IMPROVEMENTS IN WAVELET-BASED RATE SCALABLE VIDEO COMPRESSION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Eduardo Asbun

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2000

Because it is there.

George L. Mallory, 1924.

To my family: Tere, Felix, Ana Pau, Cecy, David, and Paty
for their love, support and encouragement.

ACKNOWLEDGMENTS

I would like to thank Prof. Edward J. Delp for his academic guidance and professional advice. My co-advisor, Dr. Paul Salama, from whom I have learned so much, for his insight and endless help. And Professors Edward J. Coyle, Susanne E. Hambrusch, and Leah H. Jamieson, for serving in my Ph.D. committee, and for their comments and suggestions to improve the quality of my work.

I am particularly indebted to Prof. John K. Antonio, who guided me through the first years of my Ph.D. He suggested me to take my first course on digital signal processing because *"... all electrical engineers should know this."* Little did I know the tremendous impact that these words were going to have in my professional life.

To Prof. Jan P. Allebach from whom I have learned to work with the highest degree of professionalism. To the Faculty of the School of Electrical and Computer Engineering, Department of Computer Science, and Department of Mathematics at Purdue, for their dedication to teaching and education, and their genuine desire to make a difference.

My officemates, past and present: Gregory Cook, Eugene Lin, Dr. Sheng Liu, Dr. Ke Shen, Jennifer Talavage, Cuneyt Taskiran, and Raymond Wolfgang, who have shared many great times with me. So have the other members of the Video and Image Processing Laboratory: Dr. Moses Chan, Dan Hintz, Hyung Cook Kim, Linda Kohout, and Martha Saenz. They all make a team of great colleagues and friends.

I have been extremely lucky to have friends who have been there for me many times during my career at Purdue. My dearest friends Adriana and Agustin, Claudia and Luis, Carmen Teresa and Luis Henrique, Lorena and Marco, Silvia and Alejandro, and Susana and Diego. I have found a treasure in them. They have made the Purdue experience an unforgettable one.

Last, but not least, I would like to thank the Science and Technology Council of Mexico (CONACyT), the Fulbright Program of the Institute of International Education (IIE), and Texas Instruments, for partially supporting this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AMC:	Adaptive block-based Motion Compensation
B frame:	Bidirectionally predictive-coded frame
bps:	Bits per second
CEZW:	Color Embedded Zerotree Wavelet
CIF:	Common Intermediate Format
DCT:	Discrete Cosine Transform
DSP:	Digital Signal Processor
DWT:	Discrete Wavelet Transform
EZW:	Embedded Zerotree Wavelet
fps:	Frames per second
GOP:	Group of Pictures
I frame:	Intracoded frame
kbps:	Kilobits per second (kilo (k) : 1,000)
Mbps:	Megabits per second (Mega (M) : 1,000,000)
MPEG:	Moving Pictures Experts Group
P frame:	Predictive-coded frame
PEF:	Predictive error frame
PSNR:	Peak Signal to Noise Ratio
QCIF:	Quarter Common Intermediate Format
SAMCoW:	Scalable Adaptive Motion Compensated Wavelet
SOT:	Spatial orientation tree
SPIHT:	Set Partitioning in Hierarchical Trees
UMV:	Unrestricted motion vectors

ABSTRACT

Asbun, Eduardo, Ph.D., Purdue University, December, 2000. Improvements in Wavelet-Based Rate Scalable Video Compression. Major Professors: Edward J. Delp and Paul Salama.

Delivery of video in the presence of bandwidth constraints is one of the most important video processing problems. Most current compression techniques require that parameters, such as data rate, be set at the time of encoding. However, it is difficult to predict the traffic on a network when video is to be delivered. Compression techniques that allow the change of the compression parameters at the time of decoding are very valuable due to the flexibility they provide. These techniques are said to be "scalable." Rate scalability, the capability of decoding a compressed image or video sequence at different data rates, is the one of the most important modes for video streaming over packet networks such as the Internet.

SAMCoW is a rate scalable, wavelet-based video compression algorithm developed at Purdue University in 1997 by Shen and Delp. *SAMCoW* is fully and continuously rate scalable, and has performance similar to MPEG-1, MPEG-2, and H.263 at comparable data rates. In this thesis, we investigate several extensions to *SAMCoW*. In particular, we investigate the use of advanced video coding techniques, preprocessing, postprocessing, and rate-distortion theory, in order to develop a framework for efficiently encoding intracoded (I) and predictive error (PE) frames as part of *SAMCoW*. These extensions are known as *SAMCoW+*.

We also investigate the use of digital signal processors for real-time video processing. We demonstrate the use of a Texas Instruments *TMS320C6201* for real-time error concealment of digital video streams.

1. INTRODUCTION

The delivery of video in the presence of bandwidth constraints is one of the most important video processing problems. The continuing progress in image and video processing technology is making possible the development of new applications that were only a dream 10 or 20 years ago. Video processing technology has revolutionized the world of multimedia with products such as the Digital Versatile Disk (DVD), the Digital Satellite System (DSS), high definition television (HDTV), digital still and video cameras, and hard-disk recorders.

The transmission of video over heterogeneous networks for multimedia applications has recently become an area of active research. Applications, such as video streaming on the Internet, are becoming commonplace as a better understanding is obtained of the problems associated with delivery of video over packet networks.

1.1 Video Scalability

The objective of a video compression algorithm is to exploit both the spatial and temporal redundancy of a video sequence such that fewer bits can be used to represent the video sequence at an acceptable visual distortion. For example, it is frequently desired to transmit video over standard telephone lines, where data rates are typically restricted to 56,000 bits per second (bps). A video sequence with frame size of 176×144 pixels (a size commonly used for this application) at 30 frames per second (fps) and 3 bytes per pixel, would require 18.25 Mbps, making impractical the transmission of video without compression. For different applications, different resolutions, visual quality, and therefore, different data rates, are required.

The available bandwidth of most computer networks almost always poses a problem when video is to be delivered. A user may request a video sequence at a specific

data rate. However, the variety of requests and the diversity of the network may make it difficult for an image or a video server to predict, at the time the video is encoded and stored on the server, the video quality and data rate it will provide to a particular user at a given time.

Meeting bandwidth requirements and maintaining acceptable image quality simultaneously is a challenge. Rate scalable compression that allows the decoded data rate to be dynamically changed, is appealing for many applications, such as video streaming and multicasting on the Internet, video conferencing, video libraries and databases, and wireless communication. In these applications, the bandwidth available cannot be guaranteed due to variations in network load. When a video sequence is transmitted over a heterogeneous network, network congestion may occur, decreasing the quality observed by the user.

Consider the following scenario: A media provider digitizes, compresses, and stores news clips in a digital video library using MPEG-2 at 6 Mbits/sec (Mbps), and makes them available to the public. Most current video compression techniques and standards require that parameters, such as data rate, be set at the time of encoding. A problem exists if the server receives a request for the video sequence not at 6 Mbps, but at 4 Mbps. A solution to this problem would be to have the video server transcode the compressed bit stream. However, this is a computationally intensive task.

In general, a media producer faces the difficult task of providing content at different resolution (temporal, spatial, and/or rate) levels depending on the receivers' capability as well as possibly users' choice. One solution to this problem is to compress and store a video sequence at different data rates. The server will then be able to deliver the requested video at the proper data rate, given the network load and the specific user request. There are two problems with this approach: i) The need to store a sequence at various data rates introduces the added overhead of storage, duplicity and management of different sequences. ii) For real-time applications, it is impractical to have several encoders compressing the sequence at the same time. An alternative solution to this problem is to use a video coder that is capable of

dynamically selecting the decoded data rate. This is a very attractive solution for the flexibility it introduces to the system. This is known as "video scalability."

Video scalability is different from the concept of scalability used in networking. In this thesis, when we refer to scalability, we mean "video scalability." Most current compression techniques require that parameters, such as data rate, frame rate, and frame size, be set at the time of encoding, and are not easily changed. Video compression techniques that allow one to encode the sequence once and then decode it at multiple data rates, frame rates, spatial resolutions, and video quality are known as "scalable." The goal of scalable video compression is to author and encode a video sequence **once**, and decode it on **any** platform fed by **any** data pipe.

At the 1999 Picture Coding Symposium, a panel on "The Future of Video Compression" addressed the issue of scalability modes in video compression. The conclusion of the panel was that the following modes, in order of importance, needed to be addressed by the research community.

- Rate Scalability: where the receiver can request a particular data rate, either chosen form a limited set of rates (layered rate scalability) or from a continuous set of data rates (continuous rate scalability). Rate scalability is perhaps the most attractive mode for video streaming over packet networks such as the Internet.

- SNR (Signal-to-Noise Ratio) Scalability: where the receiver can request a particular level of quality, as determined by the degree of "fidelity" between the original and decoded frames. This degree of "fidelity" is commonly determined by the SNR, a metric that will be described in Section 2.1.

- Temporal Scalability: where the receiver can request a particular frame rate.

- Spatial Scalability: where the receiver can request a particular frame size.

- Content Scalability: which is the ability to determine the content of a scene at the decoder, and is part of MPEG-4.

An important question relative to scalability that impacts its use in streaming is how often can the compression parameters be changed. Therefore, we have:

- Static Scalability: when the compression parameters can only be set at the beginning of the encoding of the video sequence. This mode is useful in that a media producer would not have to store multiple copies of a given sequence on his server, but its limited in terms of its networking implications.

- Dynamic Scalability: when the compression parameters can be changed many times during encoding, that is, streaming, of the sequence. This could then be used by the network when congestion occurs to change the compression parameters on-the-fly.

The most useful scalable video compression techniques are the ones that can scale the data rate over several orders of magnitude. To achieve this goal, a combination of rate, spatial, temporal, and content scalability may need to be used.

1.2 Organization of This Thesis

In Chapter 2, we present the basic terms and concepts used throughout this thesis. We briefly overview current high and low data rate video compression standards. We discuss the use of the wavelet transform for image and video coding. Previous work in subband and scalable image and video compression is surveyed also in this Chapter. In particular, we are interested in rate scalable and wavelet-based compression techniques. Also in Chapter 2, we describe the use of digital signal processors for image and video processing, and error concealment techniques for digital video.

In Chapter 3, we describe the Color Embedded Zerotree Wavelet ($CEZW$) still image compression technique [1, 2, 3, 4]. $CEZW$ exploits the interdependence between color components to achieve a higher degree of compression. It is a basic component of the Scalable Adaptive Motion Compensated Wavelet ($SAMCoW$) video compression technique [2, 4, 5] which is also presented in Chapter 3.

Our contributions are presented in Chapters 4 and 5. Improvements to the $SAMCoW$ video compression techniques are introduced in Chapter 4. The aim of this

work is to improve the performance of *SAMCoW* at data rates of 64 kbits/sec (kbps) and below. Our work has focused on the use of advanced video coding techniques [6], preprocessing and postprocessing techniques for predictive error frames [7, 8], and a rate-distortion approach to encoding of predictive error frames [9].

In Chapter 5, we describe the implementation of real-time video processing applications on digital signal processors (DSP). We have implemented in real-time the error concealment technique described in [10, 11] on the Texas Instruments *TMS320C6201* (*'C6201*) DSP [12]. We also describe the implementation on the *'C6201* of wavelet filters for image decomposition and reconstruction. The implementation of *CEZW* on fixed-point DSPs is also discussed. Finally, in Chapter 6, we present concluding remarks and future work.

2. PRELIMINARIES

In this Chapter, we define the basic terms and concepts used throughout this thesis. We overview current high and low data rate video compression standards. We also describe the use of the Discrete Wavelet Transform (DWT) in image and video compression. Previous work in subband and scalable image and video compression is also surveyed in this Chapter. In particular, we are interested in rate scalable and wavelet-based compression techniques. Finally, we describe the use of digital signal processors for image and video processing, and error concealment techniques for digital video.

2.1 Video Basics

In this Section, we provide a brief overview of the basic principles and terms that are used throughout this thesis. We describe the different chrominance subsampling techniques that are used for different applications, the format of the video frames typically used for video transmission, and the method we use to quantify "fidelity" in a video sequence.

2.1.1 Chrominance subsampling

A digital image is represented in a color space with three components used to convey the color information [13]. There are three main color models: RGB, used in computer graphics; YUV, YIQ, or YCbCr, used in television systems; and CMYK, used in color printing [14]. In this thesis, we work with YUV images, the color space commonly used in color image and video compression.

In a YUV image, each pixel has three 8-bit values associated with it, one for each of the Y, U, and V components, as shown in Figure 2.1. The human eye is more sensitive to luminance (Y) information [15, 16, 17, 18], therefore the chrominance

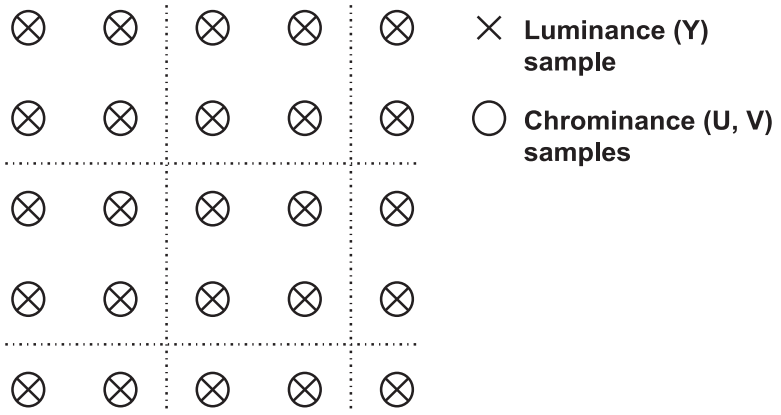


Fig. 2.1. In a YUV image, each pixel has three 8-bit values associated with it. This is referred to as 4:4:4 chrominance format.

components (U, V) can be subsampled while still preserving good viewable image quality. An added benefit of subsampling is the reduction of bandwidth needed for transmission. Subsampling is denoted in the format X:X:X, where the first digit represents the number of luminance samples, used as a reference and typically "4." The second and third digits are the number of chrominance samples, with respect to the number of Y samples. For instance, 4:1:1 means that for every four Y samples, there are one U and one V samples. An exception to this nomenclature is 4:2:0 chrominance subsampling, which has the same number of chrominance samples as 4:1:1, but indicates a different method for downsampling, as explained next.

An important question is what samples to discard when subsampling is done. In Figures 2.2, 2.3, 2.4, and 2.5, the 4:2:2, 4:1:1 (two different methods), and 4:2:0 chrominance subsampling factors, respectively, are shown [14, 19, 20]. Depending on the application, chrominance subsampling can be done differently, as it is the case for 4:1:1. In one method, the chrominance components are subsampled by a factor of two in both the horizontal and vertical directions. In other method, they are subsampled by a factor of four in the horizontal direction, but not subsampled in the vertical direction. The choice of subsampling method depends on the application. 4:2:0 has the same number of samples for chrominance components as 4:1:1; however,

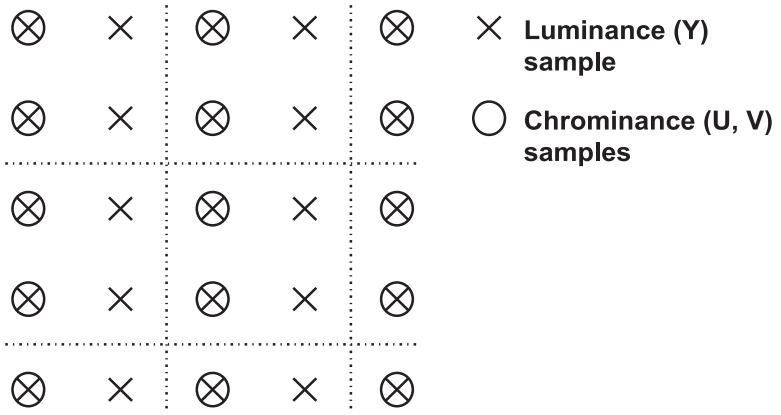


Fig. 2.2. 4:2:2 chrominance subsampling. For every four luminance (Y) samples, there are two U and two V samples.

the location of the samples with respect to the luminance component is different, as shown in Figures 2.3, 2.4, and 2.5.

2.1.2 Image quality

In this thesis, we use the term "visual quality" of an image or video sequence to refer to "fidelity," or how close the decoded image is to the original as perceived by an observer. There is not a simple and easily computable metric that will accurately predict how an observer will perceive a decoded image. However, we will use the peak signal-to-noise ratio (PSNR), based on the mean-squared error (MSE), as our "quality" measure. This measure, while unsatisfactory, does track "quality" in some sense. However, we believe that the ultimate indication of image quality is the actual observation of the decoded images. The PSNR of a YUV image is obtained by using the following equation:

$$PSNR = 10 \log \frac{255^2}{(MSE(Y) + MSE(U) + MSE(V))/3} \tag{2.1}$$

2.1.3 Frame formats

The size of the frames, that is their width and height, used in digital video are also standardized to facilitate equipment interoperability. The Common Intermediate Format (CIF) was created to facilitate interoperation between the NTSC and PAL television formats. CIF is a non-interlaced format, with a frame size of 352 pixels per

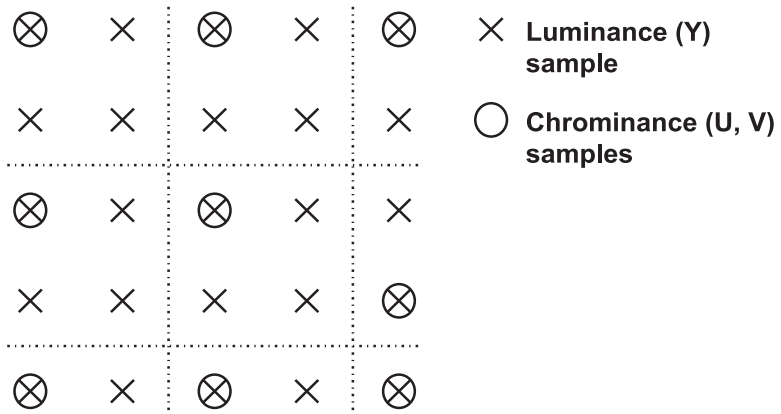


Fig. 2.3. 4:1:1 chrominance subsampling. For every four luminance (Y) samples, there are one U and one V samples. In this case, the chrominance components are subsampled by a factor of two in both the horizontal and vertical directions.

