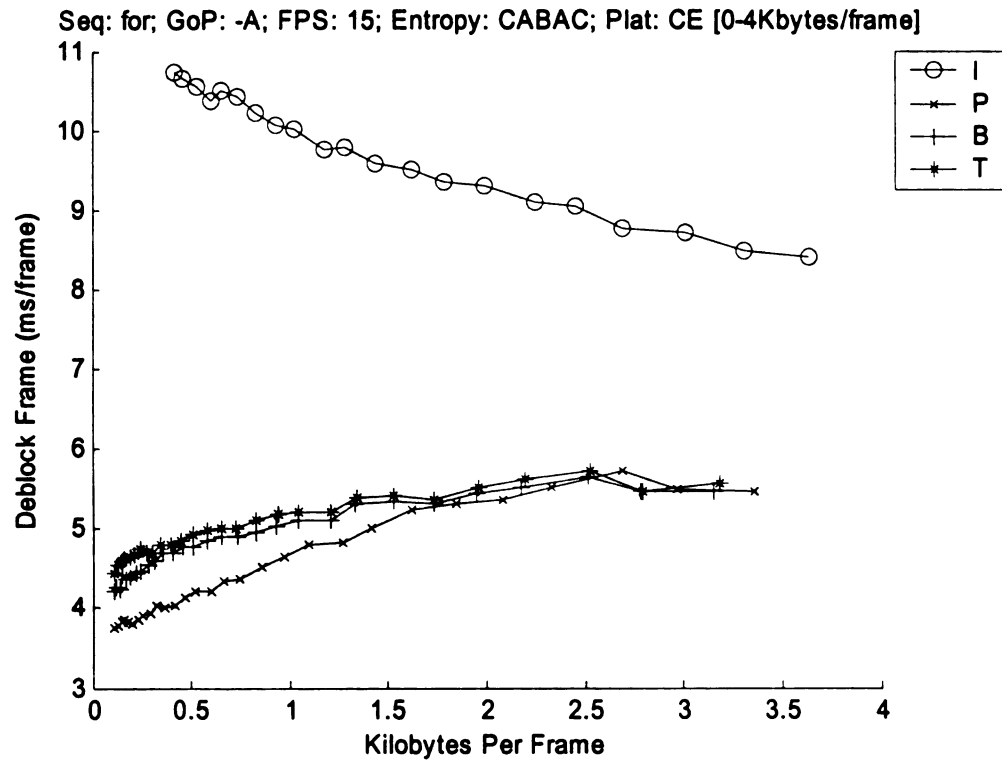
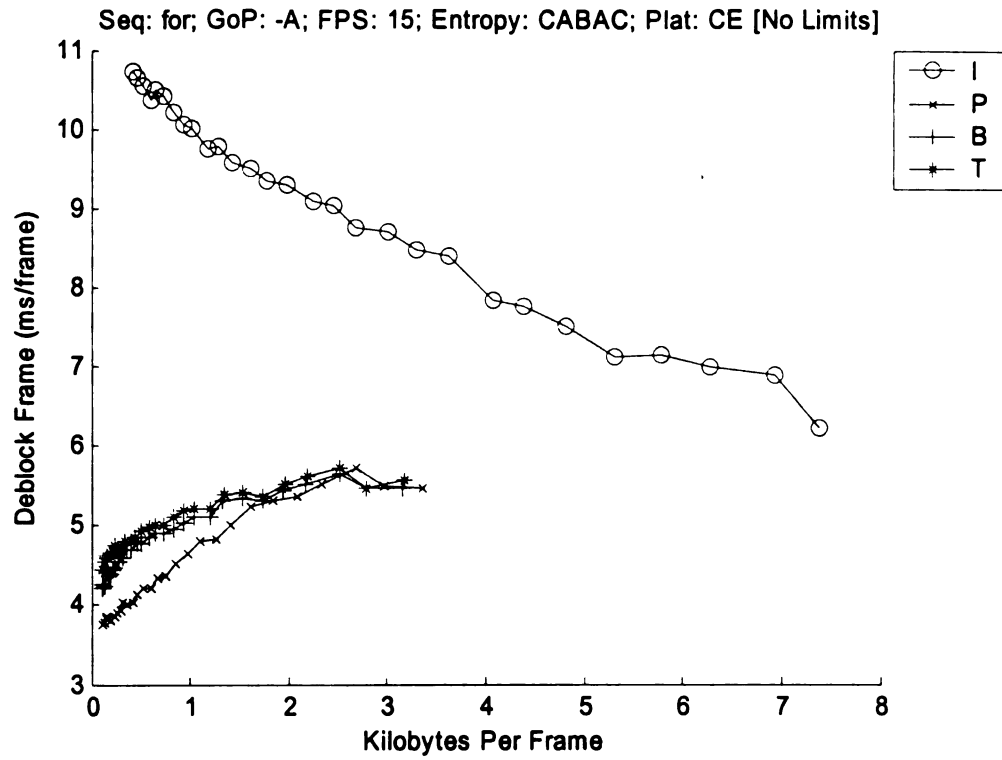


Figure 17: Foreman Deblock Rate-Complexity (15 FPS, CE)

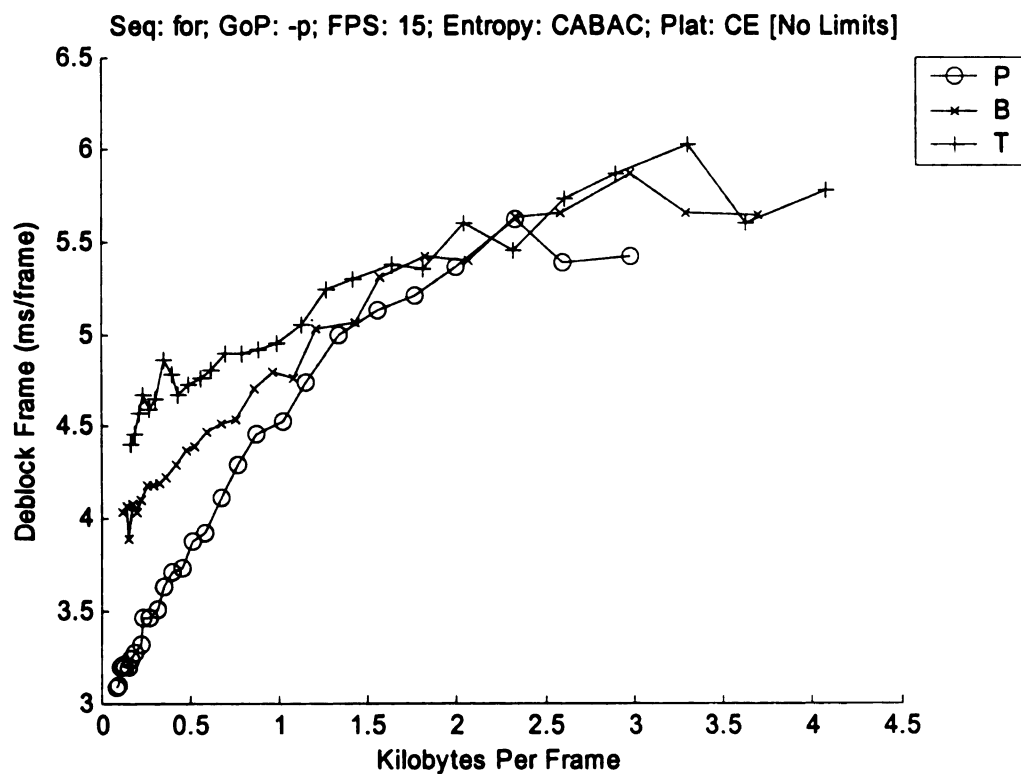
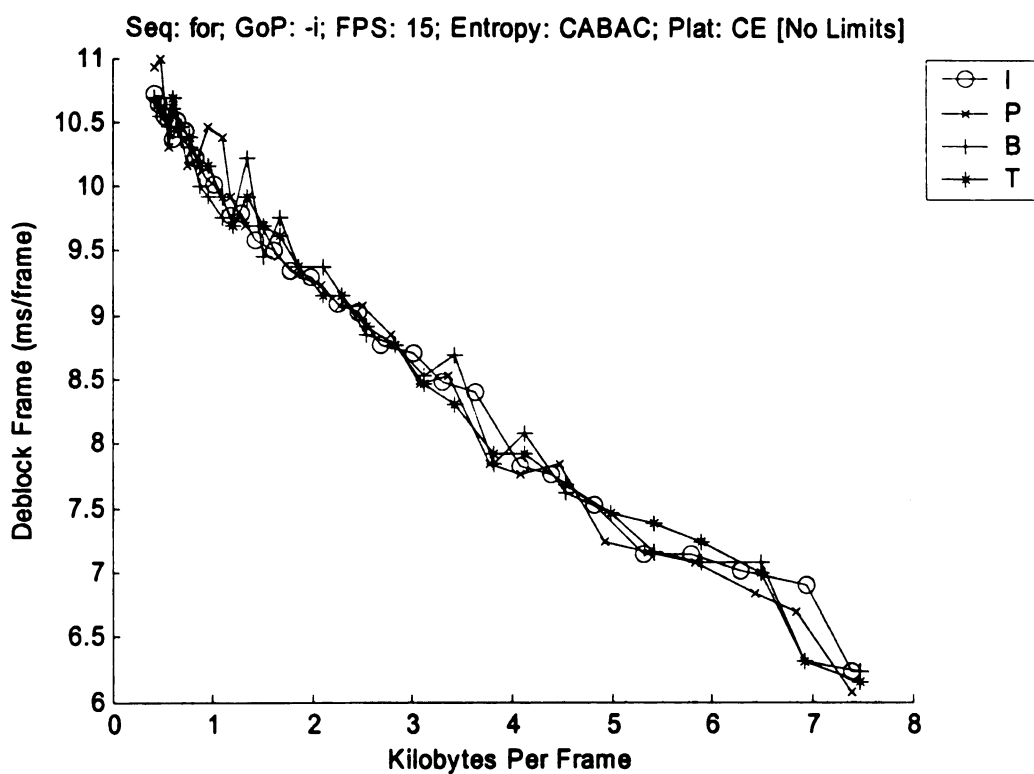