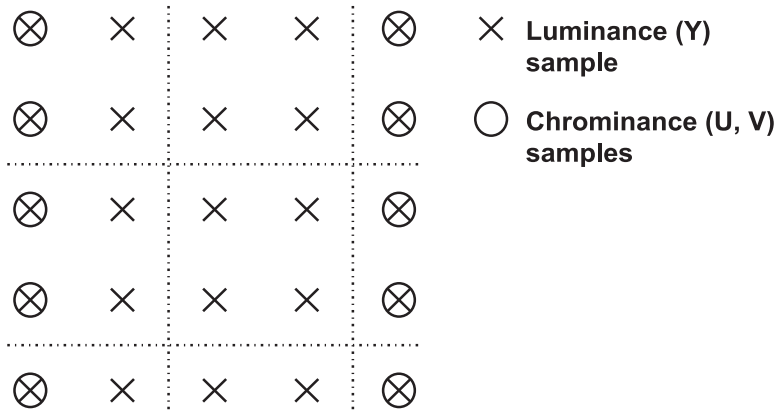


Fig. 2.4. 4:1:1 chrominance subsampling. For every four luminance (Y) samples, there are one U and one V samples. In this case, the chrominance components are subsampled by a factor of four in the horizontal direction, but not subsampled in the vertical direction.

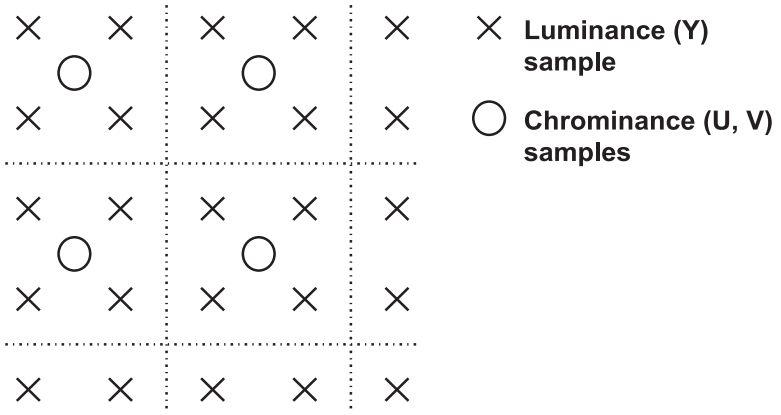


Fig. 2.5. 4:2:0 chrominance subsampling. For every four luminance (Y) samples, there are one U and one V samples, as in 4:1:1. However, the location of the samples with respect to the luminance component is different than 4:1:1.

line, 288 non-interlaced lines per frame at 30 frames per second [14, 21]. To convert a frame from an NTSC format (525 lines per field at 30 frames per second) to a CIF format only a line-number conversion is needed, since the frame rate is the same. To convert a frame from a PAL format (576 active lines at 25 frames per second) to a CIF format, only a frame rate conversion needs to be performed, because CIF has half the active lines than PAL. Other frame formats are also used; some of them are derived from CIF. They are shown in Table 2.1.

A YUV 4:1:1 frame is organized into "macroblocks," a 16×16 array of non-overlapping luminance (Y) pixels together with one 8×8 block of spatially corresponding pixels for each of the chrominance components (Cb and Cr). A macroblock of luminance pixels is composed of four 8×8 blocks of pixels. Most current video compression algorithms operate on 8×8 blocks pixels, because it provides a good trade-off between moderate computational requirements and efficient temporary redundancy reduction [21], as we discuss later in this Chapter.

2.2 MPEG Video Compression

The development of international standards for image and video compression that allow interoperability among systems from different manufacturers is one of the rea-

Table 2.1
Resolution of the luminance component for different frame formats.

Frame Size (Y)	Number of pixels/line	Number of lines
Sub-QCIF	128	64
QCIF	176	144
CIF	352	288
4CIF	704	576
16CIF	1408	1152
SIF	352 (360)	240
CCIR 601	720 (704 active)	480

sons digital video technology has become an integral part of a number of multimedia applications [22]. In January 1988, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) established the Moving Picture Experts Group (MPEG), with the mandate to develop standards for the coding of moving pictures for storage on digital media. Formally known as ISO/IEC JTC1/SC29/WG11, MPEG has produced standards, such as MPEG-1 [23] and MPEG-2 [24], that define common formats for coding digital video and associated audio. The new standard for audio-visual coding in multimedia applications, known as MPEG-4 [25], is a departure from MPEG-1 and MPEG-2, which are based on reducing spatial and temporal redundancy in a video sequence using frame-based techniques. MPEG-4 performs content-based coding to provide functionalities common in multimedia applications. Other video communication standards have been developed for specific applications, such as H.261 [26] and H.263 [27] for low data rate video conferencing.

In this Section, we present a brief overview of the aspects of the MPEG-1, MPEG-2, MPEG-4, and H.263 standards that are more relevant to our research. It is important to point out that these standards do not define the implementation of the

encoder; rather, they define the syntax of the encoded data stream, and hence the decoder, leaving the adopters ample room for product differentiation. In the standardization process, a software version of the set of techniques proposed for inclusion in the standard is developed. This software is known as "verification model" (VM). The VM is tested and iteratively improved until an agreement is reached that the performance is acceptable. Then, the set of techniques is said to have reached "Standard" status.

2.2.1 MPEG-1

MPEG-1, formally known as "Information Technology - Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbits/s" [23], was designed for the storage and distribution of digital video with quality comparable to VHS video-tape systems. It consists of five parts [19]:

- Part 1. Systems: Definition of the structure of the MPEG-1 bit stream, including timing information, description of multiplexing/demultiplexing of a video data stream and one or more audio data streams, and synchronization information.

- Part 2. Video: Complete specification of the methodology for encoding and decoding a video sequence, including layers, coding order, coding techniques and models, quantization and encoder decision strategies.

- Part 3. Audio: Description of the syntax of the audio data stream, the encoding and decoding process, and tests for compliance.

- Part 4. Conformance Testing: Describes the procedure to check for conformity in implementations of the standard.

- Part 5. Software Simulation: A software implementation of the standard, used as a reference model for MPEG-1.

MPEG-1 was designed for non-interlaced video sources, common in computer graphics and displays. Although it can be used with interlaced video streams, such as

television signals, its compression efficiency is smaller than other techniques due to its non-interlaced frame-based processing. In addition, its performance has been tailored for certain pictures sizes other than those common in television. Other standards, such as MPEG-2, provide better performance for applications where the video source is interlaced.

2.2.2 MPEG-2

MPEG-2 [24] was proposed jointly by ISO/IEC and ITU-T to target all-digital transmission of broadcast-quality video, typical of a wide class of applications ranging from NTSC/PAL quality signals to near studio quality to digital high-definition TV (HDTV). The Digital Satellite System (DSS) and the Digital Versatile Disk (DVD) [28], are examples of consumer applications that use MPEG-2 as their coding strategy. MPEG-2 consists of eight parts [20]:

- Part 1. Systems: Its description is similar to MPEG-1 Systems. Additions have been made to improve error resilience, its ability to support ATM networks, and the capacity of handling more than one independent programs in the same stream.

- Part 2. Video: Maintains the syntax from MPEG-1, adding functionalities such as: extensions for scalability (in the form of multiple resolutions) to appeal a wide variety of applications. Definition of profiles to distinguish classes of applications with different scalability requirements (non-scalable, spatial, temporal, and SNR scalable). Definition of levels in a hierarchical structure to subdivide applications in the same profile but with different range requirements (frame size, frame rate, and data rate).

- Part 3. Audio: an extension of MPEG-1, adding multichannel input, multilingual support, lower data rates, and additional sampling rates. Two modes: Backward compatible and Non-Backwards Compatible with MPEG-1.

- Part 4. Conformance testing: Similar in purpose to that of MPEG-1.

- Part 5. Simulation software: Similar in purpose to that of MPEG-1.

- Part 6. Digital Storage Media - Command and Control (DSM-CC): Defines a high level transparent interface between the user and the DSM where the MPEG-2 data is stored. Playback, editing, and storage are examples of the functionalities provided by DSM-CC.

- Part 7. New audio formats: Non-backward compatible (NBC) with MPEG-1. Establishes higher data rates necessary for high-quality audio, while maintaining characteristics from Part 3.

- Part 9. Real-time interface: Imposes restrictions on Part 1 Systems for real-time delivery of an MPEG-2 data stream in time-sensitive applications.

Part 8, 10-bit video has been dropped from the standard because applications that would benefit from it are too specialized (production of high-quality material and specialized visual effects, for example). The small number of systems capable of processing 10-bit video was another reason for suspending the standardization work in this area.

MPEG-2, formally known as ISO/IEC 13818 and ITU-T Recommendation H.262, targets data rates between 1.5-60 Mbits/s, although it is not restricted to this range. Part 2, Video, defines five profiles (Simple, Main, SNR scalable, Spatially scalable, and High) that can operate at four different levels (Low, Main, High-1440, and High). Although the range of values for each profile/level combination varies, the syntax of the data stream remains the same. Main Profile at Main Level (YCbCr, 4:2:0, up to 15 Mbits/s, 29.97 and 25 frames/s, at different spatial resolutions) offer the functionalities and complexity that meet the needs of the majority of current applications. Typical resolutions are: 720×480 (CCIR-601), 640×480 (square pixel), and 352×288, among others.

Both MPEG-1 and MPEG-2 exhibit high asymmetry between the complexity of the encoder and the decoder. The encoders are many times more complex than the

decoders and, therefore, more expensive. Real-time encoders are very specialized and costly systems because of the computational power that they must provide in order to meet the time constraints. However, real-time encoding is not necessary for a number of applications. In contrast, every player needs a real-time decoder. Therefore, in order to make the cost of decoders as low as possible, most of the computational load is in the encoder side.

In the case of video conferencing applications, the requirement for real-time processing is for both the encoder and the decoder. This is referred to as "symmetric coder." For that reason, new coding techniques were developed to address this class of applications. They are presented in Section 2.3.

2.2.3 MPEG-4

MPEG-4 [25] became an International Standard in December 1998. It represents a different approach to video compression, in what has been called "second generation" video coding techniques [29, 22]. In these techniques, concepts from image analysis, such as segmentation and model based compression, are used to obtain higher compression. The MPEG-1 and MPEG-2 standards use a variety of techniques to reduce the spatial and temporal redundancy in a video sequence. However, there is no provision for the incorporation of new functionalities common in other multimedia applications. For example, when browsing a document on the Internet, it is possible for the users to interactively select the information that interests them the most. MPEG-4 recognizes that in the future, distinctions between broadcast television and computer-based multimedia applications will likely disappear. In this new standard, algorithms are being used to allow interactivity between the user and the application, universal accessibility, and high degree of compression.

The basic unit of compression is an audio-visual object (AVO) [30, 31]. An AVO represents objects of interest in a picture. A person, the scenery, and background music are recognized as primitive AVOs. In general, a AV object is of arbitrary shape. The composition of primitive AVOs creates compound AVOs. This distinction gives the user the ability to identify and selectively decode, reconstruct and manipulate

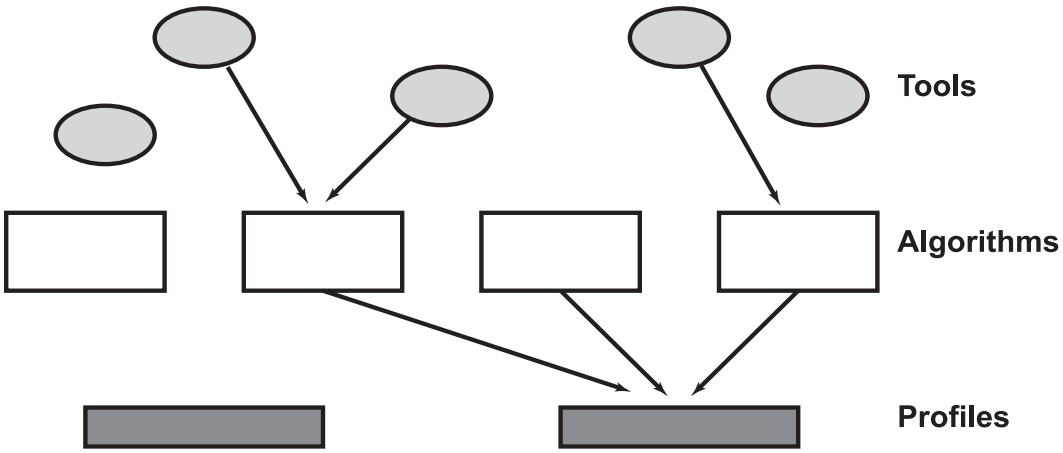


Fig. 2.6. Hierarchical approach for operations (coding and manipulation) on audio-visual-objects (AVO).

AVOs of interest. This is known as content-based scalability. It is important to note that in MPEG-4, it is possible to access an AV object without the need to decode the entire scene. This feature is appealing for bandwidth-constrained applications.

In a multimedia environment, audio and video of different types coexist, and techniques need to be flexible enough to accommodate the requirements of each class of AVO. Therefore, MPEG-4 provides a variety of compression techniques in the form of a toolbox. Natural audio and video, synthetic audio, 2-D and 3-D synthetic (computer generated) video, are some of the possible sources of AVOs in an MPEG-4 environment. These toolboxes contain specialized tools for natural and synthetic scenes. Algorithms operating on a video sequence, are able to select the most appropriate tool for the current audio-visual scene. Standards such as MPEG-1 and MPEG-2 could be some of the tools used by algorithms to compress video sequences and associated audio. Profiles are defined to address the specific requirements (i.e., low complexity, low delay) of certain applications. This hierarchical approach is shown in Figure 2.6.

In an MPEG-4 system, AV objects (including their spatial and temporal relationships) are compressed, multiplexed and transmitted from a source to the MPEG-4 decoder. Transmission takes place over a variety of networks with different qualities of

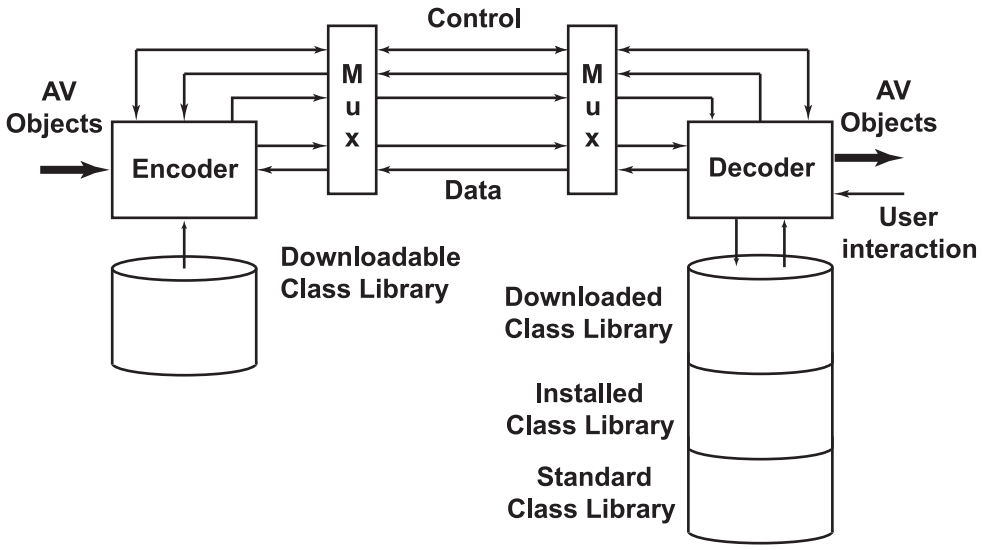


Fig. 2.7. High-level MPEG-4 system diagram.

services, and some may be more error-prone than others. Therefore, error resilience measures are used for reducing the sensitivity of the compressed stream to transmission errors. The decoder demultiplexes and decodes the AV objects. Using the relationships and context information sent with the objects, they are composited and presented to the user. The end user can interact with the objects, thus interaction information may need to be sent to the source, as illustrated in Figures 2.7 and 2.8.

A software environment, known as MPEG-4 Systems & Description Language (MSDL), is used by MPEG-4 to construct a model of an audio-visual scene [31]. MSDL defines the interfaces between coding tools, how they are combined, and a mechanism to download new tools. MPEG-4 is an open standard, not a fully defined algorithm. New tools can be added as they are developed, providing MPEG-4 with great flexibility to adapt to changing needs in multimedia applications. MSDL provides a description of the bit streams (Elementary Streams), corresponding to the AVOs, as well as how and which decoding tools are used to decode the data. The decoder can dynamically request missing tools from the encoder. User interaction is facilitated by MSDL, conveying to the source the requests made in the decoder side.

The requirements for MPEG-4 can be summarized as follows [32]:

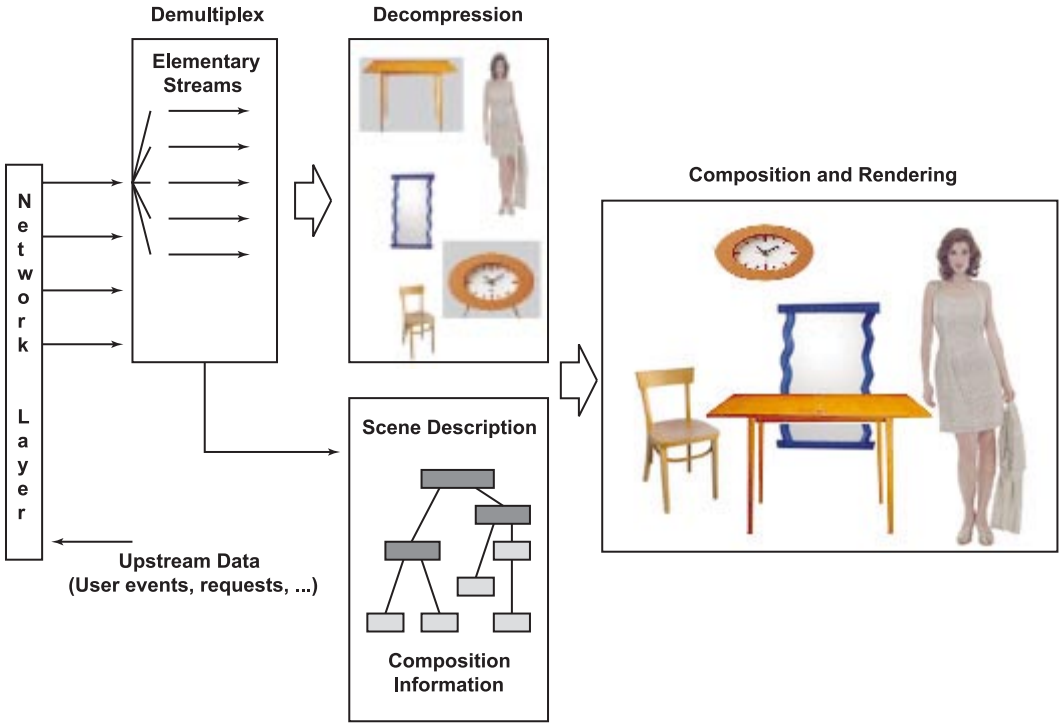


Fig. 2.8. Content-based approach of the MPEG-4 video standard. Audio-visual objects are composed and presented to the user by the decoder, according to a scene description.

- Content-based interactivity: Frame-based (such as MPEG-1 and MPEG-2) compression, as well as content-based access, manipulation and bit stream editing are supported. Compositing of synthetic and natural data, and provisions for interactivity also are supported. Temporal random access, within certain time limits, is improved.

- Compression: MPEG-4 provides better subjective visual quality than current and emerging standards at comparable data rates, in particular at low data rates (64 kbps and lower). Requirements are made to support the coexistence of diverse network environments, and constrained decoder resources.

- Universal access: Provides an array of error robustness capabilities, even under severe error conditions, such as long error bursts. Particularly targeted are low data rate applications, which would be affected the most by this type of noisy environments.

- Content-based scalability: Fine granularity for quality (temporal and spatial resolution), complexity, and content scalability is achieved by MPEG-4.

2.3 Low Bit Rate Video Compression

Applications such as video conferencing and video telephony have become increasingly popular in recent years, due to a combination of factors. These include the emergence of international standards that allow systems from different vendors to be compatible, the increasing data rates in telecommunication networks, and the availability of more powerful processors. However, some video applications operate in environments where the use of video coding standards, such as MPEG, at full resolution is impractical or not possible. These applications typically make use of phone or wireless networks, where the available bandwidth may not be enough, or it may be too expensive to afford. The computational load also needs to be taken on account, because encoding and decoding a stream needs to be performed in real-time. In these scenarios, a trade-off needs to be made between the quality of the compression and

the time available for coding. In video conferencing and videophone applications, the compressed video data rate is typically restricted to be 1 Mbps or below. Frame sizes and color formats may be restricted also.

2.3.1 H.261

Several international organizations have proposed standards that have found limited support from vendors. Recommendations H.120 and H.130 for video conferencing at data rates close to 2 Mbps, were proposed by the International Telephone and Telegraph Consultative Committee (CCITT) in the early 1980's. These standards were not widely accepted by manufacturers due to limited signal quality. In the late 1980s, the ITU-T (International Telecommunications Union, Telecommunication Standardization Sector) (formerly CCITT) developed Recommendation H.320 [33] for the transmission of non-telephone signals for video telephone systems using ISDN circuits. H.261 [26] is the video coding standard that is part of H.320, which targets compressed video data rates of p$\times$64 kbps (p is an integer between 1 and 30).

The color space used by H.261 is YCbCr with 4:2:0 chrominance subsampling. Several of the elements in H.261 compression are similar to the ones used in MPEG. In this thesis, we investigate the compression of video streams at data rates lower than 64 kbps, thus we describe another standard for video conferencing, known as H.263, which targets these data rates.

2.3.2 H.263

The ITU Recommendation H.324 [34], ratified in 1995, establishes standards for low data rate multimedia communications over ordinary telephone lines, known as General Switched Telephone Network (GSTN). A part of H.324, the H.263 standard [27] is a video coding algorithm, based on H.261, that provides better picture quality than H.261 at a complexity comparable to that of H.261 [21]. No constraints on the target data rate are part of the Recommendation; the equipment or the network imposes the restrictions. The encoder and decoder block diagram is shown in Figure 2.9. The color space used in H.263 is YCbCr with 4:2:0 chrominance subsampling as in H.261.

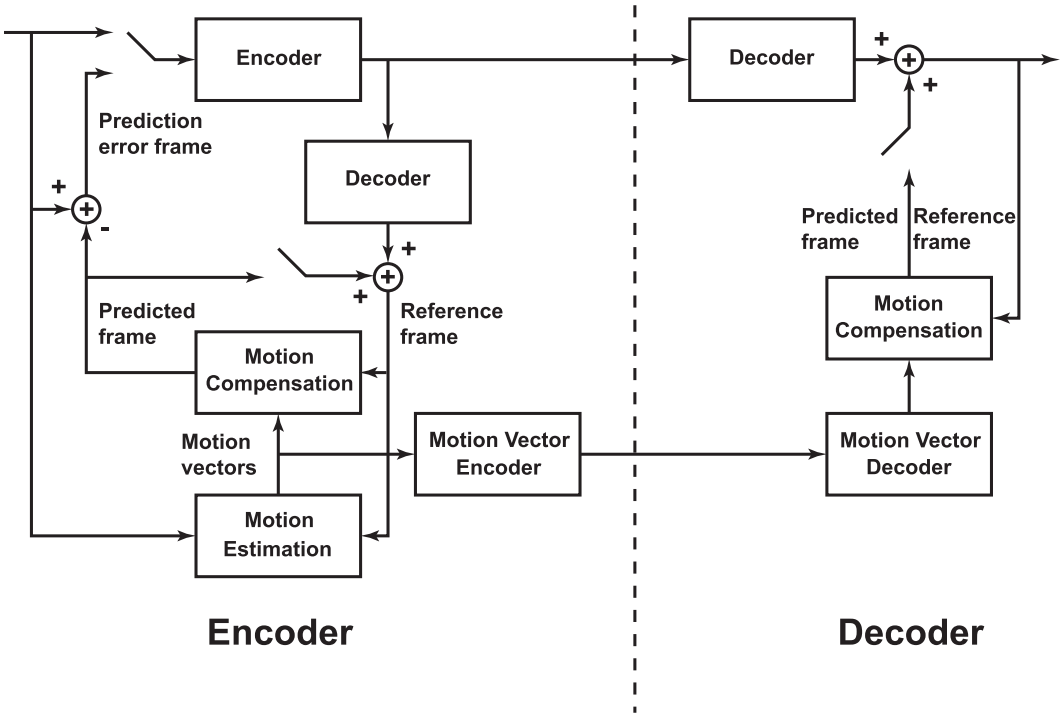


Fig. 2.9. H.263 video coder block diagram.

The encoder accepts frames in Common Intermediate Format (CIF). The frame size can be a multiple of a CIF size frame, as described in Table 2.1 for the luminance component. H.263 encoders must support sub-QCIF (SQCIF), QCIF, or both. H.263 decoders must support SQCIF and QCIF. CIF and larger picture sizes are optional for encoders and decoders.

Several compression techniques in H.263 are used in H.261 as well as in the MPEG standards, therefore the following description is, in general, applicable to them too.

To achieve significant compression efficiency, temporal redundancy between pictures needs to be exploited. That is, a frame is selected as a reference, and subsequent frames are predicted from the reference. This is done after realizing that pictures in a video sequence are taken at small intervals of time. Therefore, adjacent pictures have high temporal correlation, especially in simple scenes typical of video conferencing applications. This compression technique is known as interframe coding. Interframe coding is used extensively in video compression.

Interframe coding works well with translational motion of rigid objects. However, for scene changes, zooming, and rotational motion, interframe coding results in poor picture quality because temporal correlation is low. Therefore, a different coding model is used, where spatial correlation between adjacent pixels is exploited. This technique is known as intraframe coding. Intraframe and interframe coding are the basis of H.263 and the MPEG-1 and MPEG-2 standards.

Block Transform Coding: The Discrete Cosine Transform

In an intracoded frame, known as an "I" frame, each block of pixels in a macroblock is transformed into another domain using a two-dimensional discrete cosine transform (DCT) [35, 36]. The transformation of the block results in a discrete set of basis blocks, where the correlation and redundancy between pixels in the block is reduced [37]. The DCT is a unitary transform, that is, the transformation preserves the energy of the signal. Unitary transforms pack a large portion of the energy of the image into relatively few components of the transform coefficients. When the transform is applied to a block of pixels that are highly correlated, as is the case in a block of an image, the transform coefficients tend to be uncorrelated.

The Karhunen-Loève transform (KLT) [38, 39], is optimal in many applications, but it depends on the statistics of the image, and its computational cost is high. The DCT is not an optimal transform in terms of decorrelation and energy compaction properties. However, it presents a good trade-off between computational complexity and the above properties. The 2-D DCT $F(\mu,\nu)$ for an 8×8 block $f(x,y)$ is given by [19]:

$$F(\mu,\nu) = \frac{C(\mu)}{2}\frac{C(\nu)}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} f(x,y) cos\left[\frac{(2x+1)\mu\pi}{16}\right] cos\left[\frac{(2y+1)\nu\pi}{16}\right] \qquad (2.2)$$

where

$$C(\mu) = \frac{1}{\sqrt{2}} \quad if \quad \mu = 0$$
$$C(\nu) = \frac{1}{\sqrt{2}} \quad if \quad \nu = 0$$

$$C(\mu) = 1 \quad if \quad \mu > 0$$

$$C(\nu) = 1 \quad if \quad \nu < 0$$

In the decoder, the block is reconstructed by taking a 2-D inverse DCT of the transform coefficients obtained above. The 2-D inverse DCT is given by the following equation, with $C(\mu)$ and $C(\mu)$ as defined above:

$$f(x,y) = \sum_{\mu=0}^{7} \sum_{\nu=0}^{7} \frac{C(\mu)}{2} \frac{C(\nu)}{2} F(\mu,\nu) cos \left[\frac{(2x+1)\mu\pi}{16} \right] cos \left[\frac{(2y+1)\nu\pi}{16} \right] \qquad (2.3)$$

The 2-D DCT is a separable transform. Therefore, a 1-D DCT can be applied first to each row of pixels in a block. Then, the 1-D DCT is applied to each column. This decomposition facilitates the computation of the transform, an important consideration for low data rate applications. Fast algorithms for computing the DCT exist [40, 41, 36], making its use attractive. For applications where performance needs to be maximized, VLSI implementations are also available [42, 43].

H.263 does not define the implementation of the DCT. Instead, the accuracy of the reconstruction, that is, the accuracy of the inverse DCT, must meet certain criteria. This is because small changes in the implementation of the inverse DCT can produce significant picture artifacts in the reconstructed images [21].

Due to the compaction property of the DCT, lower order coefficients contain most of the energy of the data. In fact, most of the higher order coefficients have a value of zero, or very close to zero. This means that they do not have to be transmitted to approximately reconstruct a block in the decoder, since their contribution is insignificant when compared to the lower order coefficients.

The coefficient $F(0,0)$ of the DCT in Equation (2-2), is referred to as DC coefficient, because it represents the average of the pixels in the block. All the other coefficients are known as AC coefficients, because they represent the weight of the basis functions in which the image is decomposed.

After taking the 2-D DCT of a block, the resulting AC coefficients are scanned in a zig-zag pattern (see Figure 2.10), which results in an ordering by increasing

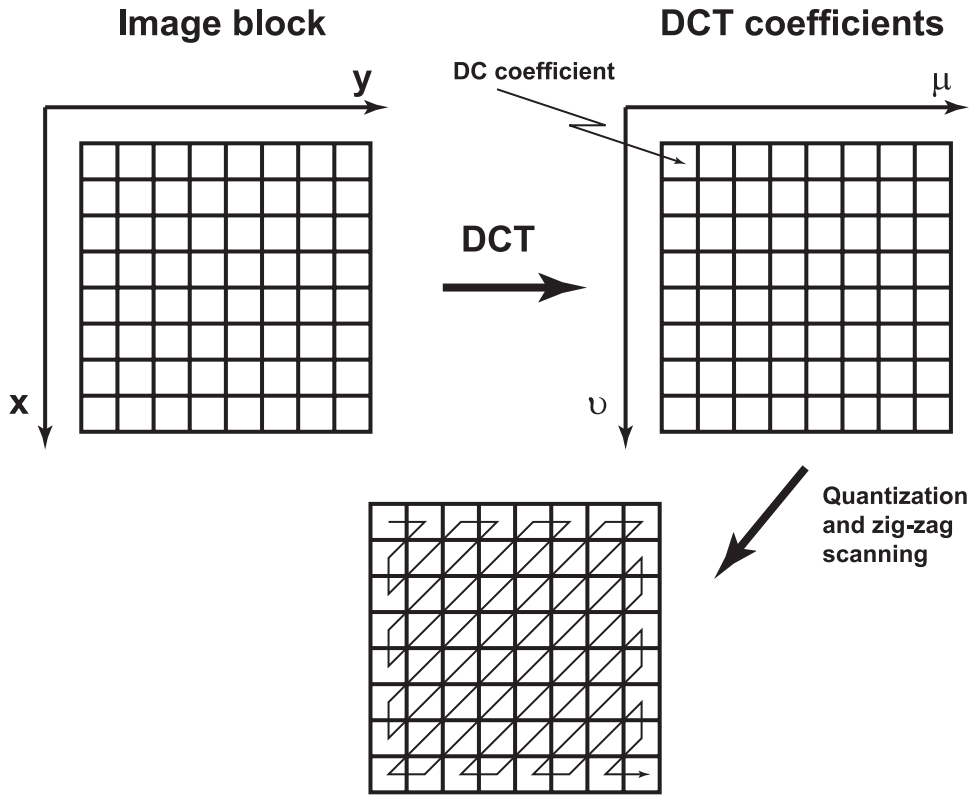


Fig. 2.10. Scanning order of 2-D AC DCT coefficients.

frequency. Because many of the coefficients are zero, this results in long sequences of zeros, which can be efficiently encoded using run-length coding.

Two other important properties of block transforms are exploited:

- The accuracy of representing non-zero coefficients can be reduced without significantly impacting the reconstruction of the block. Different quantization scales are used for the DC coefficient of the DCT and the rest of the coefficients. Because human perception is more sensitive to large luminance areas, the DC coefficient is losslessly coded.

- The probability distribution of the coefficients is highly non-uniform, which is exploited by entropy coding [44, 45]. After the DCT coefficients are quantized, they are coded with a 3-D variable length coder (VLC): *run*, *level*, and *last coefficient*, improving compression efficiency. "Run" is the number of zero co-

efficients between two non-zero coefficients. "Level" is the magnitude of the non-zero coefficients. "Last coefficient" indicates that there are no more non-zero coefficients in the block.

Quantization is the lossy step of the encoding process being described, which causes the reconstruction to be imperfect. The goal of block-based compression is to minimize the distortion in the decoded image subject to a certain bit budget. Quantization is also used as a means to control the data rate of the coded image. Larger step size is used to quantize the coefficients coarsely, therefore reducing the bits needed for representing them. A smaller step size produces a finer scale that represents the coefficients more accurately, but requires more bits to represent each coefficient.

When the resulting data is received in the decoder, the data is put through an entropy decoder, followed by a run-length decoder. The zig-zag scanning is reversed, the resulting matrix of coefficients is multiplied by the corresponding quantization step sizes, and an inverse DCT of the resulting coefficients is performed to reconstruct the block.

Block processing yields good results when the bits allocated to encoding the frame is enough to guarantee a good reconstruction in the decoder. However, if the bit budget is limited, as in low data rate applications, blocking artifacts may be evident in the reconstructed frame. This problem can be reduced by performing pre- and post-processing on the sequence. However, the visual quality can only be improved to a certain degree, and additional processing requires additional resources from the encoder and decoder. Another approach to solving this problem is to use a non-block-based transform, such as the wavelet transform. This approach is used in *SAMCoW*, as described in Chapter 3.

Predictive Coding

In interframe coding, the temporal redundancy of a video sequence is reduced by using motion estimation and motion compensation techniques. There are two types of frames used in interframe coding: predictive-coded (P) frames, which are coded

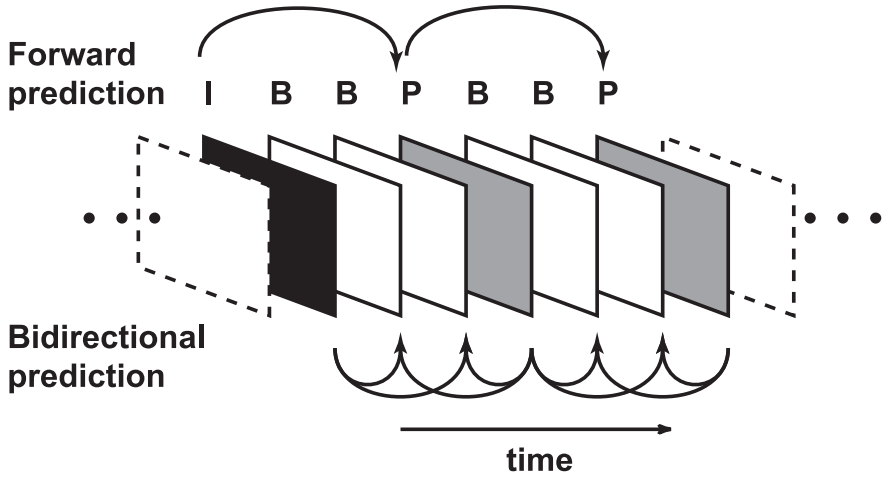


Fig. 2.11. The temporal relation between I, P, and B frames.

relative to a temporally preceding I or P frame; and bidirectionally predictive-coded (B) frames, which are coded relative to the nearest previous and/or future I and P frames. This is shown in Figures 2.11, 2.12, and 2.13. Macroblocks in a B frame may be coded based on previous frames, future frames, both, or neither (that is, the block is intracoded). It is important to note that in the default mode of H.263 there are no B frames. However, a variation of B frames is implemented in one of the advanced modes.