Figure 18: Foreman Deblock Rate-Complexity by Picture Type (15 FPS, CE)

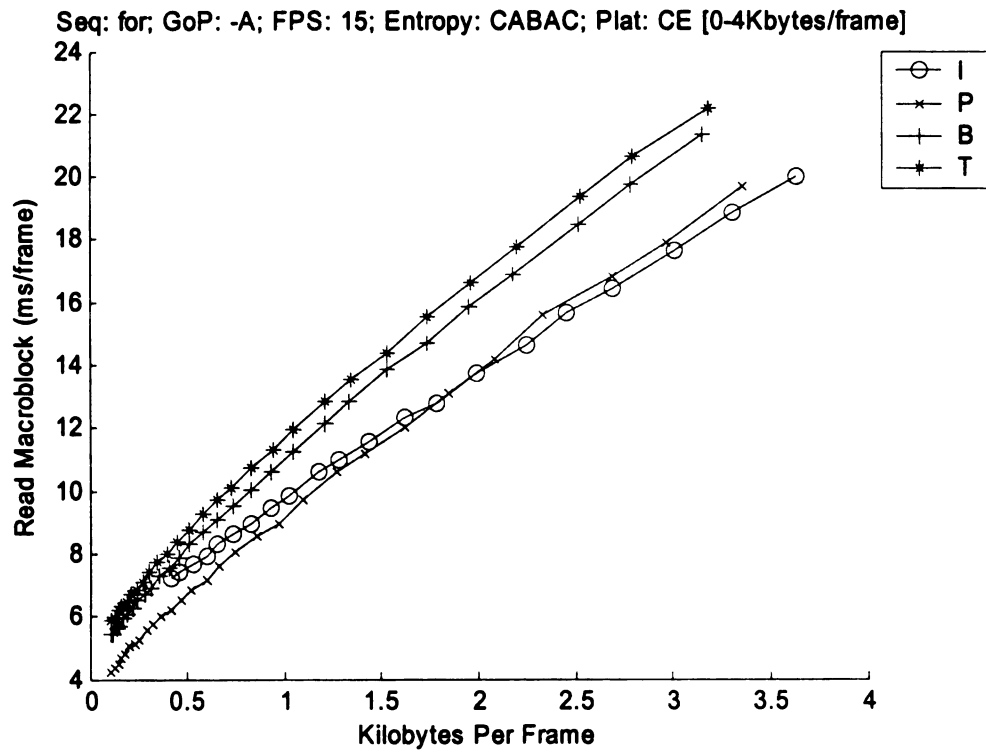
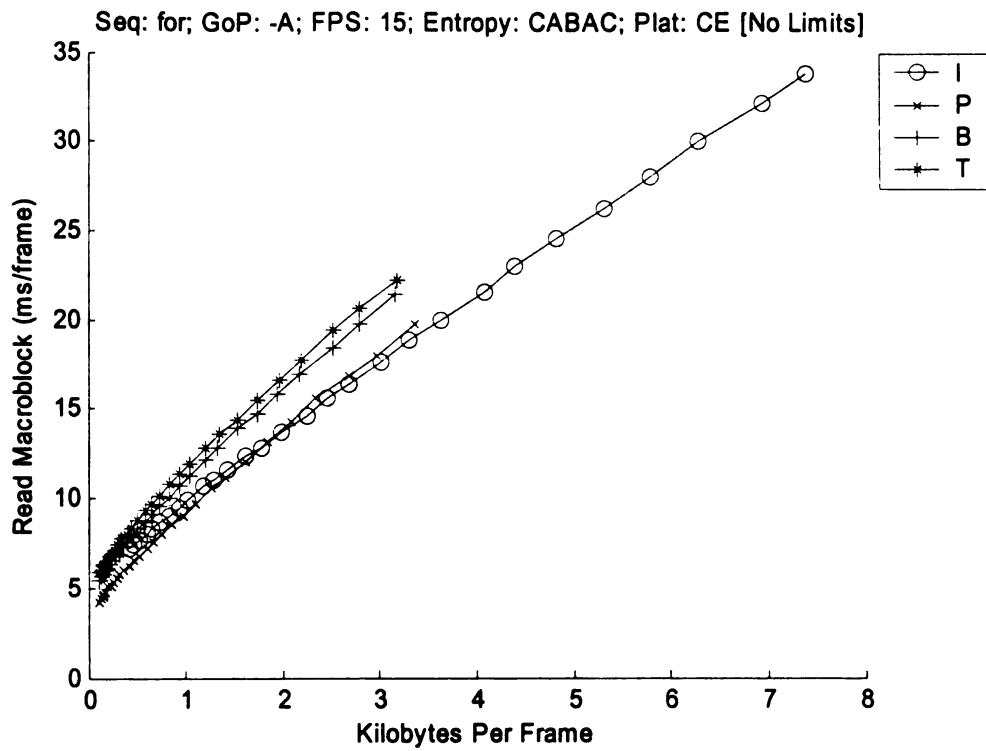


Figure 19: Foreman ReadMB Rate-Complexity (15 FPS, CE, CABAC)

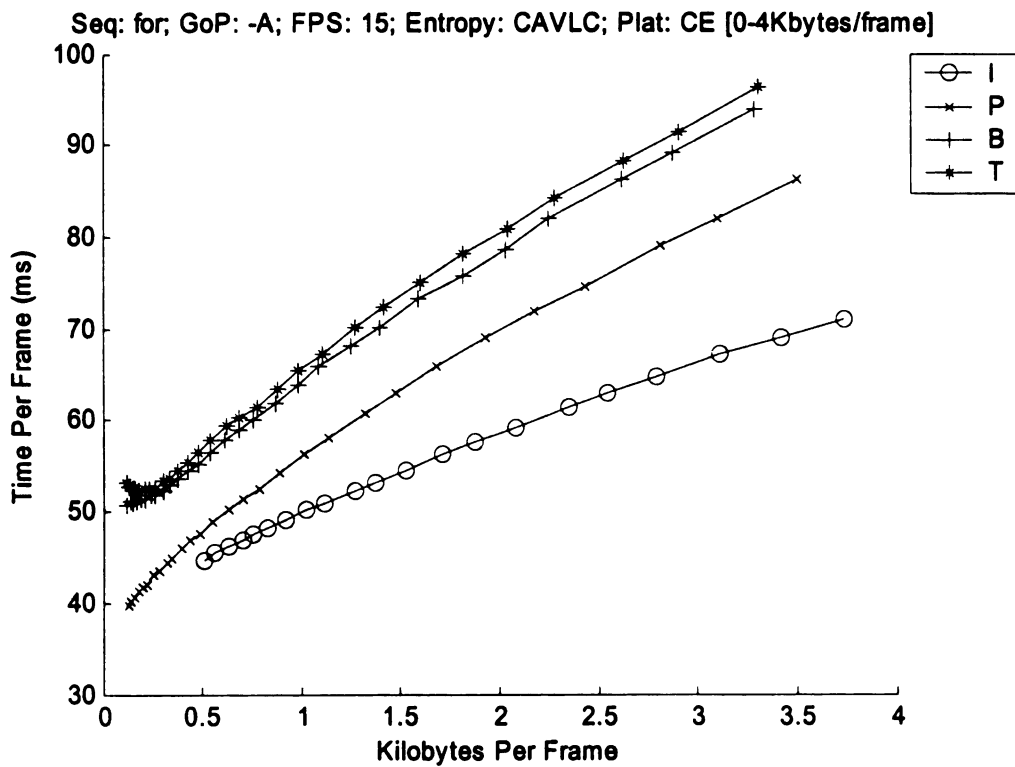
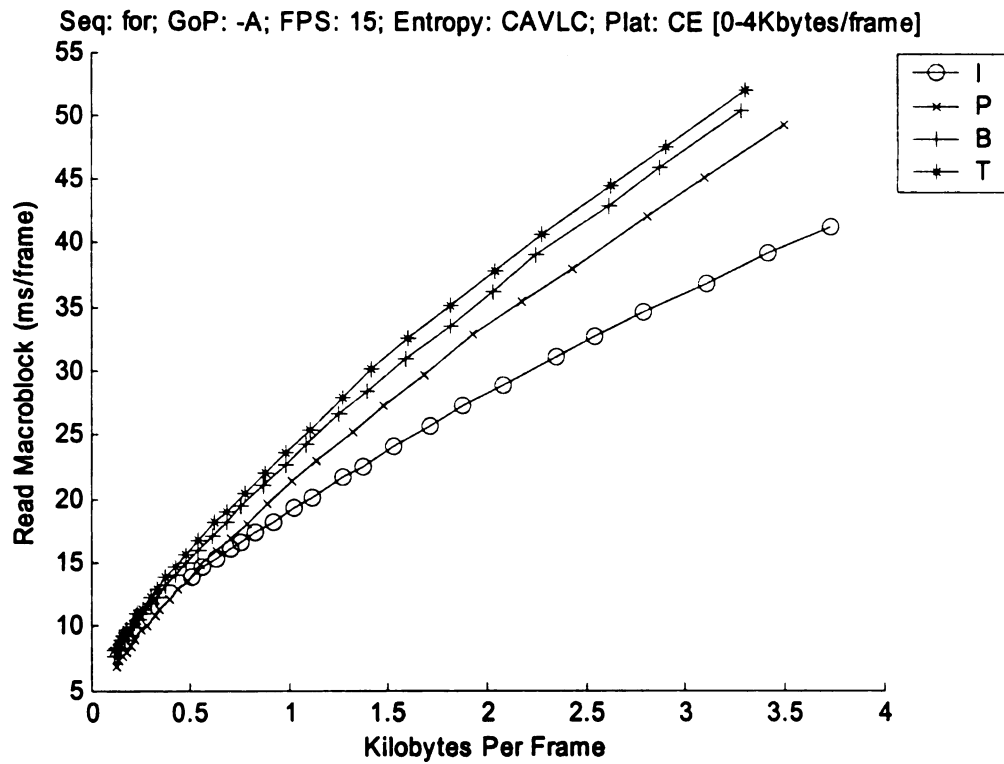