Since I and P frames are used as references for other frames, it is important to preserve their quality. On the other hand, B frames are not used as reference, therefore they can tolerate a lower quality to obtain a reduction in the number of bits needed for encoding. Thus, a larger quantization step can be used. Typically, an intracoded frame requires rates close to 1 bit per pixel (bpp); a predictive frame requires 0.5 bpp; and a B frame can be encoded using 0.25 bpp [46]. A trade-off exists between lower computational complexity, higher data rate, and comparatively higher quality for I frames; medium data rates, medium complexity and medium quality for P frames;

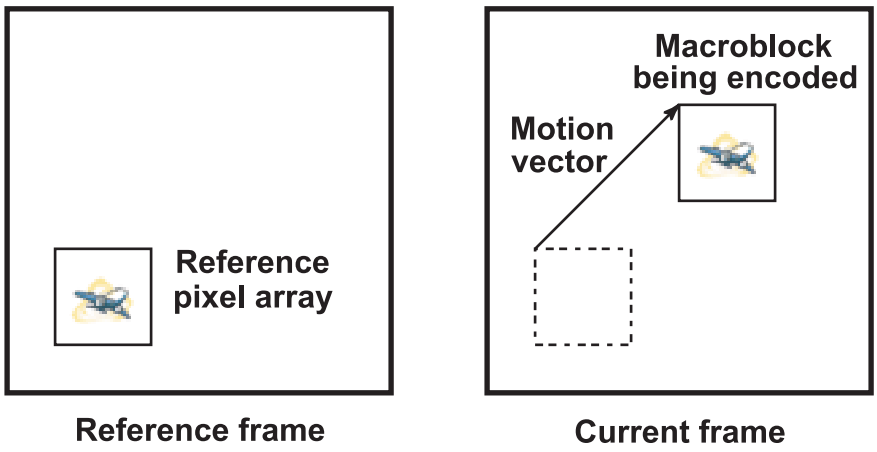


Fig. 2.12. Forward motion-compensated prediction.

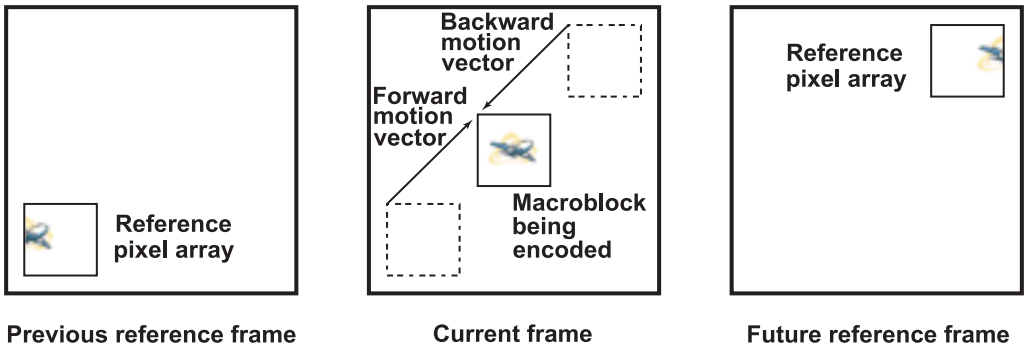


Fig. 2.13. Bidirectional motion-compensated prediction.

and higher complexity, lower data rate, and comparatively lower quality for B frames. A compromise is typically found in using a mix of I, P, and B frames in what is known as a "group of pictures" or GOP. A GOP commonly starts with an I frame, followed by P and B frames, although in some applications no B frames are used. The number of frames in a GOP is application dependent. In broadcast quality TV, a common GOP size is 12 or 15 frames, with a common GOP structure or "display order" being, for example, "IBBPBBPBBPBB." Note that the "encoding order" of the frames in a GOP is different than the "display order." In the previous example, the "encoding order" would be "IPBBPBBPBBI*BB" in order to use bidirectional prediction for the B frames. The I* frame belongs to the next GOP.

Motion estimation

Motion estimation is the technique used to determine the movement of a macroblock from the reference frame to the current frame. Motion of a macroblock in the current picture with respect to a reference picture, is estimated by searching for the macroblock in the reference picture that "matches" it the closest. To reduce the distortion between the decoded and the original picture, the encoder uses a reconstructed reference frame to perform motion estimation, as this is the actual frame that the decoder uses in the reconstruction (see Figure 2.9). It is important to note that the variable length coder (VLC) encoder is not included in the loop because this step is lossless. As shown in Figure 2.12, in forward prediction, one motion vector per macroblock is obtained. For bidirectional prediction, two motion vectors are found, as shown in Figure 2.13. The search is conducted for the luminance component only, although schemes for incorporating the chrominance components have also been investigated [47, 19]. The horizontal and vertical components of the motion vector can be integer or half-integer values within the range [-16, 15.5]. The matching criterion is not specified by H.263. In practice, minimum mean square error (MSE), minimum mean absolute difference (MAD), or sum of absolute difference (SAD) can be used as a cost function. If the cost function exceeds a predetermined threshold value, the macroblock is intracoded.

Searching for the closest match is a computationally expensive task, and may impact the real-time performance of the algorithm. The implementation of motion estimation is not part of the H.263 standard. Therefore, encoders may implement the search strategy that better fits their system. Several methods have been proposed [21] to reduce the computational cost of this task. Among others:

- Full search: Simplest method that guarantees that the minimum of the cost function will be found. A full search is conducted on a limited region of the reference pixel. Intuitively, spatially close pixels are more likely to be correlated than pixels far away, especially for head-and-shoulders-type video scenes typical of video conferencing applications. H.263 limits the value of the horizontal and vertical components of the motion vector, but the shape of the region is not specified by the standard. The complexity of full search is extremely high, and it is rarely used.

- Two-dimensional logarithmic search: Similar to binary search, where the search range is gradually reduced based on a matching criterion. For a 16×16 macroblock, the search is conducted only at nine locations: [0, 0], [±8, 0], [0, ±8], [±8, 8], and [8, ±8], that is, at a distance of 8 pixels from the center of the search region. Using the best match as the center of the new search region, a second search is conducted at nine points at a distance of 4 pixels from the center. The procedure is repeated two more times. The complexity of this approach is significantly less than full search.

- Hierarchical motion estimation: This is a bottom-top approach, where the motion estimation is first done in a low-resolution version of the current picture and the reference picture. Once the best match has been found, it is used as the starting point for the search in the higher resolution picture. Several levels of resolution can be used.

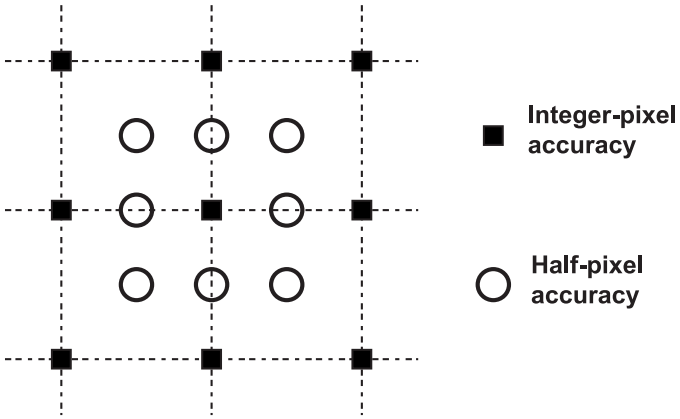


Fig. 2.14. Sampling grid showing half-pixel accurate positions.

Half-pixel accurate motion estimation

H.263 allows the motion vector to take half-integer values (half-pixel accuracy), as shown in Figure 2.14. The half-pixel accuracy motion vector is found using bilinear interpolation of the values of adjacent pixels in the integer sampling grid. The perceptual quality of the resulting picture using half-pixel accuracy is better than using integer-pixel accuracy. which uses integer-pixel accuracy, because of the filtering effect of the interpolation step [48]. Recently, schemes such as H.26L, have considered the use of variable block size in motion estimation [49].

Motion compensation

In motion compensation, a predictive frame is constructed from the motion vectors obtained for all macroblocks in the frame, by replicating the macroblocks from the reference frame at the new locations indicated by the motion vectors. This technique is known as Motion Compensation. The difference between the values of the predicted and the current frames, known as predictive error frame (PEF), is then encoded using the same procedure as for an intracoded frame. The frame obtained by adding the predictive frame to the PEF is known as the reconstructed frame. The energy of the PEF is low, thus many coefficients are zero, reducing the number of bits needed to encode the frame. PEFs are also known as Motion Compensated Residue Images [50], Motion Compensated Frame Differences [51], and Displaced Frame Differences [52].

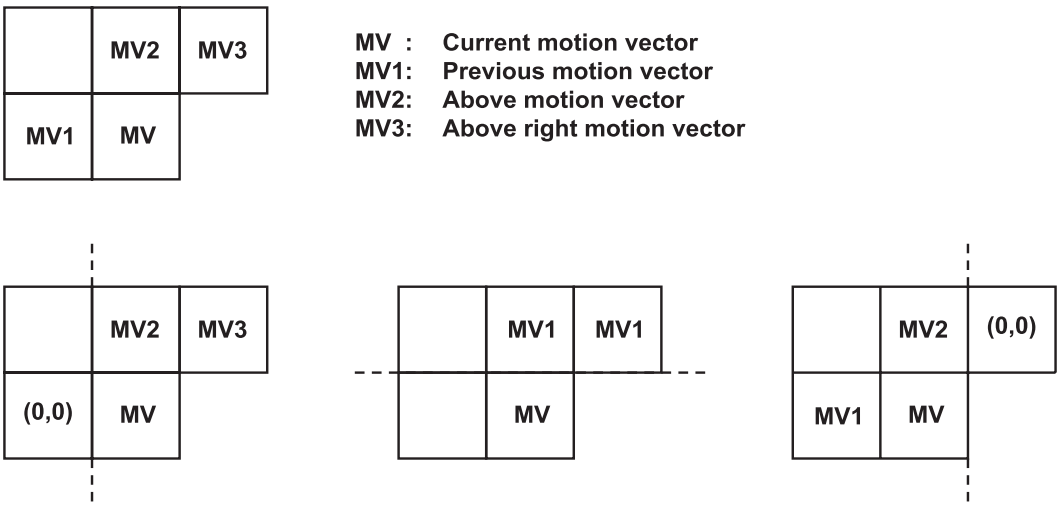


Fig. 2.15. Motion vector prediction.

The value of the motion vector obtained from the search is subtracted from the value of a predictor motion vector. The difference is then encoded and transmitted along with the PEF. The predictor is the median of the motion vectors from three previously coded macroblocks as shown in Figure 2.15. The predictor is calculated separately for the horizontal and vertical components.

Rate control, that is, producing a bit stream whose data rate is as close as possible to the target data rate, can be achieved by controlling the various parameters in the encoder. Examples of these parameters are: Select a finer or coarser quantizer to vary the data rate used while encoding the transform coefficients; B frames can be encoded or skipped; lower or higher prediction error can be tolerated.

H.263 Advanced Modes

At a cost of higher complexity, an H.263 encoder can use several advanced modes to improve picture quality or reduce data rate [53]. These modes are:

- Unrestricted motion vectors (UMV) mode: Motion vectors are restricted to point to pixels inside the reference frame. This mode allows the encoder to generate motion vectors that point to positions outside the reference frame, a useful feature when objects enter and exit the frame. This mode is shown in

Figure 2.16. The quality of the encoded frame is significantly improved. When a reference to a pixel outside the frame is made, the closest boundary pixel is used. The range of the motion vectors is extended to [-31.5, 31.5]

- Advanced prediction mode: UMV mode is activated. In addition, the following modes can also be used:

  - Four motion vectors per macroblock: Each block in a macroblock generates its own motion vector, which is differentially encoded with a predictor similar to that used by the default mode.

  - Overlapped block motion compensation (OBMC): Performed at the block level of the luminance (Y) component, the value of each pixel in a predicted block is the weighted sum of three prediction values: the value predicted using the current motion vector, plus the values predicted using the motion vectors of two adjacent blocks. This mode creates a smoother motion field, while producing a similar or smother predictive error frame than the default mode (no OBMC).

- Syntax-based arithmetic coding mode: Arithmetic coding [54, 55] is used instead of variable length coding, removing the restriction of using at least one bit per symbol, thus improving coding efficiency.

- PB-frames: Two frames are coded as a unit: a P frame, predicted from a previously decoded P frame; and a B frame, predicted from the current P frame and a previously decoded P frame. There are 12 blocks per macroblock in a PB frame. Only a part of the B frame is bidirectionally predicted. This mode allows for the doubling of the frame rate while producing only a minimum increase in the data rate, because the overhead for an extra B frame is eliminated [21]. However, the computational cost is increased, and real-time applications may be affected due to the coding of an additional frame.

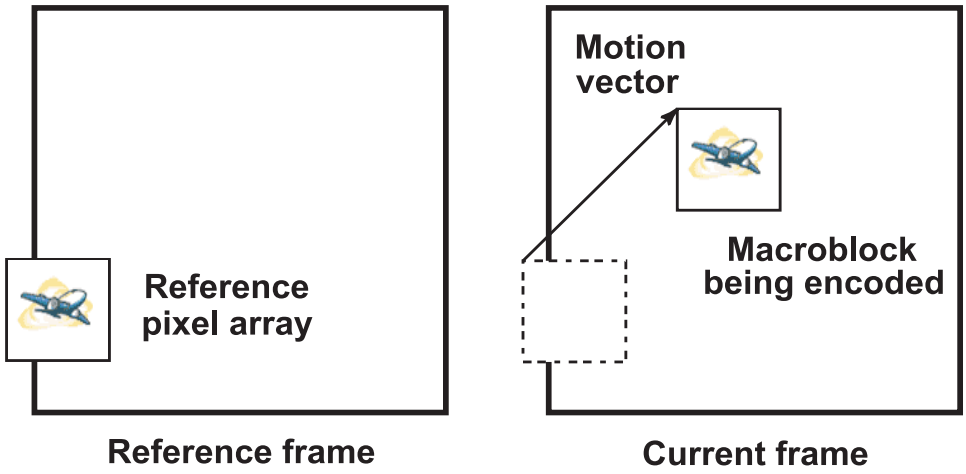


Fig. 2.16. In the unrestricted motion vectors mode, motion vectors can point outside of the reference frame.

In PB-frames, the forward and backward motion vectors are derived from the P frame motion vector and a difference motion vector, causing the performance to be poorer than that obtained by using B frames. The difference motion vector is obtained by subtracting the P frame motion vector from the motion vector obtained for the B frame. On the other hand, it provides lower overhead, particularly important for very low data rate encoding of sequences with limited motion, typical of videotelephony.

2.3.3 H.263+

H.263 version 2 [56, 57], known as H.263+, adds optional features to H.263 in order to broaden its range of applications, and to improve its compression performance. H.263+, which is backwards compatible with H.263, has functionalities such as:

- Custom picture formats: The source format (picture size) and the frame rate are negotiated during setup.
- Addition of scalability pictures:
  - Bidirectionally predictive-coded (B) pictures, as part of the temporal scalability mode. These pictures can be discarded without impacting the quality of future pictures, because they are not used to predict other pictures.

  - SNR scalable pictures: A difference frame, obtained by subtracting the original and the decoded frames, can also be encoded with finer quantization and sent to the decoder, producing an enhancement to the decoded picture. This effectively increases the subjective picture quality of the picture and, consequently, increases the signal-to-noise ratio of the video picture. Two new types of pictures are defined (see Figure 2.17):
    * Enhancement layer picture (EI): Prediction is only formed only from the temporally simultaneous reference picture in the base layer.
    * Enhancement P picture (EP): A predicted picture created using both a prior EI and a temporally simultaneous base layer reference picture.
  - Spatially scalable pictures: Similar to SNR scalable pictures. The picture in the base layer that will be used to predict the picture in the spatial enhancement layer is interpolated by a factor of two either horizontally or vertically (1-D spatial scalability), or both horizontally and vertically (2-D spatial scalability). This is shown in Figure 2.18. The decoder must be capable of handling custom picture formats, because interpolation step may produce pictures in non-standard format.

- Reduced-resolution update: Useful for complex scenes. It allows the encoder to maintain a high frame rate by encoding a low-resolution update to a higher-resolution picture, while maintaining high resolution in stationary areas.

- Improved PB frames: PB frames in H.263 are not sufficiently robust for continuous use, due to the limited types of prediction that the PB frame can use. B frames can be forward, backward, or bidirectionally predicted, making PB frames less susceptible to significant changes that may occur between pictures.

- Reference picture selection: Improves error resilience by allowing a temporally previous reference picture to be selected which is not the most recent encoded picture that can be syntactically referenced.

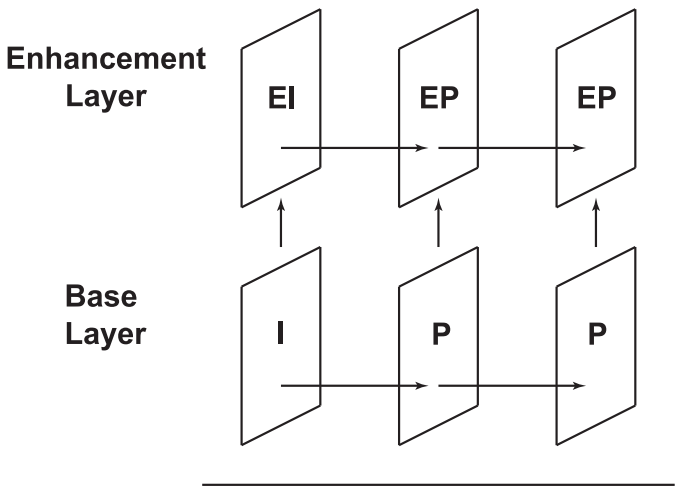


Fig. 2.17. Illustration of SNR scalability.