Figure 20: Foreman ReadMB Rate-Complexity (15 FPS, CE, CAVLC)

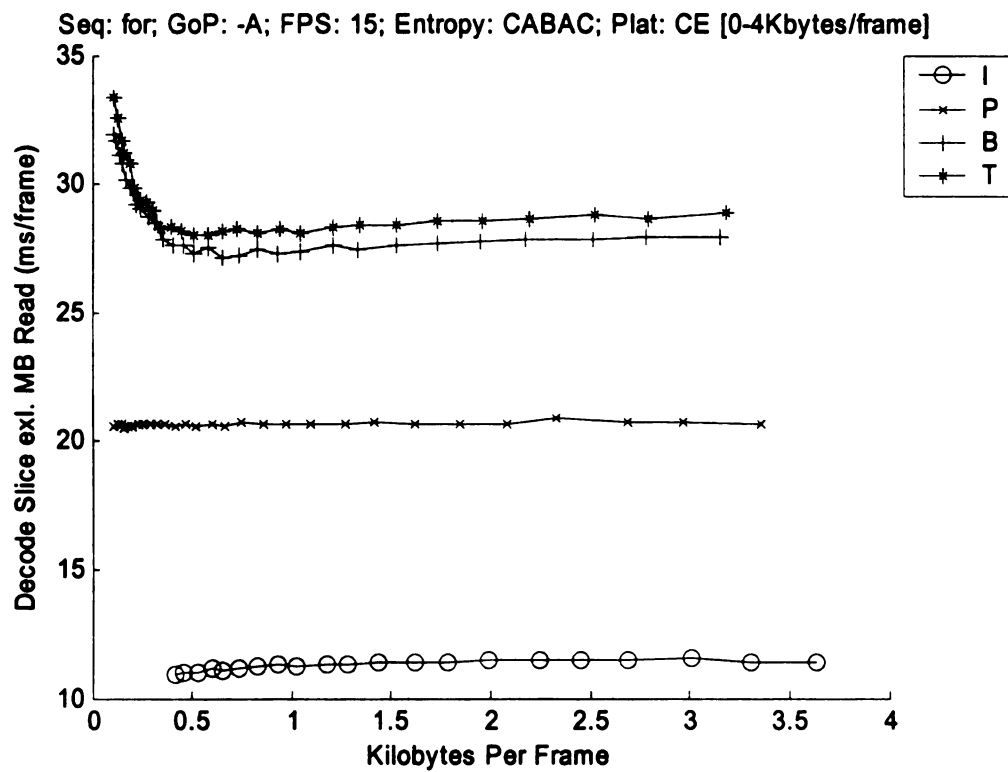
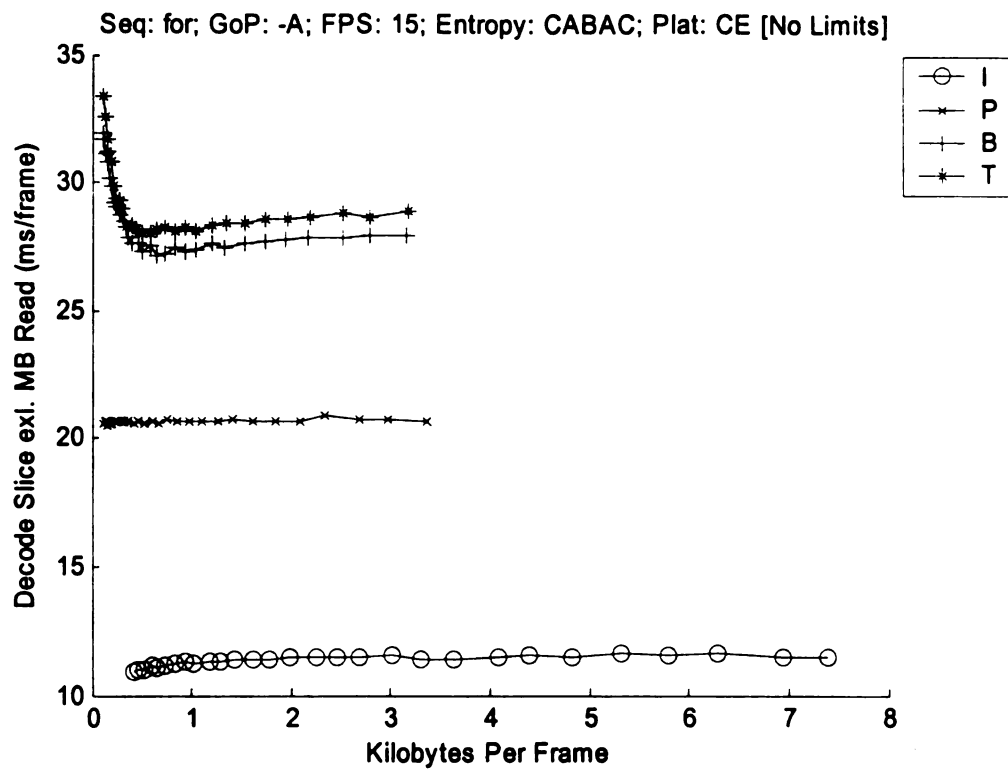


Figure 21: Foreman ProcessMB exl. ReadMB Rate-Complexity (15 FPS, CE, CABAC)

Mobile

This section breaks down the rate-complexity response of the decoder for the Mobile sequence. It deals first with the time invariant probes, then moves on to examine deblocking, parsing, and decoding excluding parsing. As before, the overall rate-complexity is shown in Figure 22. Table 8 presents the same time invariant probes shown for the Akiyo and Foreman sequences. Examining table against the previous tables (Table 6 and Table 7), it is clear that even with a highly complex sequence, these measures remain highly stable. One exception, the read new frame probe, slows down for the mobile at high rates. Excluding the Mobile I-GoP, the highest value for read new frame is 2.3ms (for CABAC). Even this increase over the Foreman results is at the higher bitrates for the mobile sequences. This again suggests that reading the new frame information is linked to the bitrate, but not in a significant way. The other significant increase, CAVLC ProcessMB other, is caused by higher times for the B and T-GoPs at high bitrates. However, even here the total increase is slight (0.6ms over Foreman) even after doubling the bitrate. (The 45dB B and T Foreman sequences took around 30Kb per frame; the Mobile sequences around 70Kb.)

The deblocking rate-complexities are presented in Figure 23 and then the specific complexity for I and P frames shown in Figure 24. As before, the deblocking complexity for the I and P frames is GoP independent although individual experiment variation can be seen. Figure 23 shows the deblocking complexities following a similar pattern to the Foreman series. The P, B, and T GoPs start out with low complexity and gradually increase until the rate reaches about 30Kb per frame. For higher rates (beyond the highest

Foreman rate), the deblock complexities drops slightly. The I GoP starts at a high complexity, roughly the same complexity as the rate equivalent Foreman, and continues dropping as the rate increases. This complexity response is similar to the previous two sequences and seems to be more closely related to bitrate than to sequence complexity.

The parsing complexity graphs are present in Figure 25 for CABAC (compare against Figure 22) and for CAVLC, Figure 26, which includes overall time as well as parsing time. Here the relation between parsing and overall complexity can be clearly seen. For example, the net increase in complexity for the CABAC I-GoP is roughly 45ms; the parsing increase also roughly 45ms. For the CAVLC P-GoP, a net increase of 90ms, with a parsing increase of over 75ms. The contribution of parsing to the overall complexity increase is further highlighted by Figure 27 which shows a very steady decoding complexity for non-parsing related decoding. The exception is a slight complexity increase during the 0-10Kb per frame portion of the B and T GoPs. Interestingly, the B and T non-parsing complexity is slightly less than the Akiyo complexity while slightly more complex than the Foreman GoPs. All three sequences show extremely close decoding complexities for the I and P GoPs once parsing time is excluded.

Summary

Overall, the Mobile sequence continues a pattern of parsing being a significant factor in the decoding complexity, especially with this implementation of the decoder. Because the Mobile sequence achieves poorer complexity than the Akiyo or Foreman sequences, it spans a broader set of bitrates, which highlights the complexity dependence on rate.

Deblocking has a small correlation to rate as do a handful of other portions of the

decoder. Aside from parsing, the frame decoding complexities are very similar to the Akiyo and Foreman frames, suggesting that sequence complexity does not significantly impact the non-parsing complexity.

Mobile Tables and Figures

Mobile CE Seq.	Time Range (ms)		Time Variance (ms)	Average Time (ms)
	<i>Fastest</i>	<i>Slowest</i>		
CABAC:				
<i>Read New Frame</i>	1.5	3.5	2.0	2.0
<i>Overhead</i>	0.21	0.60	0.39	0.45
<i>Unknown</i>	0.01	0.23	0.22	0.1
<i>Decode Frame Init</i>	3.88	4.13	0.25	3.97
<i>Process MB Other</i>	2.93	3.90	0.97	3.3
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.4; P: 2.5 B: 1.3; T: 0.8
CAVLC:				
<i>Read New Frame</i>	1.5	3.5	2.0	2.3
<i>Overhead</i>	0.25	0.63	0.38	0.43
<i>Unknown</i>	0.02	0.22	0.20	0.11
<i>Decode Frame Init</i>	3.66	3.90	0.24	3.76
<i>Process MB Other</i>	3.4	5.5	2.1	4.0
<i>Post Frame Decode</i>			+/- 0.1 ms from average times	I: 2.3; P: 2.4 B: 1.2; T: 0.8

Table 8: Mobile Rate Invariant Probes (15fps, CE)

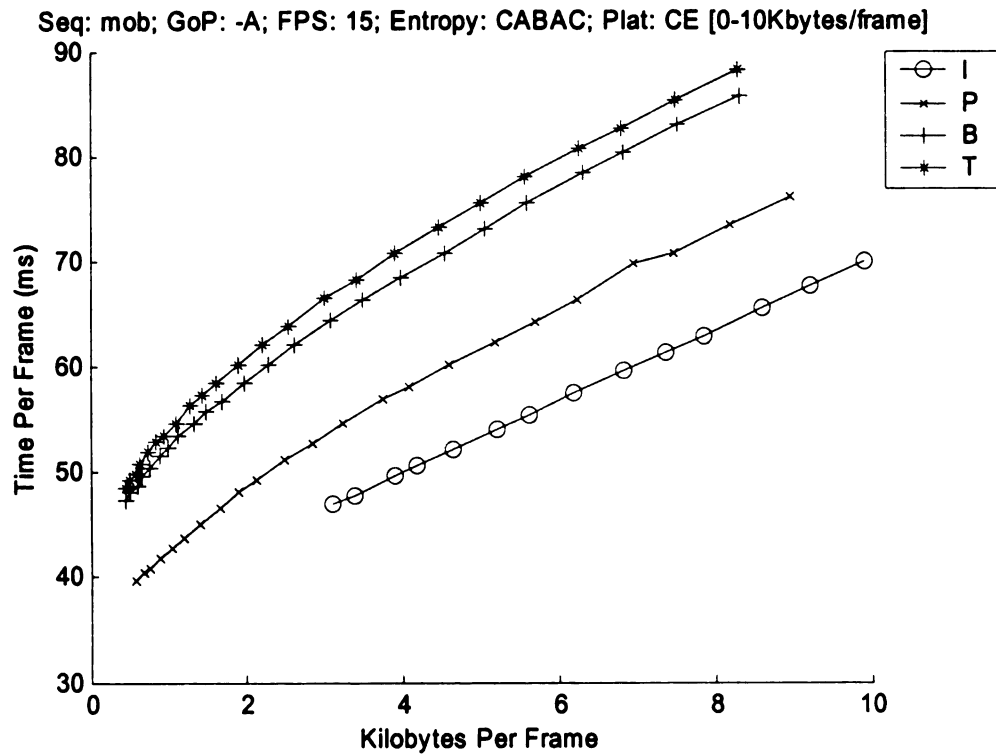
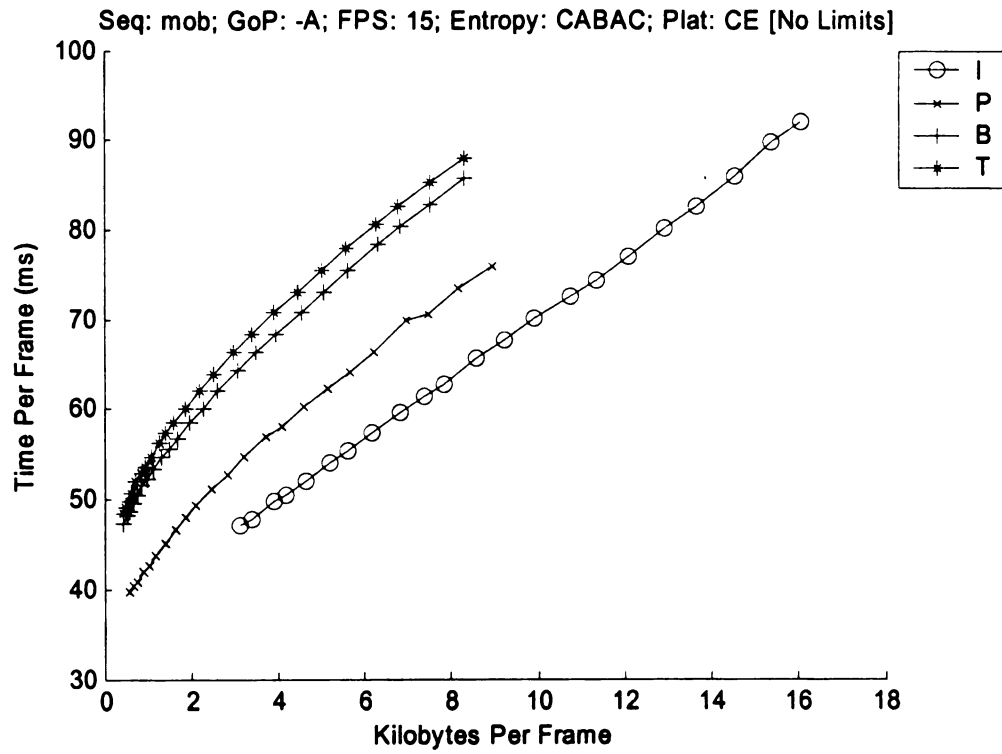


Figure 22: Mobile Rate-Complexity (15 FPS, CE)