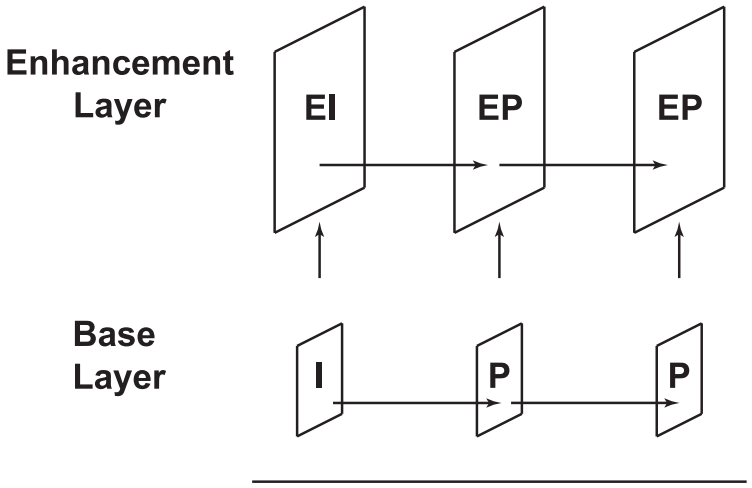


Fig. 2.18. Illustration of spatial scalability.

2.4 Scalable and Subband Image and Video Compression

Scalable and subband image and video compression have been areas of active research in recent years, mainly in connection with the efforts of JPEG2000 and MPEG-4. However, scalable and subband image and video schemes have been proposed since 1984 [58, 59, 60, 61, 62].

In this thesis, we use the terms "scalable" and "subband" image and video compression as follows: "Scalable" compression techniques have the capability to allow the change of the compression parameters, such as data rate or frame rate, at the time of decoding. "Subband" compression techniques first decompose the image or video sequence into a set of frequency channels or subbands, and the resulting subbands are then compressed.

Another term used in this thesis is "embedded" bit stream. An embedded bit stream can be truncated at any point during decoding, and be used to obtain a coarse version, in the SNR sense, of the image or video sequence. This version can then be refined by decoding additional data from the compressed bit stream. This process is terminated when a desired data rate or distortion is attained [3]. In contrast, in a "layered" bit stream, a base layer or stream is sent to the decoder. In addition, a set of refinement or enhancement layers is also transmitted. Each layer (base or enhancement) must be decoded entirely to produce a valid version of the image or video sequence. Most of the current video compression standards provide layered scalability. It is important to note that if a compression technique produces an embedded bit stream, it implies that the technique is scalable, because embedded bit streams can be truncated at any point during decoding. However, not all scalable compression techniques are embedded.

Motion compensation is used in video compression techniques to reduce the temporal redundancy in a video sequence. After motion compensation, the spatial redundancy in the predictive error frame is reduced using transform or subband coding. This approach is known as "out-band" motion compensation [63, 64]. Techniques using this approach are, for example, *SAMCoW*, MPEG-1/2, and H.26X. When mo-

tion compensation is used after subband decomposition, the approach is referred to as "in-band" motion compensation. This scheme is used, for example, in [65, 66].

Care must be taken when comparing the performance of different image compression techniques. In many cases, comparisons are made between a highly optimized approach and baseline JPEG, which is not optimized [50]. Methods have been developed to optimize JPEG while still maintaining compliance with the standard [67, 68].

2.4.1 Wavelet transform in image and video processing

There are many tools available for signal analysis. Common tools for studying one-dimensional signals are the Fourier, Laplace, and Z Transforms. For two-dimensional signals, the Haar transform, the Discrete Cosine Transform (DCT), and the Karhunen-Loève Transform (KLT) have been used to decompose the original image into a set of *basis images*, which correspond to *basis functions* in the 1-D case [37]. The DCT has also been used in image compression due to its energy compaction properties. More recently, the Discrete Wavelet Transform (DWT) [69, 70, 71, 72] has become a popular tool for signal processing, due to its superior space-frequency decomposition properties.

The DWT is a special case of a subband transform, with a filter bank being the basic building block [73]. The elements of a filter bank are a decomposition filter, a reconstruction filter, decimators and interpolators [74]. A two-channel filter bank, shown in Figure 2.19, typically separates the input signal into two bands: low-pass and high-pass. The decimators exist to maintain the number of samples in the original image.

In image and video processing, the DWT is obtained for the entire image or frame, and results in a set of independent, *spatially oriented* frequency channels or subbands [75, 76]. The wavelet transform is typically implemented using separable filters, although this is not a requirement. In the implementation of the DWT using separable filters, the rows of the image (the horizontal direction) are initially passed through the filters and downsampled. Then, the result of this operation is filtered and downsampled, but in the vertical direction. This results in four frequency subbands,

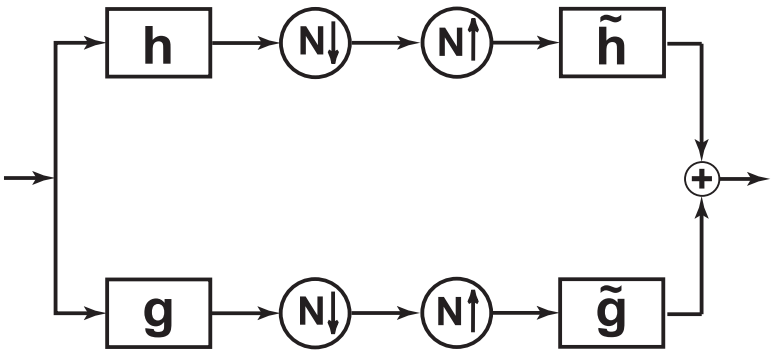


Fig. 2.19. Filter bank decomposition of a 1-D signal into two bands. h and g are the decomposition filters. $\tilde{h}$ and $\tilde{g}$ are the reconstruction filters.

namely LL, LH, HL, and HH, as shown in Figure 2.20. To obtain the next four frequency subbands, the lowest subband (LL), which represents the information at all coarser scales, is decomposed and subsampled. This procedure is repeated until the desired number of levels of decomposition is reached.

The analysis filters h and g effectively decompose the image into independent frequency spectra of different bandwidths or *resolutions* [71, 77, 78], producing different levels of detail. In Figure 2.21, a three-level wavelet decomposition from a grayscale image (512×512 pixels) is shown. The wavelet coefficients have been normalized on a band-by-band basis so that their range is in the interval $[0, 255]$. This decomposition structure is known as "dyadic" wavelet decomposition [77]. Other decomposition structures exist, such as wavelet packets [79, 80], which also have been investigated for image compression.

The reconstruction is accomplished using the inverse process, that is, upsampling the lower resolution images and passing them through synthesis filters $\hat{h}$ and $\hat{g}$, as shown in Figure 2.22.

In the implementation of the filtering stage of the wavelet transform, care must be taken with the boundaries of the images, as artifacts can be introduced if the symmetry of the subbands is not preserved. The boundaries of the image can be extended in several ways: periodic extension, whole-sample symmetric (WSS) extension,

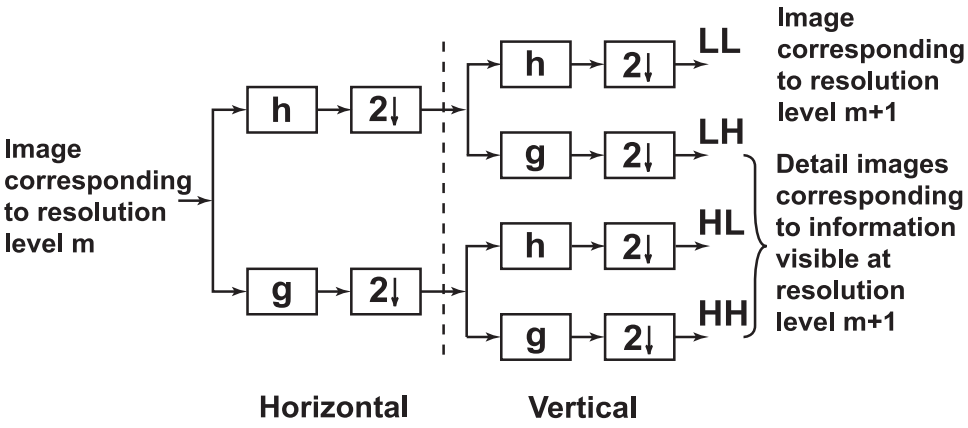


Fig. 2.20. One level of the wavelet transform decomposition.

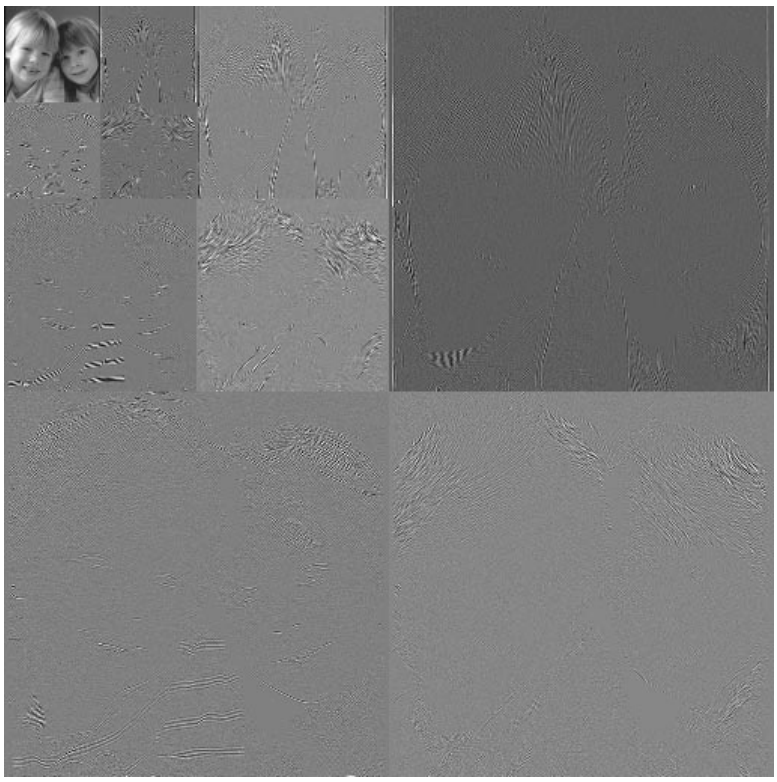


Fig. 2.21. Three-level wavelet decomposition of a grayscale image (512×512 pixels). The wavelet coefficients have been normalized on a band-by-band basis so that their range is in the interval [0,255].

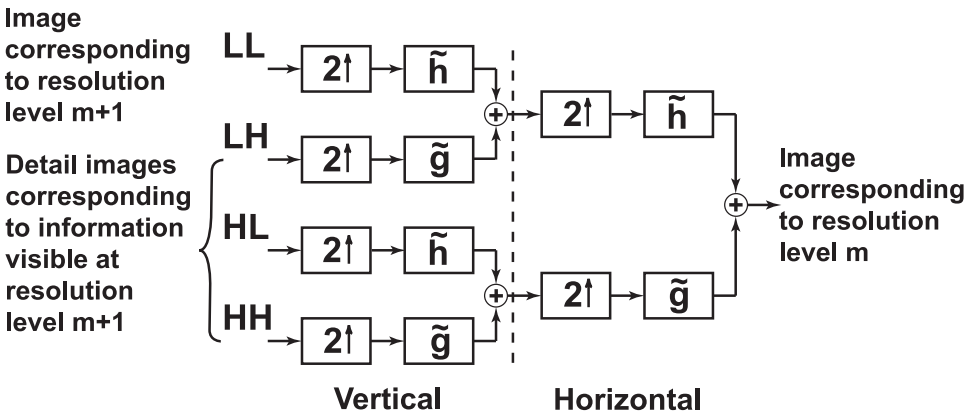


Fig. 2.22. One level of the wavelet transform reconstruction.

Table 2.2
Possible extensions at the boundaries of the image.

Input signal	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9	
Periodic	s7 s8 s9	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s0 s1 s2
Whole-sample symmetric	s3 s2 s1	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s8 s7 s6
Half-sample symmetric	s2 s1 s0	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s9 s8 s7
Repetition	s0 s0 s0	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s9 s9 s9
Zero extension	0 0 0	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 0 0 0

half-sample symmetric extension, repetition, or zero extension. This is illustrated in Table 2.2.

In [81, 82, 83, 84], it is shown that perfect reconstruction when using filter banks, such as the filter banks shown in Figures 2.20 and 2.22, can be obtained if the boundaries of the image are extended using WSS extension. Perfect reconstruction may also be obtained using periodic extension. However, when using the DWT as part of a image or video coding scheme, artifacts appear near the boundaries if periodic extension is used. This is due to the low correlation between the pixels in the image and the pixels used for extension. No artifacts near the boundaries are noticeable if WSS extension is used, as shown in Figure 2.23. Downsampling and upsampling are

Table 2.3
Downsampling and upsampling by a factor of two in a two-channel filter bank for periodic and whole-sample symmetric extension.

Periodic extension

Downsampling	Input signal	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
	Low pass	s0 s2 s4 s6 s8
	High pass	s0 s2 s4 s6 s8
Upsampling	Input signal	s0 s1 s2 s3 s4
	Low pass	s0 0 s1 0 s2 0 s3 0 s4
	High pass	s0 0 s1 0 s2 0 s3 0 s4

Whole-sample symmetric extension

Downsampling	Input signal	s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
	Low pass	s0 s2 s4 s6 s8
	High pass	s1 s3 s5 s7 s9
Upsampling	Input signal	s0 s1 s2 s3 s4
	Low pass	s0 0 s1 0 s2 0 s3 0 s4 0
	High pass	0 s0 0 s1 0 s2 0 s3 0 s4

done differently for WSS and periodic extension, as shown in Table 2.3 for the case of a two-channel filter bank like the one shown in Figure 2.19. In addition, for periodic extension to result in perfect reconstruction, the filter coefficients in one branch of the decomposition and reconstruction need to be shifted by one position. This is not necessary in WSS extension.

2.4.2 The EZW and SPIHT still image compression techniques

Several techniques have been proposed to achieve rate scalability in still image compression. Two of the most important techniques are Shapiro's Embedded Zerotree Wavelet (EZW) [85], and Said and Pearlman's Set Partitioning in Hierarchical Trees (SPIHT) [86]. Both make use of "spatial orientation trees" (SOT). Spatial orientation trees are structures that use quad-tree representations of sets of wavelet coefficients

(a) (b)

Fig. 2.23. Comparison between periodic and whole-sample symmetric extension when used within an image compression scheme. Artifacts near the bottom boundary of the image are noticeable when using (a) periodic instead of (b) whole-sample symmetric (WSS) extension.

that belong to different subbands, but have the same spatial location. This structures, which can be efficiently represented by one symbol, have been used extensively in rate scalable image and video compression. EZW and SPIHT are described next.

Embedded Zerotree Wavelet (EZW) algorithm

The Embedded Zerotree Wavelet (EZW) [85] algorithm exploits the interdependence between the coefficients of the wavelet decomposition of an image, by grouping them into SOTs. This is based on the following observations [3]:

- Most of the energy is concentrated at the low frequency subbands of the wavelet decomposition
- If the magnitude of a wavelet coefficient in the lowest subband of a decomposition is insignificant with respect to a threshold, then it is more likely that wavelet coefficients having the same spatial location in different subbands will also be insignificant
- The standard deviation of the wavelet coefficients decreases when proceeding from the lowest to the highest subbands of the wavelet pyramid

The combination of the above observations allows for the coding of a large number of insignificant wavelet coefficients by coding only the location of the root coefficient to which the entire set of coefficients is related. Such a set is commonly referred to as a "zerotree." The SOT used in EZW is shown in Figure 2.24. The root of the tree is the coefficient located in the lowest subband (LL) of the decomposition. Its descendants are all other coefficients in the tree.

The EZW algorithm consists of successive approximation of wavelet coefficients, and can be summarized as follows [85, 3]:

Let $f(m,n)$ be a grayscale image, and let $W[f(m,n)]$ be the coefficients of its wavelet decomposition.

1. Set the threshold $T = \max(W[f(m,n)])/2$

2. Dominant pass:

   - Compare the magnitude of each wavelet coefficient in a tree, starting with the root, to the threshold T
   - If the magnitudes of all the wavelet coefficients in the tree are smaller than the threshold T, then the entire tree structure (that is, the root and all its descendants) is represented by one symbol. This symbol is known as the zerotree (ZTR) symbol [85]
   - Otherwise, the root is said to be "significant" (when its magnitude is greater than T), or "insignificant" (when its magnitude is less than T). A significant coefficient is represented by one of two symbols, POS or NEG, depending on whether its value is positive or negative. The magnitude of a significant coefficients is set to zero to facilitate the formation of zerotree structures. An insignificant coefficient is represented by the symbol IZ, isolated zero.
   - This process is carried out such that all the coefficients in the tree are examined for possible subzerotree structures.

3. Subordinate pass: The significant wavelet coefficients in the image are refined by determining whether their magnitudes lie within the interval $[T, 3T/2)$, represented by the symbol LOW, or the interval $[3T/2, 2T)$, represented by the symbol HIGH.

4. Set $T = T/2$, and go to Step 2. Only the coefficients that have not yet been found to be significant are examined.

This coding strategy is iterated until the target data rate is achieved. The order in which the coefficients are examined is predefined and known by the decoder. The initial threshold is encoded in the header of the bit stream, followed by the resulting symbols from the dominant and subordinate passes, which are entropy coded using an arithmetic coder [54, 55]. An important element of EZW is the "significance map." The significance map is used to code the location of the significant coefficients in the wavelet decomposition. The simplest significance map is that where a significant coefficient is accompanied by the actual coordinates of its location. In EZW, the significance map is determined by having both the encoder and decoder know the scanning order of the coefficients. In essence, the dominant pass of EZW is the determination of the significance map, whereas the subordinate pass is the refinement of the wavelet coefficients that have been found significant. In consequence, the reduction of distortion during the dominant pass is smaller than during the subordinate pass. We will exploit this fact in our work.

EZW was developed for grayscale images. However it has been used for color images by using EZW on each of the color components separately. A drawback of this approach is that the interdependence between color components is not exploited. An improvement over this approach is described in Chapter 3. Statistical techniques for modeling wavelet coefficients [87, 88] have been investigated, as an alternative to the use of SOTs in image compression.

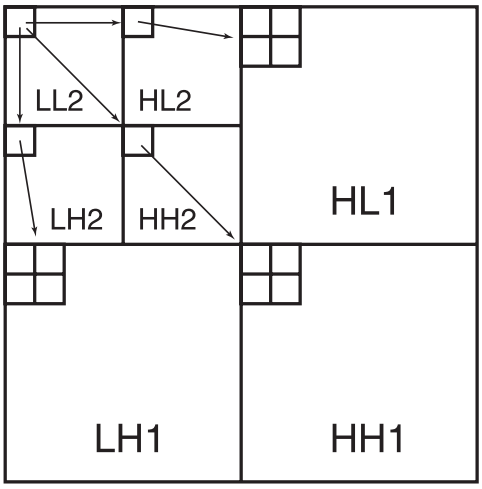


Fig. 2.24. Parent-descendant relationships in the spatial-orientation tree of Shapiro's EZW algorithm. The wavelet coefficient in the LL band has three children. Other coefficients, except for the ones in the highest frequency bands, have four children.

Set Partitioning in Hierarchical Trees (SPIHT)

Said and Pearlman [86] investigated different tree structures that improved the quality of the decomposition. Using Set Partitioning in Hierarchical Trees (SPIHT), the coefficients are divided into partitioning subsets using a known ordering in the SOT. Then, each subset is classified as significant or insignificant according to a prespecified rule based on the magnitude of the coefficients in the subset. In the SOT, descendants (a group of 2×2 adjacent coefficients) correspond to the pixels of the same spatial orientation in the next finer level of the decomposition, as shown in Figure 2.25. In contrast to Shapiro's zerotree, one coefficient in each group does not have descendants.

SPIHT groups the wavelet coefficients into three lists: the list of insignificant sets (LIS), the list of insignificant pixels (LIP), and the list of significant pixels (LSP). The SPIHT algorithm can be summarized as follows [86, 3]:

1. Initialize the LIS to the set of subtree descendants of the nodes in the highest level, the LIP to the nodes in the highest level, and the LSP to an empty list.

2. Sorting pass:

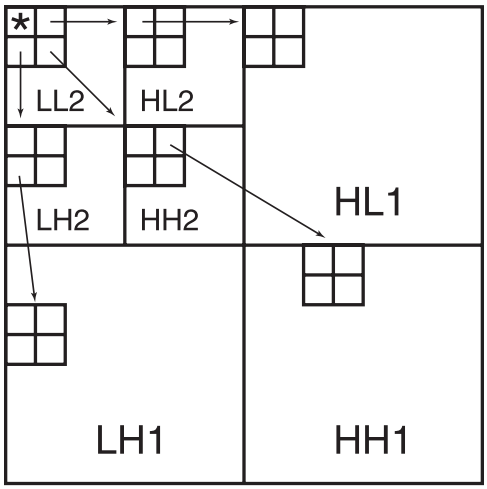


Fig. 2.25. Parent-descendant relationships in the spatial-orientation tree in Said and Pearlman's algorithm. One wavelet coefficient in the LL bands (noted with "*") does not have a child. Other coefficients, except for the ones in the highest frequency bands, have four children.

- Traverse the LIP testing the magnitude of its elements against the current threshold and representing its significance by 0 or 1. If a coefficient is found to be significant, its sign is coded and is moved to the LSP.
- Examine the LIS and check the magnitude of all the coefficients in that set. If a particular set is found to be significant, it is then partitioned into subsets and tested for significance. Otherwise, a single bit is appended to the bit stream to indicate an insignificant set.

3. Refinement pass: Examine the LSP, excluding coefficients added during the sorting pass. This pass is accomplished by using progressive bit plane coding of the ordered coefficients.

4. Set $T = T/2$, and go to Step 2.

The process is repeated until the target data rate is achieved. It is important to note that the locations of the coefficients being refined or classified is never explicitly transmitted. This is because all branching decisions made by the encoder as it searches

throughout the coefficients are appended to the bit stream. The output of the sorting-refinement process is then entropy coded.

The performance of SPIHT is better than EZW, at a modest increase in the computational complexity [86]. SPIHT was developed for grayscale images, but has been used for color images by first transforming the color components using the KL transform, and then using SPIHT on each of the transformed components separately.

2.4.3 Previous work in scalable and subband image compression

In [89], the image compression scheme adopted for JPEG2000 [90] is described. The scheme offers rate and spatial scalability with excellent compression performance. In the Embedded Block Coding with Optimized Truncation (EBCOT), each subband is partitioned into relatively small blocks of samples, referred to as "code-blocks," and generates a scalable and embedded bit stream for each code block. Code-blocks are of 32×32 or 64×64 samples each, and provide random access to the image. The layered bit stream may be truncated at any point during decoding. EBCOT uses the Daubechies (9,7) wavelet filter [75] (5-level decomposition), although JPEG2000 can use other filters. Results shown performance gains of about 0.5 dB against SPIHT [86] at various data rates.

In [91], the scheme adopted for MPEG-4 scalable texture coding, known as Multiscale Zerotree Wavelet Entropy (MTZE), is presented. This scheme is based on the Zerotree Wavelet Entropy (ZTE) [92] algorithm, extended to support multiple levels of spatial and quality scalability. ZTE is a wavelet-based image compression technique, similar to EZW, but with the following differences: i) explicit quantization (quantization is implicit in EZW); ii) bit-plane coefficient scanning (in EZW, scanning is done on the magnitude of the coefficient); iii) coefficient scanning can be depth-first (EZW always uses breadth-first); and iv) the alphabet of symbol is different to improve the performance at low data rates. In [91], the embedded bit stream is formed by several sections, each of which contains coefficients for a given spatial dimension with multiple quality levels. The first section of the bit stream is obtained by quantizing the coefficients using the coarsest quantizer. The quantized coefficients

and the significance map are then entropy-coded. The quantized coefficients are then reconstructed and subtracted from the original coefficients to obtain residuals, which in turn are quantized with the second-coarsest quantizer and entropy-coded to form the second section of the bitstream. This method can be used to obtain N levels of spatial and/or quality scalability. The results show that MTZE performs about 2.5 dB better than baseline JPEG for the luminance component, and about 6 dB better for the chrominance components.

In [93], a wavelet-based image compression scheme is presented. This iterative algorithm seeks to minimize a distortion measure (MSE) by successively pruning a tree for a given target rate. The survivor node in the tree are quantized and sent in the data stream along with significance map information. To further compress the stream, this map is predicted based on statistical information. Results show about a 0.5 dB gain in PSNR versus SPIHT for grayscale images.

2.4.4 Previous work in scalable and subband video compression

In [94], subband decomposition along the temporal axis is investigated. A group of W subsequent frames are analyzed. Pixels in a frame are classified as: covered, uncovered, and moving object. The classification is made two techniques: hierarchical block-matching motion estimation (BM), and interpolative motion compensation (IMC). Full- and half-pixel accuracy motion compensation is used.

A motion compensated frame is obtained by filtering the pixels in the frames in W, along the temporal axis, according to rules that depend on the category to which the pixels belong. A solution for the case $W = 2$ using a two-band subband decomposition is considered. Higher order filters (that is, $W > 2$) are also considered. It is shown empirically that with higher order filters ($W = 4, 8, 16$), better frequency separation is obtained. A drawback is the added delay when using longer filters. After decomposition, frequency bands are compressed using 2-D time-domain aliasing cancellation, followed by an adaptive lattice VQ.

The results show that both 3-D subband coding with IMC and 3-D subband coding with BM, outperform 2-D subband coding with traditional motion compensation by

about 2 dB. Target data rates were 1-10 Mbps, at 25 fps, for SIF and CCIR 601 spatial resolutions (see Table 2.1). This technique is not scalable.

In [65], a two-band temporal frequency decomposition is performed on the frame sequence, followed by a tree-structured spatial decomposition. The temporal filtering scheme is a two-tap Haar filter, basically the difference and average between frames. The spatial filtering scheme decomposes the two temporal subbands into an 11-band structure.

The spatio-temporal subbands are encoded using different schemes, according to the target data rate: for high data rates (1 Mbps), adaptive differential pulse code modulation (ADPCM) is used; for medium data rates (384 kbps), ADPCM plus geometric VQ (GVQ) is used; and for low data rates (128 kbps), unbalanced tree-structured VQ (UTSVQ) is used. The bit budget is allocated to each band according to its perceived significance in the reconstruction.

ADPCM refers to using one of five different DPCM schemes on a subset of the 11-band spatio-temporal decomposition, according to the spatial characteristics of the band. The rest of the spatial-temporal subbands are discarded because they are deemed to be perceptually insignificant.

GVQ is used to efficiently code high-frequency subbands that have perceptually significant data. It exploits the edge structure of these subbands. The code vectors reflect the basic shapes found in high-frequency subbands.

UTSVQ is used to encode the lowest frequency subband instead of ADPCM. It refers to the method used to create the VQ codebook, where the node that corresponds to the largest distortion reduction is split, creating an unbalanced tree.

The target data rates were 128, 384, and 1,000 kbps, at 30 fps, for CIF resolution frames. No PSNR results are reported, just a discussion of the visual quality of the decoded frames. This technique is not scalable.

In [95], an embedded, rate scalable video coding strategy is presented. The single bit stream can be decoded at a different frame sizes, frame rates, and is embedded and continuously rate scalable.

To support "multiresolution" (different frame sizes), a subset of the bit stream corresponding to the subbands associated with the desired frame size. Frame sizes supported are the original size ($m \times n$), and $m/a \times m/a$, where $a = 2^i$, $i = 1, 2, 3$, with frame rates of 30, 15, 7.5, and 3.75 fps, each color or monochrome.

The number of decomposition levels in the spatio-temporal subband structure used is four spatial levels and three temporal levels. Both spatial and temporal dimensions are two-channel decompositions. Due to temporal filtering, there exists latency between encoding and decoding.

Motion compensation is done using camera pan compensation. For camera and scene motion, decoding a sequence at a reduced frame rate introduces blurring. Thus, camera pan parameters are obtained by matching successive frames.

Three techniques are used for encoding the coefficients resulting from the subband decomposition: 1) Layered PCM, used for the high frequency subbands; 2) DPCM with layered conditional replenishment, used for the lowest frequency subband; and 3) Layered zero coding, used in all subbands to exploit the correlation among subband coefficients in inactive and smooth regions.

Results presented show better PSNR performance compared to EZW and SPIHT when used on still images, and better performance than MPEG-1 on sequences that adapt to the simple camera pan model used. Target data rates from 50 kbps to 2 Mbps are obtained with a single bit stream. This technique is rate and spatial scalable.

In [66], a 3-D wavelet based video compression algorithm is presented. A two-channel three-level decomposition is applied in both the temporal and spatial domains, on groups of 16 frames. A 3-D SPIHT algorithm is then applied to the decomposition.

This scheme is applied to grayscale SIF sequences (see Table 2.1). The results show that its performance is better (PSNR and visual quality) than MPEG-2. Data rates shown are 760 kbps and 2.53 Mbps. No results are shown for spatial or temporal scalability. This technique is rate, spatial, and temporal scalable.

In [96], an extension to the 3-D wavelet based video compression algorithm in [66] is presented. The extension includes support for color frames, groups of frames of different sizes (4, 8, and 16) to minimize delay, and unbalanced spatio-temporal orientation trees.

Embedded color coding is used, but it does not exploit the correlation between color components. That is, SPIHT is used separately on each color component, and an embedded bit stream is formed during encoding. Unbalanced spatio-temporal orientation trees are needed to support smaller groups of pictures (GOP). Coefficients in the same spatial position but different temporal position will have a different number of descendants.

Results show that this scheme performs better with larger GOP sizes. Also, using small GOP sizes performs lower during scene changes. The performance of this technique for small and large GOPs is comparable or bettern than that of MPEG-2 and H.263. Target data rates are 30, 60, 500, and 760 kbps. This technique is rate scalable.

In [97], a scheme based on "sprite" and subband coding is presented. Sprite coding is a video compression technique based on the object-based representation of the motion in a sequence, where the goal is to describe the motion of an object by a set of few parameters according to an object and a motion model. A long-term memory that contains all pixel information of the object that was visible over the sequence is known as background mosaic or sprite [98]. In [97], the compression scheme uses global motion compensation, that is, warps a frame using a single coordinate transformation, prior to to three-dimensional wavelet coding.

The method used to estimate global motion is least squares, by minimizing the squared difference between two frames. A matrix $(A_t)$is obtained to transform frame t into frame $t-1$. A_t is invertible, thus a frame F_i can be warped into another frame's F_j coordinate system by composition of matrices $A_i \cdots A_j$.

Groups of 16 to 32 frames are processed at a time, thus introducing delay unsuitable for applications such as video conferencing. The computational expense of this

scheme is an order of magnitude larger than H.263+. However, the scheme is fully rate scalable. Results show improvements over H.263+ (V3.2) of 0.2-0.5 dB. Target data rates are 50, 100, and 200 kbps. This technique is rate scalable.

In [99], a motion compensated scheme similar to that of [94] is presented. Two consecutive frames are filtered using the Haar wavelet (decomposition in the temporal dimension), and the prediction error signal is analyzed using the wavelet transform in the spatial dimension, using the Daubechies (9,7) wavelet filter. A rate-distortion analysis is used to manage the bit allocation between motion vectors and 3-D subbands. Comparisons against MPEG-1 at 1.2 Mbps show about a 1-2 dB performance gain in SNR. This technique is not scalable.

In [100], a technique for in-band motion compensation is proposed. This technique uses rate-distortion optimization to obtain a smooth motion vector field. The results show about a 1 dB improvement over H.263 at low (16 kbps) and high (18 Mbps) data rates. This technique is not scalable.

In [101], an in-band video compression scheme is presented. Motion vectors are hierarchically obtained for each subband, starting from the lowest subband. Pulse code modulation (DPCM) and differential PCM (DPCM) is used to encode the quantized data in the subbands. The number of reference frames and the intervals between them are adjusted according to the content of the video using a basic scene change detector. The scheme is compared against MPEG-1. This technique is not scalable.

In [102], a scheme based on "video significance-linked connected component analysis (VSLCCA) is presented. Motion estimation and overlapped block motion compensation (OBMC) is performed first. A wavelet transform decomposition of the predictive error frames is obtained. Then, the VSLCCA algorithm is used to code the resulting coefficients.

In the video SLCCA algorithm, the clustering property of the wavelet coefficients is exploited, by organizing them in "connected components" using morphological operations. In essence, the algorithm removes insignificant coefficients, favoring clusters of coefficients, and then using conditioning when entropy-coding the significance

map. Results show an improvement of about 0.5 dB when compared against H.263 and ZTE [92]. This technique is not scalable.

In [103], a hybrid coding scheme is presented. Motion estimation and compensation is used to reduce the temporal redundancy in the sequence. For the intraframe (I) and predictive error frames (PEF), a scheme known as partitioning, aggregation, and conditional coding (PACC) is used on the coefficients resulting from their discrete wavelet transform (DWT) decomposition.

PACC refers to a partitioning of the quantized wavelet coefficients into two groups: zero and non-zero coefficients. The zero coefficients are encoded using a zerotree structure as in Shapiro [85]. A zerotree symbol is used only for coefficients in the lowest subband of the decomposition. Non-zero coefficients are represented using two maps: a magnitude map, that contains the absolute value of the coefficients, and a sign map. All three maps are encoded using Conditional Coding, which exploits the spatial redundancy within the maps. Motion compensation is performed at half-pixel accuracy. The motion vectors and symbols generated by PACC are coded using a binary arithmetic coder.

A comparison versus the MPEG-4 verification model (VM) version 5.1 is presented (see Section 2.2). The results show PSNR improvements of about 0.15-1.2 dB at data rates from 10-123 kbps at 7.5-15 fps for QCIF size frames. The Daubechies (9,7) wavelet filter is used. The bit stream is not scalable.

In [92], a hybrid motion-compensated wavelet transform coder is presented. Motion compensation is done on a block-by-block basis, and the DWT and a zerotree approach is used to encode the I frames and PEFs. For the DWT longer filters are used at the start of the decomposition, and shorter filters are used at the later levels, to avoid ringing artifacts.

Two schemes are used to encode the wavelet coefficients: Shapiro's EZW [85] and zerotree entropy (ZTE) [92] coding. This technique only *provides a path* to spatial, object, and bit stream scalability. The bitstream produced by the coder *is not* rate scalable, nor embedded. Motion compensation is used with half-pixel

accuracy. Target data rates are 10-112 kbps, at 5-15 fps for QCIF size frames. A comparison versus the MPEG-4 VM version 1.0 is presented. This technique is not scalable.

In [104], a study on the performance of SNR scalability in H.263+ is presented. A difference is made in the H.263+ bit stream, between data fields and overhead fields. Data fields directly contribute to the reconstruction of the frame and, thus, improve the SNR. Overhead fields are all other fields, such as headers and quantization tables. It is shown that the enhancement layers of H.263+ can contain as much as 50% of overhead, compared to a 15% overhead in a single-layer bit stream. When two or more enhancement layers are used for a multicast session, the overhead aggregates. For example, a three-layer bit stream (10 kbps + 5 kpbs + 12 kbps) contains 30% of overhead. Thin enhancement layers (for example, 5 kbps) contain the most percentage of overhead.

It is recognized that one factor that contributes to the overhead in the coding of enhancement layers is the number of options available to the encoder for the enhancement layers, which are present on the macroblock header. A solution is to define in the picture header the options used to encode the enhancement pictures, to limit the number of options available to them, and to improve coding efficiency by optimizing the VLC tables used for the remaining options.

The results show an improvement of about 0.5 dB over H.263+ at 15 and 27 kbps. Target data rates are 10 (base layer), 15, and 27 kbps, at 5 fps, for SQCIF (166×120) frame sizes. The technique studied, namely H.263+, is SNR scalable.

A summary of the scalable and subband video coding techniques described in this Section is presented in Table 2.4.