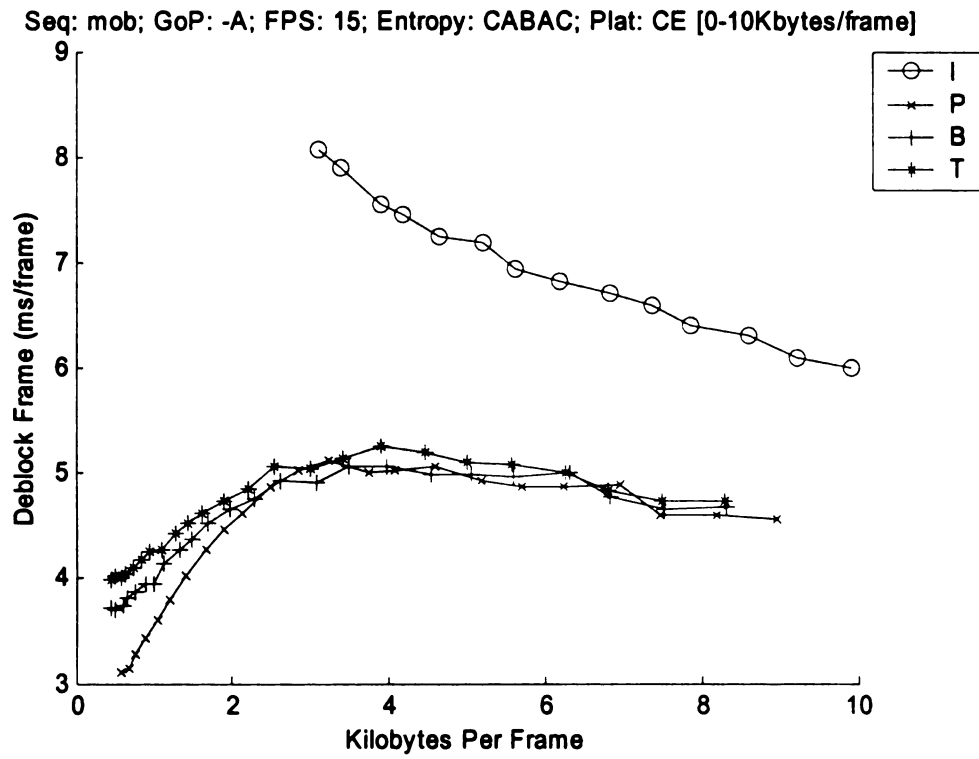
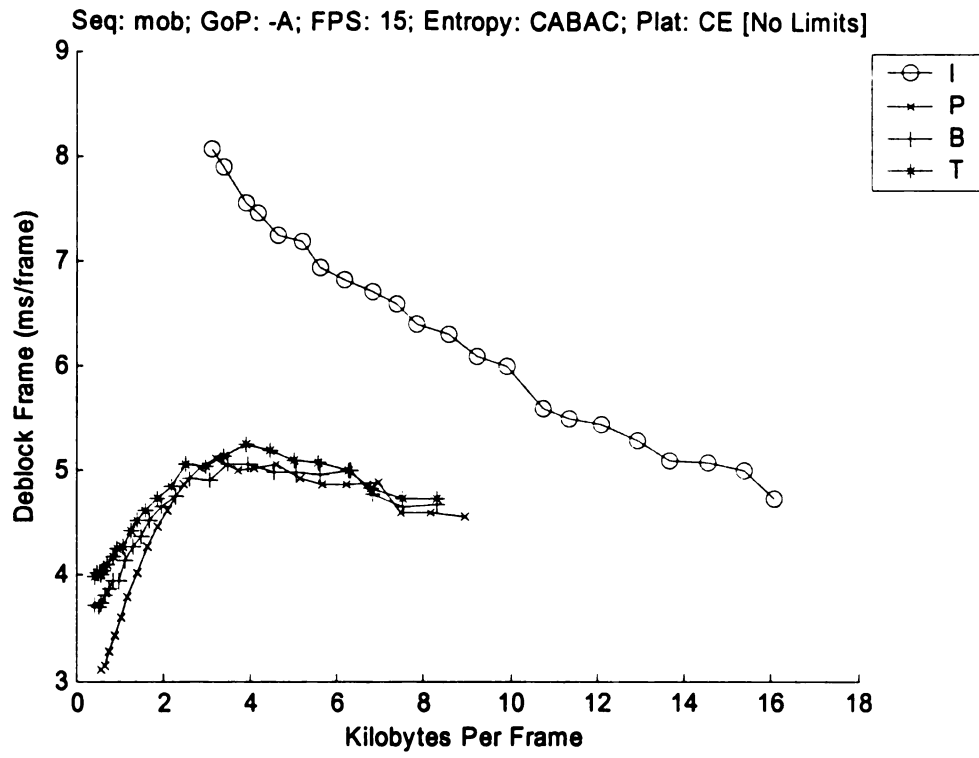


Figure 23: Mobile Deblock Rate-Complexity (15 FPS, CE)

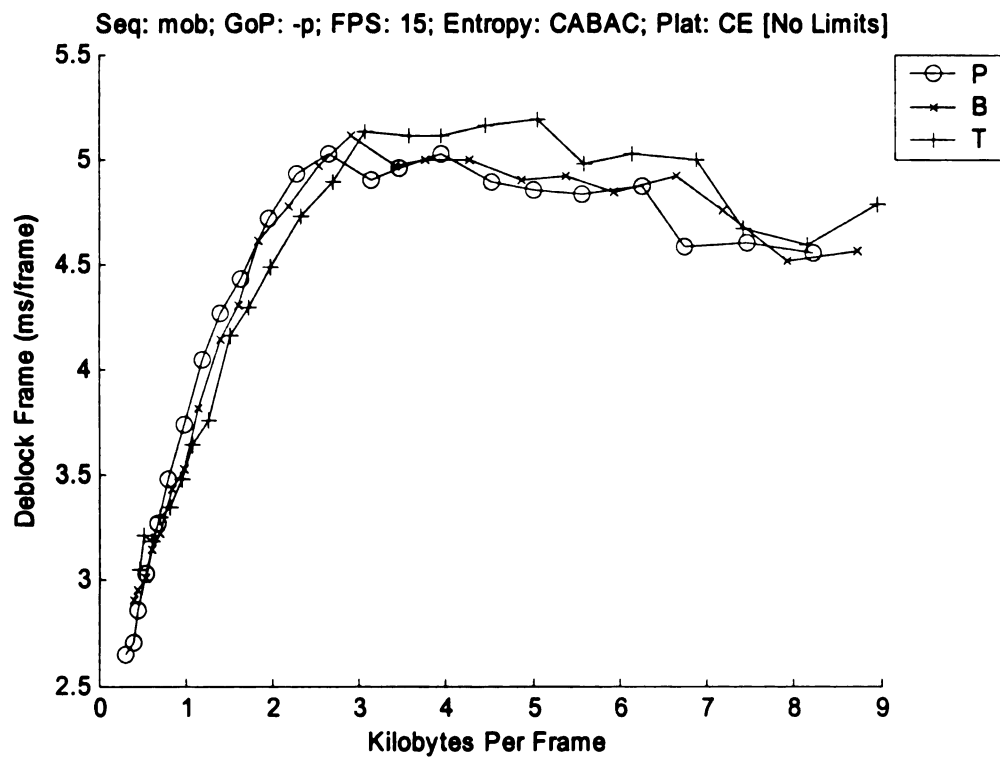
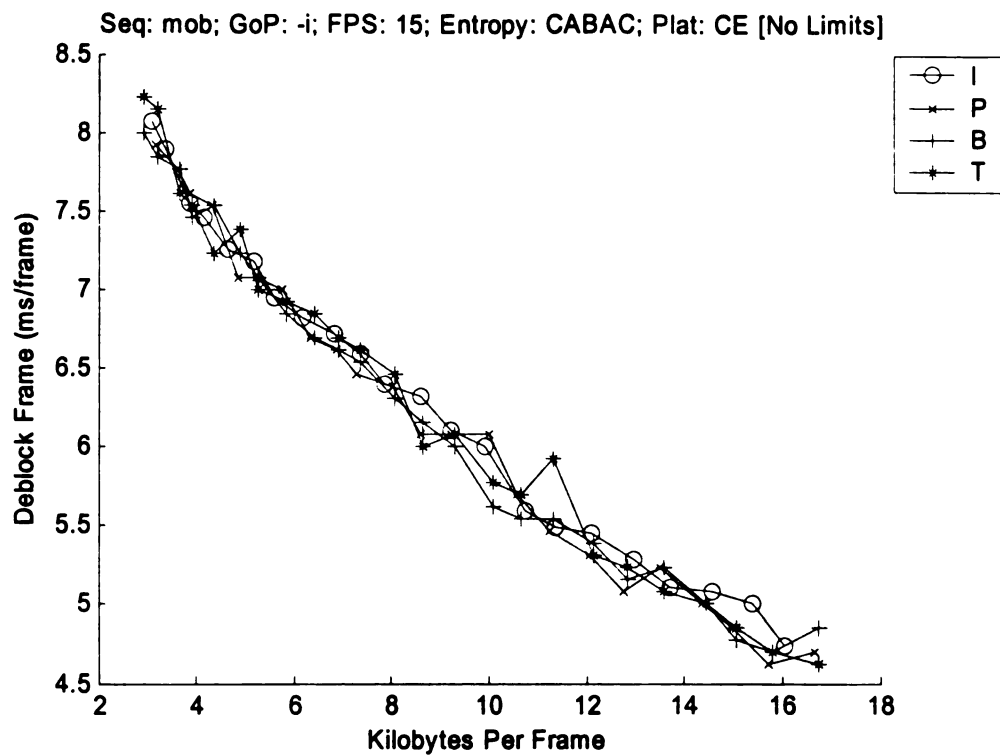


Figure 24: Mobile Deblock Rate-Complexity by Picture Type (15 FPS, CE)

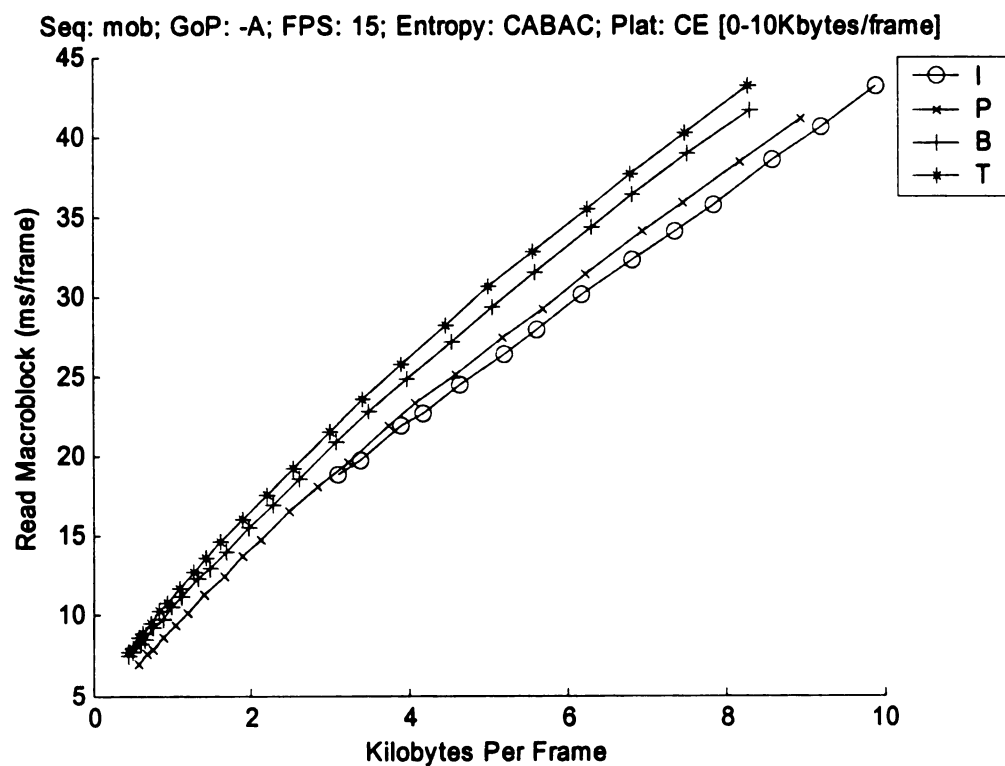
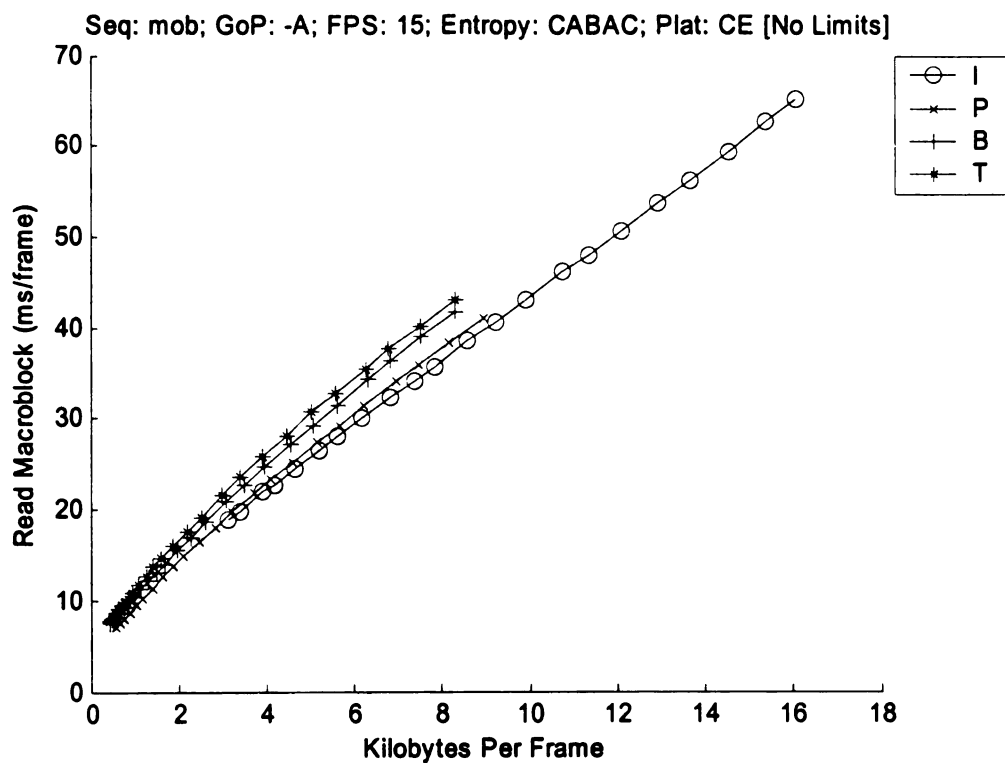


Figure 25: Mobile ReadMB Rate-Complexity (15 FPS, CE, CABAC)

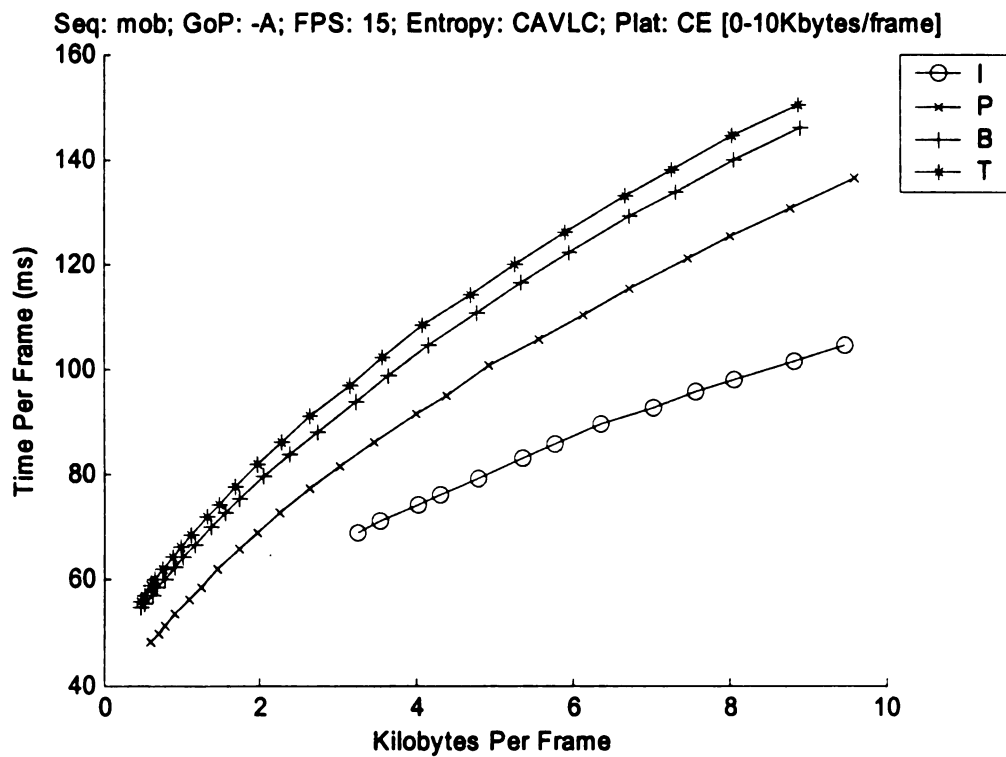
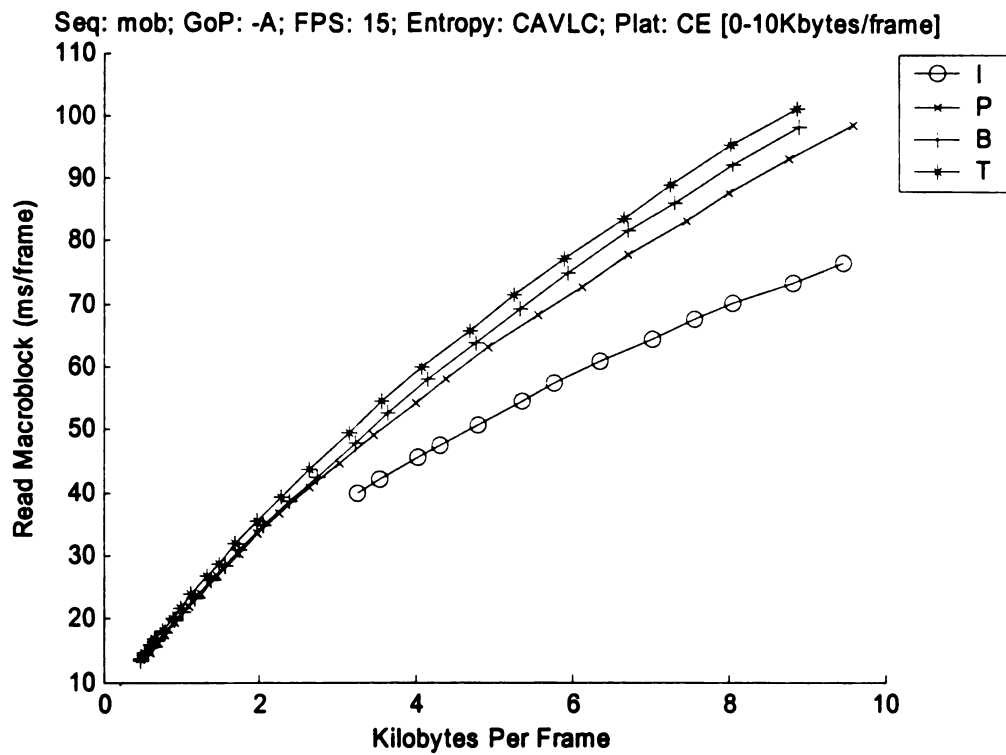


Figure 26: Mobile ReadMB Rate-Complexity (15 FPS, CE, CAVLC)