New trends in video compression

Streaming video over packet networks is considered a joint source/channel coding problem, where the necessity of real-time delivery, bandwidth constraints, and error resilience determine the design parameters for the video transmission system [105, 106]. In packet networks, Shannon's Source / Channel Separation Theorem [107,

Table 2.4
Summary of scalable and subband video coding techniques.

Ref	First Author	Motion Compensation	Scalability Modes
[99]	S.-J. Choi	Yes	None
[66]	B.-J.Kim	No	Rate, spatial, temporal
[96]	B.-J.Kim	No	Rate
[100]	F. Kossentini	Yes, in-band	None
[101]	J. Lee	Yes, in-band	None
[103]	D. Marpe	Yes, OBMC	None
[92]	S. A. Martucci	Yes, OBMC	None
[94]	J.-R. Ohm	Yes, pixel-level	None
[65]	C. Podilchuk	No	None
[95]	D. Taubman	Yes, camera pan	Rate, spatial
[102]	J. Vass	Yes, OBMC	None
[97]	A. Wang	Yes, sprites	Rate
[104]	L. Yang	Yes	SNR

108, 109] no longer applies due to channel uncertainty and multiple access channels [110, 111]. The development of end-to-end video delivery techniques incorporates video compression techniques and networks protocols that use the knowledge of the state of the channel during transmission to provide good quality of service [112, 113]. Techniques such as multiple-description coding are being considered as alternatives to current approaches [114, 115, 116]. Mechanisms such as multicasting also play an important role in video communication.

Video compression techniques that use a zerotree structure for compression are not shift invariant [117]. That is, the coding of an image and a shifted version of it can produce totally different bit streams. Other wavelet-based image compression schemes that do not use the zerotree structure have been proposed [117, 118].

2.5 Special Purpose Processors

The convergence of digital video applications, such as video conferencing and video-on-demand, has increased the need for low-cost implementations of video processors and real-time performance. The increasing computing power available in desktop systems has made it possible to perform computationally intensive problems, such as video encoding and decoding, using general-purpose processors. However, given the volume and time constraints of digital video data, restrictions on frame size, frame rate, and allowable motion in the scene may need to be imposed.

General-purpose processors are designed to handle a wide range of applications, from word processing to floating-point arithmetic. In contrast, if a processor is designed for a specific class of applications, its functions can be improved to perform these applications in the most efficient manner [119].

Special purpose processors have been extensively used in the area of image and video processing [120, 121, 122, 123]. However, the amount of data associated with digital video can be a problem for any processing system. For example, a digital video sequence that conforms to the ITU-R 601 digital video recommendation (720×480 pixels per frame, 30 frames per second, and 16 bits per pixel), has an uncompressed data rate of 168 megabits per second (Mbps). A simple 3×3 filtering operation would

require more than 372 millions additions and multiplications per second. In order to address this problem, a digital signal processor (DSP) provides a restricted but powerful set of instructions and special optimizations, designed for the most common operations performed in signal processing [124]. Efficient memory block transfers and an integrated interface with external devices are typical of DSPs.

As new capabilities are being incorporated into general-purpose processors, the performance gap between these two classes of processors is becoming smaller. Processors with multimedia extensions support DSP-like instructions, accelerating the processing of digital video and audio data. One example of these type of processors is the Intel MMX family [125].

2.5.1 Digital signal processors for image and video processing

Real-time video processing applications need to accomplish a certain amount of work in a fixed period of time. If these requirements are not met, the quality of the output can be unacceptable. DSPs are deterministic architectures, where the time a task will take to complete can be determined in advance.

Digital signal processors are microcomputers specifically designed to manipulate digital signals [124, 126, 127]. They are designed to provide high performance in a specific class of operations typically used in signal processing algorithms: multiply-accumulate operations (MACs) [124]. DSP architectures make use of a variety of techniques to maximize performance:

- Extensive pipelining, described later in this section, divides the execution of an instruction into several shorter stages which can be overlapped, significantly increasing performance.

- A fast clock cycle, as a result of pipelining, increases the number of instruction cycles available between samples.

- A reduced, but optimized instruction set that implements the most important operations, such as addition and multiplication, in a very efficient manner. Many DSPs integrate features of RISC (Reduced Instruction Set Computer)

architectures to reduce their clock cycle. Specialized instructions, such as manipulation of arrays of data and loop control, are commonly found on DSPs too.

- The ability to issue multiple instruction per cycle, which coupled with efficient compilers and a number of multi-purpose functional units, can increase the performance significantly.

- Multiple buses that provide a large bandwidth for input and output of data.

These techniques allow the DSP to achieve real-time performance in a variety of applications. Systems based on digital signal processors have been used for image and video applications [120, 121, 122, 123, 128, 129, 130, 131]. In [120], a multiprocessor specifically designed for image processing and video applications is introduced. This device, the Texas Instruments *TMS320C80* (*'C80*), integrates five DSPs onto a single chip [132]. The master processor (MP) coordinates on-chip processing activities and manages communication with external devices. Four 32-bit parallel processors (PP) have advanced features, such as parallel multiple-pixel processing and bit field manipulation, to accelerate operations on image data. The MP and PPs are connected over a high-speed crossbar network. The applicability of the *'C80* to video compression and computer graphics is investigated in [128]. Real-time H.263 video compression based on the *'C80* is reported in [129].

In [121], an analysis of the computation requirements of MPEG-2 decoding is presented. A study is done at the instruction-level in order to identify the most computational intensive tasks of MPEG-2 decoding, with the purpose of optimizing them for a generic sequential processor model. A single-board real-time H.261 codec using application-specific integrated circuits (know as ASICs) is outlined in [122]. Single-chip implementations of a H.261 and MPEG-1 [123] and JPEG [130] (an still-image compression standard) encoders have been reported.

2.5.2 The Texas Instruments *TMS320C6000*

In this thesis, we investigate the implementation on a DSP of new techniques in error concealment and video coding. Our initial target processor is the Texas Instruments *TMS320C6000* (*'C6000*). The *'C6000*, was introduced by Texas Instruments in February 1997 [133, 134]. The *'C6000* is based on a very long instruction word (VLIW) architecture capable of issuing up to 8 instructions per clock cycle. Current silicon releases operate at up to 300 MHz for a fixed-point DSP (*'C6201*). A 167 MHz floating-point device (*'C6701*) is also available.

The *'C6000* contains a CPU, program RAM (which can be configured as cache), data RAM and internal buses. These communicate the CPU to peripherals such as external memory interface (EMIF), host port interface (HPIF), and direct memory access (DMA) [133]. DMA is a mechanism that allows data transfer between a source and processor memory without intervention from the processor. Internal addressing and data path buses, as well as DMA read/write buses, are 32 bits wide. A 256-bit bus connecting program memory and CPU, allows it to fetch eight 32-bit instructions every clock cycle. A simplified *'C6000* system block diagram is shown in Figure 2.26.

The CPU consists of two similar sets of four multi-purpose functional units, two register files, each containing sixteen 32-bit registers, plus a crosspath that allows the functional units to access operands from either file. The functional units execute logic, shifting, multiply, and data address operations. They include two multipliers (M units), six arithmetic logic units (L, S and D units), and dedicated units for data transfer between registers and memory (S and D units). All instructions can be executed conditionally with no penalty. This provides flexibility by avoiding the penalty associated with branching in pipelined architectures.

Eight instructions at a time, a *fetch packet*, are fetched from program memory as part of VLIW processing. Instructions to be executed in parallel (up to eight) form an *execution packet*. A major difference between the *'C6000* and traditional VLIW architectures is that, in the *'C6000*, multiple execution packets can be part of a fetch packet, therefore increasing the utilization of the CPU resources.

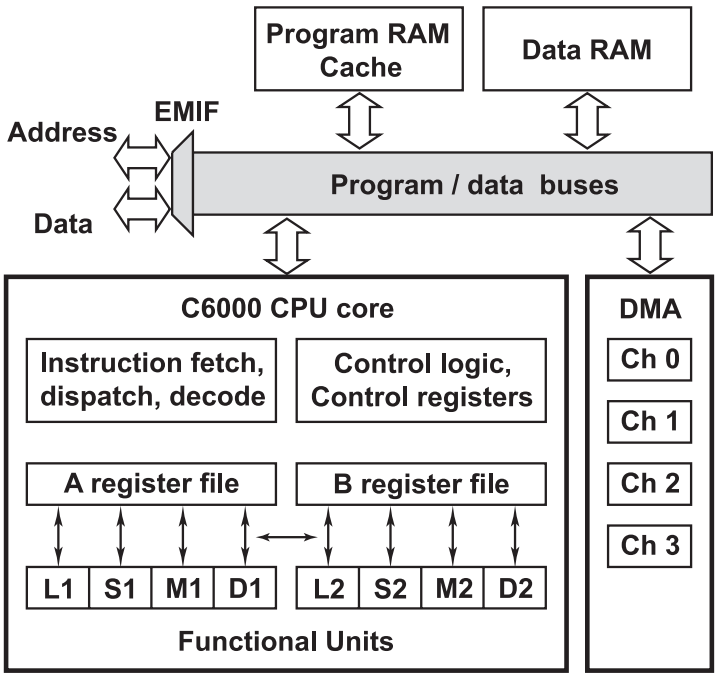


Fig. 2.26. *'C6000* system block diagram (simplified).

2.5.3 Architecture of a modern DSP

VLIW processors evolved from superscalar machines, where multiple independent instructions (typically 2 to 4) are issued every clock cycle [135, 136]. Superscalar architectures rely on hardware to determine if the instructions are independent and can be issued simultaneously. If instructions in a sequence are dependent, then only one instruction is issued at a time. In contrast, VLIW machines unload the responsibility onto the compiler, which has to create a package of instructions that can be issued simultaneously. The hardware does not play a role in the creation of this package. A potential problem of this architecture is that, if not enough independent instructions are found, the efficiency of the CPU would be low, since execution packets would not fully utilize the resources available. Therefore, the compiler must use several advanced techniques to schedule instructions from a sequence, such that they can be executed in parallel. Among other techniques, loop unrolling, software pipelining, and trace scheduling are commonly used by compilers for VLIW machines. The first

two are very effective when used in loops, which are an essential part of many signal processing applications. The third one is used in conjunction with conditional code.

Finding instruction level parallelism

In loop unrolling [135], the body of a loop is replicated several times in order to increase the number of instructions available for execution inside a loop. The compiler is then able to schedule instructions from different iterations to be executed in parallel. Since the loop body is replicated, the code used to determine the termination of a loop needs to be adjusted accordingly. The number of times the loop is executed may not be a multiple of the number of times the loop is unrolled, therefore it may be necessary to add extra code to handle this case. Loop unrolling is an effective technique to uncover instruction parallelism and decrease the start-up overhead of a loop. However, a major drawback is code size explosion as a result of code replication, which may be an issue for sensitive applications.

Software pipelining is another compile-time transformation that targets loops by reorganizing the body of the loop such that instructions from different iterations in the original loop are executed in the same iteration of the software pipelined code. As in loop unrolling, extra code is needed for starting up and ending the loop, however, no code replication is needed thus avoiding this major drawback. In contrast to loop unrolling, software pipelining is effective at attempting to achieve peak performance inside the loop body. Therefore, these techniques can be combined to produce highly efficient code.

A technique more complex to implement is trace scheduling. It is based on analyzing a piece of code to find instructions that can be executed as early as possible to form a fetch packet. This technique selects a sequence of instructions (known as trace) that would be executed with high probability (in particular, instructions in the body of loops), or instructions that if wrongfully executed would not alter the final result of the task. It then compacts the sequence by moving instructions as early in the trace as possible. Extra instructions may be necessary for clean up if the prediction is incorrect.

Pipelining

Pipelining is an architecture design for the execution of instructions in a CPU [135]. Each instruction is divided into simpler operations (known as pipe segments or stages), such as fetching, decoding, or executing. In a traditional processor, these operations are executed serially, causing the operation to take several cycles to complete. Once an instruction is fetched, the hardware associated with this operation remains idle until the next instruction is issued. In addition, some instructions are more complex than others: a floating point multiply of two 32-bit operands is likely to take longer to complete than a 16-bit integer addition. Resources would not be used effectively if all instructions were required to take the same number of cycles to complete.

In a pipelined architecture, instructions move through a sequence of stages. Once an instruction leaves a stage, the next instruction enters. Therefore, even if the number of cycles per instruction is not reduced, the execution of multiple instructions can be overlapped and hence decreasing their overall execution time. There are several implementation problems associated with pipelining:

- The CPU should have enough resources to have all stages operating at the same time, creating the need for hardware replication, thus increasing power consumption and chip size

- Instructions may need to be stalled until previous instructions are finished to maintain the correct flow of data in a program. This problem is known as *Data Hazard*. This problem does not exist in serial execution. Depending on the implementation, either the compiler or the hardware detects these conflicts and prevents them

- When branches are executed, the instructions behind in the pipeline may need to be stopped, depending on the branch being taken or not. This is known as *Control Hazard*. If the branch is taken, they need to be flushed from the pipeline, and instructions from the destination of the branch must be fetched, heavily affecting the performance of the pipeline.

The gain in performance when using a pipeline far outweights dealing with these problems. In the *'C6000*, there is enough resources to support issuing up to 8 different instructions per clock cycle, although a given functional unit can only handle certain subset of instructions.

It is responsibility of the *'C6000* user and the compiler to write code in such a way that the data hazards do not produce incorrect results. There is no run-time check for this kind of hazards. Similarly, for control hazards, the user and compiler must produce code that performs useful work while the CPU fetches instructions from the destination if a branch occurs. The pipeline, however, is not flushed.

2.6 Error Concealment in Compressed Digital Video Streams

In data networks (wired or wireless), cell or packet loss due to channel errors or congestion can cause data to be lost in the channel. When MPEG compressed video is transmitted, cell loss causes macroblocks and motion vectors to be removed from the compressed data stream [137]. Most video decoders will lose synchronization if the compressed video stream has missing data. In order to alleviate this problem, two issues need to be addressed: finding the location of the missing data and processing the sequence to recover the missing data. A frame from an MPEG sequence with missing macroblocks before and after error concealment is shown in Figure 2.27.

Fig. 2.27. Frame from an MPEG sequence with missing macroblocks before and after error concealment.

Recently, a great deal of work has been done in developing robust image and video coding techniques that include error concealment and resilience [138, 139, 140, 141,

142]. The goal of error concealment is to exploit redundant information in a sequence to recover missing data. Two approaches have been used: *active concealment* and *passive concealment* [143]. In active concealment, error control coding techniques and retransmission are used to recover errors [144], while in passive concealment the video stream is postprocessed to reconstruct the missing data. Passive concealment is necessary in many applications where active concealment cannot be used due to compliance with video transmission standards or when active concealment fails.

All video decoders that will be used in consumer applications, such as set-top decoder boxes, must implement some form of passive concealment. This problem is interesting in that it requires real-time implementation. Digital signal processors (DSPs), such as the Texas Instruments *TMS320C6201* (*'C6201*), are well suited for the demands of real-time processing, typical of error concealment. The *'C6201* is a fixed- point DSP based on the VelociTI architecture. The VelociTI architecture is a high-performance, very-long-instruction-word (VLIW) architecture developed by Texas Instruments [133].

Several techniques have been developed for the detection and reconstruction of missing data using passive error concealment [11, 145, 146, 10] in MPEG encoded streams transmitted over ATM networks. In order to reconstruct the missing data, the location of the missing macroblocks must be known. In [145], a new technique for packing ATM cells with compressed video data was introduced, with the aim of detecting the location of missing MPEG macroblocks in the encoded video stream. This technique also permits proper decoding of correctly received macroblocks, and thus preventing the loss of data from affecting the decoding process. Passive spatial and temporal techniques for the recovery of lost macroblocks in a compressed video stream have also been developed [146].

Error concealment algorithms

The popularity of digital video has increased the number of applications that transmit video bit streams over data networks, either terrestrial or wireless. The bandwidth requirements of these applications have found optical communication technologies ap-

propriate due to the large bandwidth they offer. To fully utilize optical networks, a protocol known as Asynchronous Transfer Mode (ATM) [147] has been chosen as the target protocol for broadband integrated services digital network (B-ISDN), due to its flexible transfer capability for a wide variety of data.

Data to be transmitted is buffered and packed into ATM cells that contain fixed length segments of user information, known as cell payload. Each cell also contains a header that carries routing and error detection information. Cells from different sources are then multiplexed onto the channel, thus increasing network efficiency. ATM also provides great flexibility to video applications, allowing a bursty bit stream in the source to be delivered without wasting capacity in the network during low-activity periods [148]. Cells are guaranteed to arrive in the same order as they were transmitted (cell sequence integrity); however, no provision is made for retransmission if a cell is lost due to channel error or congestion. Cells are marked as high or low priority. Low priority cells can be discarded as a way to exercise priority or congestion control in the network.

This is a significant problem when an MPEG encoded video stream is transmitted over ATM. A missing cell may contain information such as motion vectors, DCT coefficients, or a user-defined quantization scale. When the decoder tries to reconstruct a macroblock, it will use data belonging to other macroblocks. The goal of error concealment is to process the video sequence, by exploiting redundant information, to recover the missing data. For example, in I frames it is possible to have a lost macroblock surrounded by intact macroblocks that are used to interpolate the missing data. Error concealment algorithms are to be included in the decoder hardware, so they have to be simple enough to be implemented in real-time.

In [145], a new packing scheme was introduced. Important MPEG header information necessary for the proper decoding of the compressed sequences, such as timestamps and sequence number, are packed into high priority cells. All other data, such as motion vectors and DCT coefficients, are packed into low priority cells. Extra 9 bits are inserted at the start of each cell to provide the location of the first mac-

roblock being packed into the cell. These extra bits are also used to indicate when a macroblock spans across more than one cell. Another 7 bits are used to provide the relative address of the macroblock with respect to the preceding macroblock. The extra 16 bits are used to localize the loss of macroblocks within a frame while maintaining the ease of decoding the correctly received macroblocks.