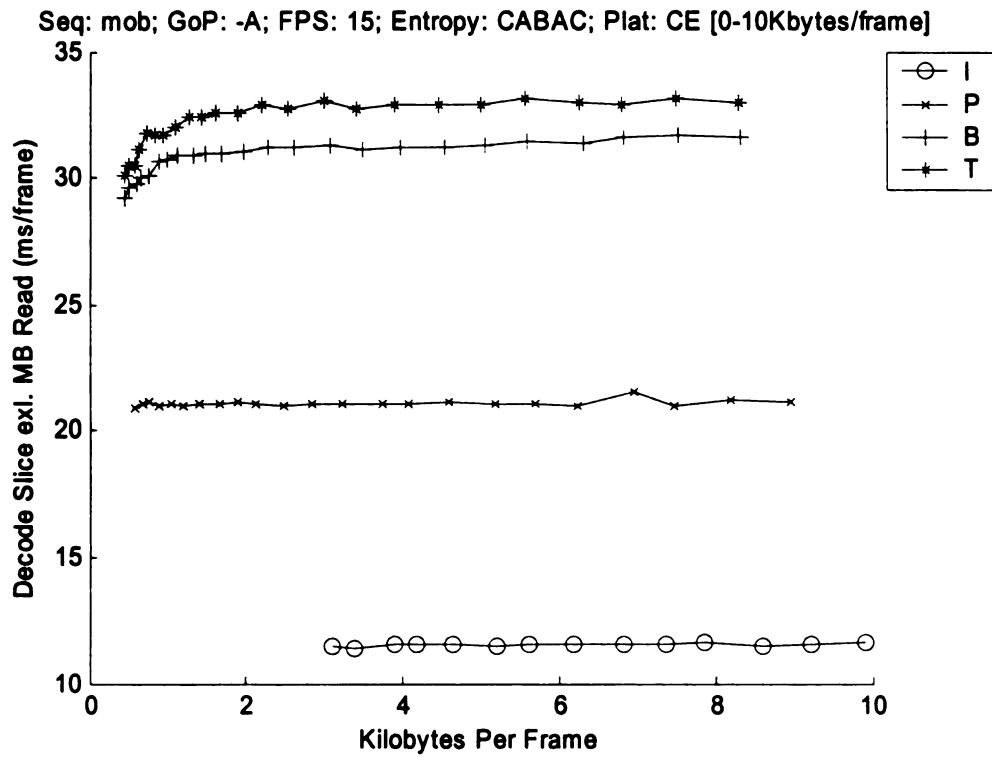
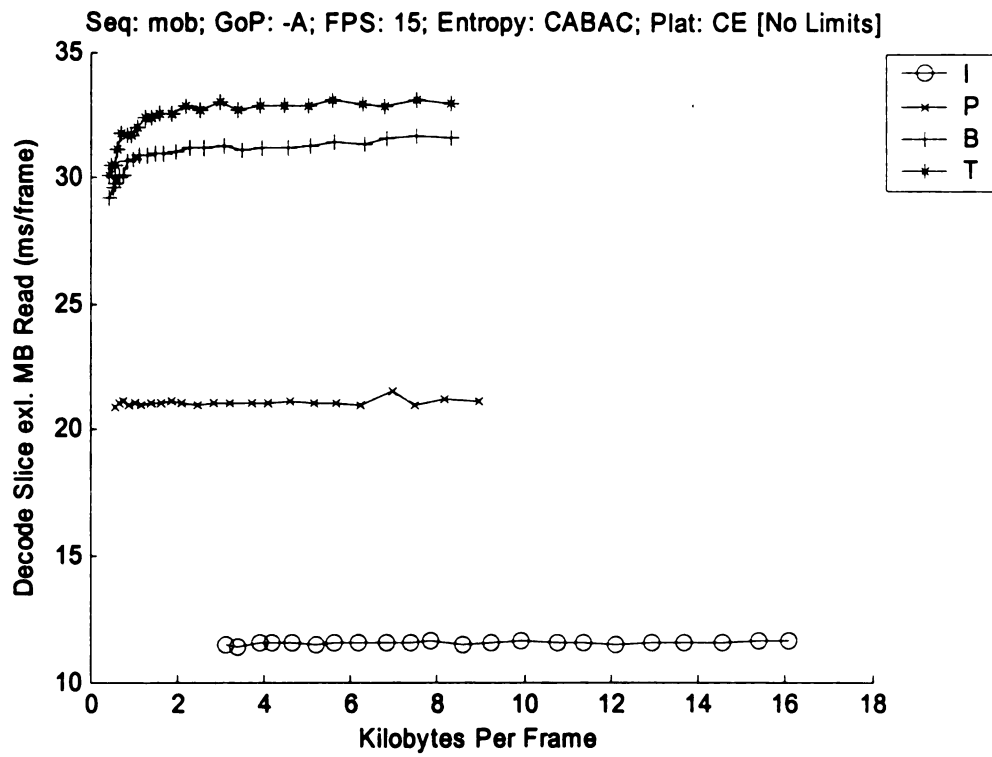


Figure 27: Mobile ProcessMB exl. ReadMB Rate-Complexity (15 FPS, CE, CABAC)

Conclusions

An overview of the relative complexities of decoding macroblocks compared to parsing rates for this implementation can be seen in Table 9¹³. The rates were found by the slope of the linear line between the 25dB complexity and the 45dB complexity for each category. The range represents the different sequences, and numbers are given for each *picture type*, not group of picture structure. That is, for mobile P frames, P frames were combined from the Mobile P, B, and T GoP structures. As can be seen from the figures presented earlier in this chapter, the linear assumption is not entirely accurate. For example, the complexity per bit tends to decrease at high bitrates. However, linearity is a close approximation and provides practical summary of the data.

The table shows very consistent rates for decoding I and P-frames, with more variance in the time for B-frame decoding. However, the I-frames clearly decode the fastest, followed by P frames. B-frames follow a distant third. The same relationship holds true for the parsing rates as well. The table highlights the poor performance of CAVLC parsing in this implementation.

The comparative performance boost from the Pentium IV is shown as well. The clock rate of the Pentium is roughly three times greater than the iPAQ, and the speed multiplier gives a relative sense of performance increase, not a direct comparison of capability. If the complexity relationship was linear, the multiplier would be roughly the same for each category. Instead, I-frames decode about four times faster, P-frames around five times faster, and B-frames a bit above six times faster. The CABAC parsing, in contrast, is

fairly consistent, running four and a half to five times faster on the PC. The CAVLC parsing, on the other hand, ranges from the four times faster for the CABAC to something like ten times faster for B frames. This range in performance speedup serves to highlight the Pentium optimizations are not being uniformly applied to the decoding process, but boost specific sections of the decoder much more than others.

	Frame Type	iPAQ 400MhZ	Pentium IV, 1.5GhZ		PC Speedup (Times Faster)	
					Low End	High End
Decoding, excl. Parsing (ms/frame)						
	I	11.5-13.5	2.75-3.10		4.18	4.35
	P	21.5-22.5	3.95-4.45		5.44	5.06
	B	33.0-48.5	5.40-7.10		6.11	6.83
Parsing (bytes/ms)						
	CABAC Coding					
	I	250-280	1100-1300		4.40	4.60
	P	160-250	720-1100		4.50	4.40
	B	80-210	450-1000		5.63	4.76
CAVLC Coding						
	I	135-245	600-1130		4.44	4.61
	P	50-100	380-520		7.60	5.20
	B	30-80	360-500		12.00	6.25

Table 9: Complexity Summary over 25-45 dB range

The timing analysis of the three separate sequences based upon complexity and bitrate suggests that much of the decoding process is both rate and sequence independent.

Improvements to these sections are likely to equally improve any video sequence. Two sections, deblocking and parsing, show a correlation between complexity and bitrate.

The deblocking correlation with rate appears extremely limited with the I-frames deblocking faster with more bits per frame. The other frames experience a slight

¹³ This is the same table as shown in the introduction.

complexity increase of roughly 2ms over the 0 to 60Kb per frame range. However, this increase is relatively insignificant compared the parsing complexity increases. I-frames are three to four times slower to deblock than other frames at high distortion. At low distortion, all the GoPs take about the same amount of time to deblock.

The parsing complexity, while it varies by frame type and to a smaller degree sequence, overall shows a strong linear correlation to bitrate. For poorly designed parsing code, this portion of the complexity easily dominates the decoding complexity. Comparing the three sequences strongly suggests that bitrate, not sequence complexity, is the primary factor in decoding time.

Aside from parsing, the baseline decoding complexity is extremely level, with the P-GoP being about twice as complex as the I-GoP, and the B and T-GoPs being about three times more complex than the I-GoP. This complexity difference suggests that in addition to parsing improvement, significant room for improvement exists for the motion prediction implementation.

CHAPTER 5 CONCLUSION

Summary

The major contributions of this research include highlighting the performance difference between computers and mobile devices; emphasizing that parsing, especially for poorly compressed sequences, can be a controlling complexity factor, and providing a framework against which to compare other handheld performance.

The complexity response between PCs and handhelds, as discussed in Chapter 3, is significant with complexity ratios varying at a high level based on both entropy and picture type. The implication is that the complexity models researched and published for desktop computers, especially Pentiums, are not entirely transferable to handheld devices. The handheld device architecture is sufficiently different that while algorithm optimizations are unlikely to hurt performance, equivalent improvement should not be assumed. Additionally, separate handheld algorithm improvements may be possible which significantly impact the handheld devices without corresponding improvement for the PCs. One example of this type of change is the memory management change which took approximately one-third of the handheld time but an estimated two percent of the PC. Additional professional tools are needed to profile the decoder performance on handheld devices.

The primary complexity controlling factor for both entropy coding methods was the read macroblock, or parsing, portion of the code. Despite the lack of research into the entropy

coding complexity constraints, practical optimum implementations, for a variety of platforms aside from hardware additions, are critical to prevent parsing from being the controlling aspect of the sequence complexity. Despite the focus on traditionally complex aspects such as transforms and deblocking, which indeed remain problematic, parsing requires attention. This attention is especially needed if handhelds are to take advantage of possessing a higher bandwidth to computation power ratio.

In general, the P-GoP provides a balance between the high bitrate of a mostly I-frame sequence and the high complexity of B-frames due to the double motion vector. The P-GoP provides a high level of compression significantly reducing the bitrate without the added overhead of a second motion vector. When parsing is ignored, the decoding complexity of the P-frame does reflect a significantly higher complexity than the I-frame, and a more tightly optimized entropy decoder might reduce the I-frame complexity to the point where I-frames are less complex. Significant work on reducing the complexity of motion compensation would be needed, however, before B frames become a viable option.

Future Directions

The research into the complexity-distortion suggests that additional research on several fronts would provide further valuable insight into video complexity-optimization for handheld devices.

One specific research is the entropy symbol parsing complexity for non-hardware specific solutions. While hardware solutions have been introduced capable of processing on the order of 50 to 100 million symbols per second, equipping handheld devices with entropy decoding processors may not be a practical solution.

There is a clear need for better profilers to do architectural analysis of handheld devices. While professional solutions such Intel's VTune exist for PCs, equivalent grade tools do not exist for handheld devices. Despite the problems in implementing such tools, the ability to easily probe specific functions and optimizations in terms of complexity would be a useful step in real time video optimization.

A third research aspect is additional investigation into architecture specific behavior and ways of utilizing the operating system and architectural features to further reduce complexity. While this research ignored the operating system interaction, operating system interaction may not be negligible. Additionally, while neither of the two compilers tested showed superior performance, the optimization quality of handheld compilers is likely to be less than their PC counterparts. The lack of complexity difference between Intel and Microsoft compilers further suggests such improvements may be possible.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," Joint Video Team (JVT), Mar. 2003.
- [2] H. Schwarz and T. Wiegand, "The Emerging JVT/H.26L Video Coding Standard," presented at Proc. of the IBIC, Sept. 2002.
- [3] N. Kamaci and Y. Altunbasak, "Performance Comparison of the Emerging H.264 Video Coding Standard with the Existing Standards," presented at IEEE ICME, July 2003.
- [4] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 704-716, July 2003.
- [5] V. Lappalainen, A. Hallapuro, and T. Hamalainen, "Complexity of Optimized H.26L Video Decoder Implementation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 717-725, July 2003.
- [6] H. Malvar, A. Hallapur, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 598-603, July 2003.
- [7] A. Chang, O. Au, and Y. M. Yeung, "A Novel Approach to Fast Multi-Block Motion Estimation for H.264 Video Coding," presented at IEEE ICME, Baltimore, Maryland, 2003.
- [8] I. Richardson, "H.264 / MPEG-4 Part 10 : Variable Length Coding," in *H.264 White Papers*, 2002, Available: http://www.vcodex.fsnet.co.uk/h264_vlc.pdf
- [9] G. Bjøntegaard and K. Lillevold, "Context-adaptive VLC (CVLC) coding of coefficients," Fairfax, Virginia: Doc JVT-C028-r1, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6), May 2002.
- [10] D. Marpe, G. Blattermann, G. Heising, and T. Wiegand, "Video Compression Using Context-Based Adaptive Arithmetic Coding," *Proc. IEEE ICMP '01*, vol. III, pp. 558-561, Oct. 2001.
- [11] D. Marpe, H. Schwarz, and T. Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in H.264/AVC Video Compression Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 620-636, July 2003.
- [12] S. Saponara, C. Blanch, K. Denolf, and J. Bormans, "The JVT Advanced Video Coding Standard: Complexity and Performance Analysis on a Tool-by-Tool Basis," presented at Packet Video, April 2003.
- [13] E. Baum, V. Harr, and J. Speidel, "Improvement of H.263 Encoding by Adaptive Arithmetic Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, August 2000.
- [14] I. Witten, R. Neal, and J. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, June 1987.
- [15] D. Marpe and T. Wiegand, "A Highly Efficient Multiplication-Free Binary Arithmetic Coder and its Application to Video Coding," *IEEE Proc. ICIP*, Sept 2003.

- [16] I. Richardson, "H.264 / MPEG-4 Part 10: Transform & Quantization," in *H.264 White Papers*, Mar. 2003.
- [17] MPEG-2: ISO/TEC JTC1/SC29/WG11 and ITU-T, "Revised text for ITU-T recommendation H.262-ISO/IEC 13 818-2: Information technology-generic coding of moving pictures and associated audio information Video," ISO/IEC and ITU-T, Ed. Genf, Switzerland, 1995.
- [18] K. Panusopone and D. Baylon, "An Analysis and Efficient Implementation of Half-Pel Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, Aug. 2002.
- [19] O. Werner, "Drift Analysis and Drift Reduction for Multiresolution Hybrid Video Coding," *Signal Processing: Image Commun.*, vol. 8, July 1996.
- [20] T. Wedi and H. G. Musmann, "Motion- and Aliasing-Compensated Prediction for Hybrid Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, July 2003.
- [21] T. Wieland, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, July 2003.
- [22] S. Akramullah, I. Ahmad, and M. Liou, "Optimization of H.263 Video Encoding Using a Single Processor Computer: Performance Tradeoffs and Benchmarking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, pp. 901-915, Aug. 2001.
- [23] K. Yu, J. Lv, J. Li, and S. Li, "Practical Real-Time Video Codec for Mobile Devices," *IEEE ICME 2003*, 2003.
- [24] SymbolicTools, *Port SDK*, 2003, Available: <http://www.symbolictools.de/public/pocketconsole/developers/portsdk.htm>
- [25] SymbolicTools, *PocketConsole Version 1.3*, 2003, Available: <http://www.symbolictools.de/public/pocketconsole/>
- [26] JVT, *JM 6.1e*, 2003, Available: <http://bs.hhi.de/~suehring/tml/download/jm61e.zip>

General References

Harald Kosch, *Distributed Multimedia Database Technologies Supported MPEG-7 and MPEG-21*, Boca Raton: CRC Press, 2004.

A. Murat Tekalp, *Digital Video Processing*, Upper Saddle River: Prentice Hall, 1995.

A. Peymandoust, T. Simunic, and G. De Micheli, "Low Power Embedded Software Optimization using Symbolic Algebra," presented at DATE'02, March 2002.

K. Yu, J. Lv, J. Li, and S. Li, "Practical Real-Time Video Codec for Mobile Devices," presented at ICME 2003, July 2003.

University of British Columbia, *TMNDecode*, 1997, Available: <http://www.openmash.org/lxr/source/codec/tmndec>

- T. Turetti and C. Huitema, "Videoconferencing on the Internet," *IEEE/ACM Transactions on Networking*, vol. 4, June 1996.
- J. Zheng and L. Chau, "A Motion Vector Recovery Algorithm for Digital Video Using Language Interpolation," *IEEE Transactions on Broadcasting*, vol. 49, Dec. 2003.
- N. Ling and N. Wang, "Real-Time Video Decoding Scheme for HDTV Set-Top Boxes," *IEEE Transactions on Broadcasting*, vol. 48, Dec. 2002.
- M. Sima, Y. Zhou, and W. Zhang, "An Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding," *IEEE Transactions on Consumer Electronics*, vol. 50, Feb. 2004.
- T. Chang, Y. Tsai, C. Chien, C. Lin, and J. Guo, "A High-Performance Bitstream Processing IP Core Design for MPEG4 Video Compression Applications," presented at *International Symposium on Nanoelectronic Circuits and Giga-scale systems (ISNCGS 2004)*.
- K. Yamada, M. Kojima, T. Shimizu, F. Sato and T. Mizuno, "A new RISC processor architecture for MPEG-2 decoding," *IEEE Transactions on Consumer Electronics*, vol. 48, Feb. 2002.
- B. Zheng and M. Atiquzzaman, "A Novel Scheme for Streaming Multimedia to Personal Wireless Handheld Devices," *IEEE Transactions on Consumer Electronics*, vol. 49, Feb. 2003.
- S. Lee and I. Park, "A Low-Power Variable Length Decoder for MPEG-2 Based on Successive Decoding of Short Codewords," *IEEE Transactions on Circuits and Systems*, vol. 50, Feb. 2003.
- A. Chimienti, C. Ferraris, and D. Pau, "A Complexity-Bounded Motion Estimation Algorithm," *IEEE Transactions on Image Processing*, vol. 11, April 2002.

MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 02504 3203