In passive error concealment techniques, the video stream is postprocessed to reconstruct the missing data. Passive techniques can be [137]:

- Spatial: Missing information is reconstructed based on data from undamaged portions of the current frame. Spatial techniques can be:
  - Deterministic: Each pixel is reconstructed by spatial interpolation using a weighted sum of the nearest undamaged pixels. Other approaches involve iterative minimization of a cost function to reconstruct the lost macroblocks [149], or by estimation of edges using edge information obtained from nearest undamaged blocks [150].
  - Statistical: Each frame is modeled as a Markov Random Field. In [146] a maximum a posteriori (MAP) estimate of the missing macroblocks is obtained based on this model. The MAP estimate for each pixel within a lost macroblock is obtained by means of the iterative conditional modes (ICM) algorithm. The iterative method is guaranteed to converge to a global maximum, although the global maximum may not be unique. A fast suboptimal approach was presented in [10]. In this approach, it is shown that the median of the neighbors of a missing pixel is an approximate MAP estimate. This technique, is attractive for real-time implementation due to its simplicity. We will use this technique in the experimental work presented in Chapter 5.
- Temporal: A search is carried out for the missing information over a reference frame. Temporal error concealment techniques can be:

- Deterministic: Uses information from previous and future frames to reconstruct the missing information. One approach is to average the motion vectors of surrounding macroblocks. The average vector is used to retrieve a version of the lost macroblock [151]. This retrieved macroblock is then averaged with another version obtained via spatial interpolation.
- Statistical: The objective is to maximize the *a posteriori* distribution of the missing macroblock given its neighbors. This search can be accomplished using exhaustive search [146], or using a fast suboptimal temporal estimate that searches for motion vectors that yields the MAP estimate for the edge pixels only [10].

Images can be modeled using a stochastic approach. In this model, a pixel in an image is considered a random variable. An image, a matrix of pixels, would have random variable for entries. Thus, a particular image is a sample from the universe of all possible images, known as the ensemble of images [37]. This sample is known as a Random Field. Using a stochastic model of an image is attractive because analysis based on the image model is applicable to the entire class.

In [146], the image is modeled as a Markov Random Field (MRF). When a damaged frame is received in the decoder, the model is used to reconstruct missing information by estimating the original frame from the undamaged portions of the received frame using a maximum *a posteriori* (MAP) estimate.

The MAP estimate of the damaged frame is obtained in a pixel-by-pixel basis. In [146], the solution is obtained using the iterative conditional modes algorithm. However, the MAP estimates are not unique and the computational cost is very high, making it difficult to implement this technique in real-time. In [10], it is shown that a nearly optimal estimate is the median of the eight neighbors of a pixel. This solution lends itself for real-time implementation. This is described in Chapter 5.

3. *SAMCOW*: RATE SCALABLE VIDEO COMPRESSION

Rate scalable video codecs have received considerable attention due to the growing importance of video delivery over heterogeneous data networks. Current video coding standards such as MPEG-2 [24], MPEG-4 [25], and H.263+ [56] can produce bit streams at different data rates, but the sequence can be encoded only at a preset rate. They do provide, however, modes for layered temporal, spatial, and SNR scalability. In a layered approach, a number of connections corresponding to the number of layers being used needs to be established between the transmitter and the receiver.

Continuous rate scalable video compression techniques, that is, techniques that allow one to encode the sequence once and decode it at multiple data rates, can prove valuable due to the flexibility they provide to a video delivery system. They allow for the dynamical selection of the target data rate, and the rapid adaptation to the changing conditions of the network.

The Scalable Adaptive Motion Compensated Wavelet (*SAMCoW*) video compression technique developed by Shen and Delp [2, 4, 5], uses an embedded rate scalable coding strategy to obtain continuous rate scalability. It allows the data rate to be dynamically changed during decoding on a frame-by-frame basis so that it can be adjusted to meet network loading. Its performance is comparable to that of MPEG-1, MPEG-2, and H.263 at comparable data rates.

The main features of *SAMCoW* are: (i) a modified zerotree wavelet color image compression scheme known as *CEZW* [1, 2, 3, 4] used for coding intracoded (I) and predictive error frames (PEF); and (ii) adaptive block-based motion compensation (AMC) [152, 153] used in the spatial domain to reduce temporal redundancy of the video sequence. In this Chapter, we describe in detail the *CEZW*, AMC, and *SAMCoW* algorithms. We also identify the limitations of *SAMCoW* that impact its

compression performance. In Chapter 4, we develop several techniques [6, 7, 8, 9] to improve the performance of *SAMCoW*, particularly at low data rates.

3.1 *CEZW*: Embedded Coding of Color Images

The EZW [85] and SPIHT [86] algorithms were originally developed for grayscale images. For color images, the same coding scheme can be used on each color component. However, this approach fails to exploit the interdependence between color components. It has been noted that strong chrominance edges are accompanied by strong luminance edges [154, 155]. However, the reverse is not true, that is, many luminance transitions are not accompanied by transitions in the chrominance components. This spatial correlation, in the form of a unique spatial orientation tree (SOT) in the YUV color space, is used in a technique for still image compression known as Color Embedded Zerotree Wavelet (*CEZW*) [1, 2, 3, 4]. *CEZW* exploits the interdependence of the color components to achieve a higher degree of compression.

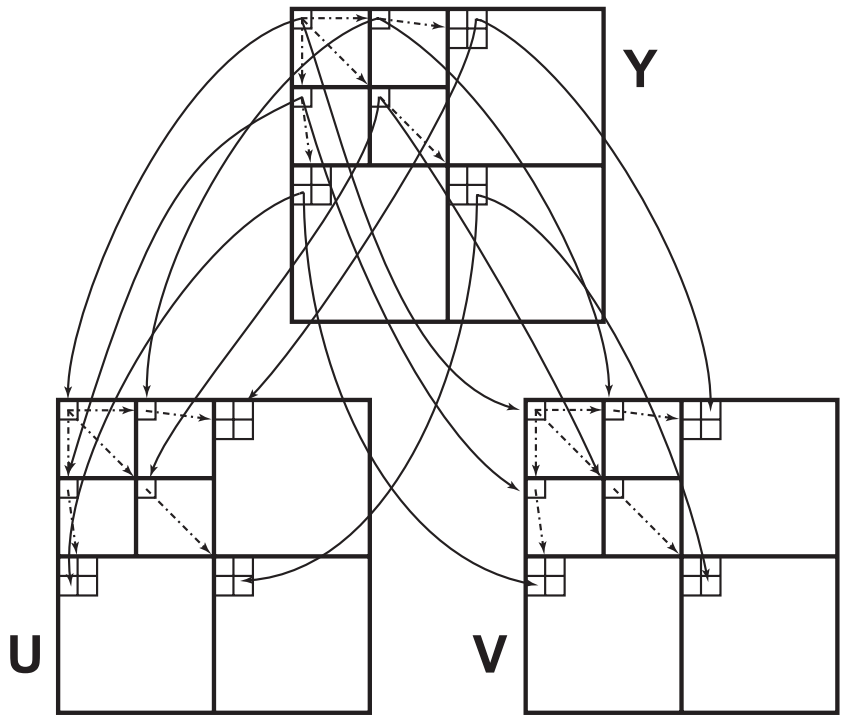


Fig. 3.1. Parent-descendant relationships in the *CEZW* algorithm.

In *CEZW*, the YUV space, consisting of a luminance component (Y) and two chrominance, or color difference, components (U, V), is used. A SOT structure as in Shapiro's is used for all three color components. In addition, each chrominance node is also a descendent node of the luminance node of the same location. Thus, each chrominance node has two parent nodes: one from the same chrominance component but in a lower frequency band, and the other from the luminance component. The SOT used in the *CEZW* algorithm is shown in Figure 3.1. The root of the tree is the coefficient located in the lowest subband (LL) of the decomposition of the luminance (Y) component. Its descendants are all other coefficients in the tree, including those of the luminance component, and those of the chrominance components.

The coding strategy in *CEZW* is similar to Shapiro's EZW [85], and can be summarized as follows:

Let $f_i(m,n)$ be a YUV image, where $i \in \{Y,U,V\}$ and let $W[f_i(m,n)]$ be the coefficients of the wavelet decomposition of component i.

1. Set the threshold $T = \max_{\forall i}(W[f_i(m,n)])/2$, $i \in \{Y,U,V\}$

2. Dominant pass:

   - The luminance component is scanned first. Compare the magnitude of each wavelet coefficient in a tree, starting with the root, to the threshold T

   - If the magnitudes of all the wavelet coefficients in the tree (including the coefficients in the luminance and chrominance components) are smaller than T, then the entire tree structure (that is, the root and all its descendants) is represented by one symbol, the zerotree (ZTR) symbol [85]

   - Otherwise, the root is said to be significant (when its magnitude is greater than T), or insignificant (when its magnitude is less than T). A significant coefficient is represented by one of two symbols, POS or NEG, depending on whether its value is positive or negative. The magnitude of a significant

coefficients is set to zero to facilitate the formation of zerotree structures. An insignificant coefficient is represented by the symbol IZ, isolated zero.

- The two chrominance components are scanned after the luminance component. Coefficients in the chrominance components that have already been encoded as part of a zerotree are not examined.
- This process is carried out such that all the coefficients in the tree are examined for possible subzerotree structures.

3. Subordinate pass (essentially the same as EZW): The significant wavelet coefficients in the image are refined by determining whether their magnitudes lie within the interval $[T, 3T/2)$, represented by the symbol LOW, or the interval $[3T/2, 2T)$, represented by the symbol HIGH.

4. Set $T = T/2$, and go to Step 2. Only the coefficients that have not yet been found to be significant are examined.

The compressed bit stream consists of the initial threshold T, followed by the resulting symbols from the dominant and subordinate passes, which are entropy coded using an arithmetic coder [54, 55].

3.2 Adaptive Motion Compensation (AMC)

In a scalable video codec, the decoded frames have different distortions at different data rates, making it impossible for the encoder to generate the exact reference frames as in the decoder for all the possible data rates [2, 4]. One solution is to have the encoder lock on to a fixed data rate (usually the highest data rate) and let the decoder run freely. The codec will work exactly as the non-scalable codec when decoding at the highest data rate. However, when decoding at a lower data rate, the quality of the decoded reference frames will deviate from that at the encoder. Hence, both the motion prediction and the decoding of the predicted error frames contribute to the increase in distortion of the decoded video sequence. This distortion also propagates from one frame to the next within a group of pictures (GOP). If the size of a GOP

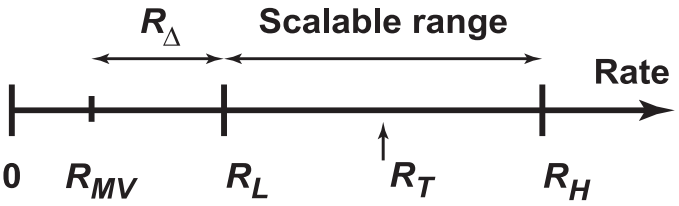


Fig. 3.2. In adaptive motion compensation, the scalable range is determined by the data rates high (R_H) and low (R_L). R_T is the target data rate, that is, the data rate at which the sequence is decoded. R_Δ is the fixed data rate at which the predicted error frame is encoded, to be used by both the encoder and decoder to generate the reference frame.

is large, the increase in distortion can be unacceptable. This distortion is sometimes referenced as *drift*.

SAMCoW uses a technique known as adaptive motion compensation (AMC) [152, 153], in which a feedback loop is added in the decoder, as shown in Figure 3.3. This is the only difference between Figure 3.3 and Figure 2.9. The data rates high (R_H) and low (R_L) determine the "scalable range," that is, the range of data rates at which the sequence can be decoded. The target data rate, R_T, is assumed to be within the range $R_L \leq R_T \leq R_H$, and R_{MV}, $R_{MV} \leq R_L$, is the data rate used to encode the motion vectors. This is shown in Figure 3.2.

At both the encoder and the decoder, the embedded bit stream is decoded at the fixed rate $R_\Delta = R_L - R_{MV}$, which is added to the predicted frame to generate the reference frame for the next frame. This allows both the encoder and decoder to use exactly the same reference frame, hence eliminating error propagation. Additionally, at the decoder, the embedded bit stream is decoded at the target data rate $R_T - R_{MV}$, which is added to the predicted frame to generate the final decoded frame.

3.3 Scalable Adaptive Motion Compensated Wavelet (*SAMCoW*) Video Compression Algorithm

The Scalable Adaptive Motion Compensated Wavelet (*SAMCoW*) [2, 4, 5] video compression algorithm is an embedded, rate scalable compression technique. *SAMCoW* uses block-based motion compensation to reduce temporal redundancy, and

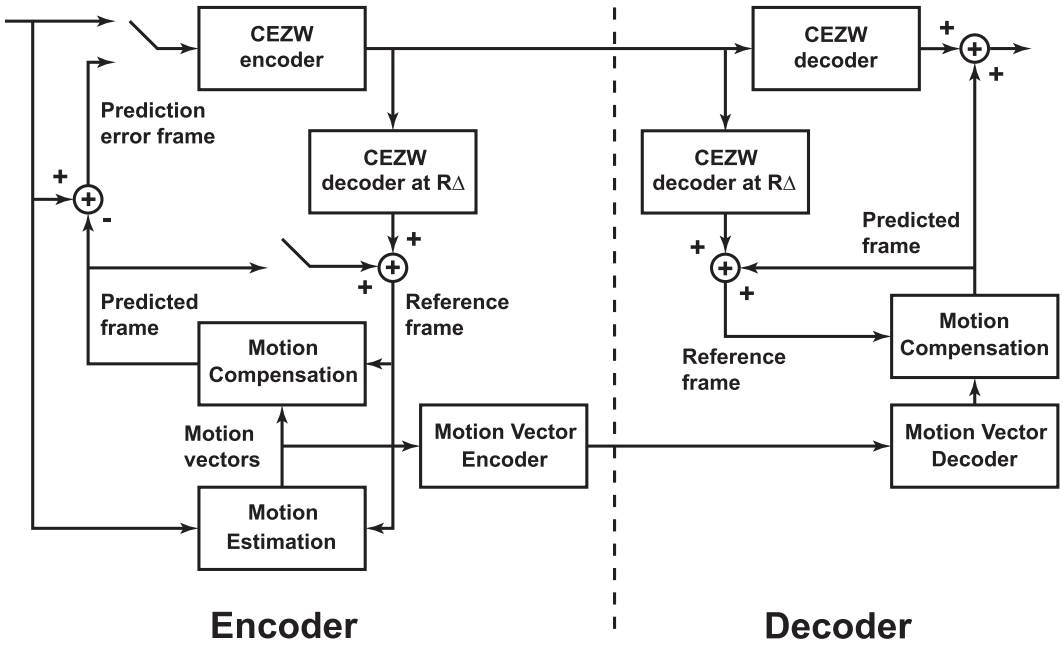


Fig. 3.3. Block diagram of the Scalable Adaptive Motion Compensated Wavelet (*SAMCoW*) video compression algorithm. It uses adaptive motion compensation (AMC), where a feedback loop is added to the decoder. This loop decodes the frame at rate R_Δ, allowing the decoder to use exactly the same reference frame as the encoder.

adaptive motion compensation (AMC) to eliminate drift at low data rates. It uses intracoded (I) and predictive-coded (P) frames only. I frames and predictive error frames (PEFs) are encoded using the Color Embedded Zerotree Wavelet (*CEZW*), which provides continuous rate scalability. *CEZW* operates on the entire frame and not just on blocks as in H.263 and MPEG. The block diagram of *SAMCoW* is shown in Figure 3.3.

SAMCoW uses integer-pixel accurate motion estimation and compensation, with the restriction that the motion vectors must point inside the reference frame. Overlapped block motion compensation (OBMC), presented in Section 2.3, is used to create a smoother PEF, where blocking artifacts caused by block-based motion estimation are reduced. Both the motion vectors and the *CEZW* symbol stream are encoded using an arithmetic coder [54, 55] to reduce their entropy.

In *SAMCoW*, a static bit allocation strategy is used to determine the bits assigned to frames in the sequence at the beginning of the encoding. This allocation remains in effect throughout the encoding process. The performance of *SAMCoW* at high data rates (1 Mbps and higher) is comparable to MPEG-1 and MPEG-2, while at medium data rates (between 64 kbps and 1 Mbps) is comparable to H.263. However, at low data rates (64 kbps and below), the performance of *SAMCoW* deteriorates [2, 4], particularly in sequences with complex scenes. In Chapter 4, we investigate in detail the reasons for this deterioration, and propose techniques to overcome the limitations of *SAMCoW*. These limitations include:

- The size of the frames used at low data rates (176×144 pixels (QCIF) and smaller) makes coding artifacts more noticeable

- The discrete wavelet transform is obtained for the entire frame causing a blurring effect

- Predictive error frames are not "natural" images. They have two main characteristics:

  - PEFs are noisy because many regions have pixel values near zero. That is, their value is small indicating that their contribution to the quality of the recontructed frame during motion compensation is minimal.
  - PEFs have block artifacts caused by block-based motion estimation despite the use of overlapped block motion compensation (OBMC) which greatly reduces their effect in the PEFs.

  When *CEZW* is used on PEFs, many bits are wasted on encoding these characteristics, and not enough bits are available to encode regions of the frame where the predictive error has a large magnitude, indicating a poor match in motion estimation.

- *SAMCoW* uses only basic video compression techniques, including restricting the motion vectors to point inside the reference frame, integer-pixel accurate motion estimation, I and P frames, and static bit allocation

4. *SAMCOW+*: LOW BIT RATE SCALABLE VIDEO COMPRESSION

In this Chapter, we investigate in detail the limitations of *SAMCoW*, and propose techniques designed to improve its quality at low data rates. Our work has focused on the use of advanced video coding techniques [6], and preprocessing and postprocessing techniques for predictive error frames [7, 8]. We also develop a approach based on rate-distortion theory to improve the performance of *CEZW* on intracoded (I) and predictive error frames (PEF) [9]. These techniques are integrated into an extension of *SAMCoW* referred to as *SAMCoW+*.

In *SAMCoW*, *CEZW* is used for the I frames and PEFs, as shown in the block diagram in Figure 4.1. The I frames and PEFs are obtained using basic video coding techniques, including:

- Restricting the motion vectors to point inside the reference frame
- Integer-pixel accurate block-based motion estimation and compensation
- The use of only intracoded (I) and predictive-coded (P) frames

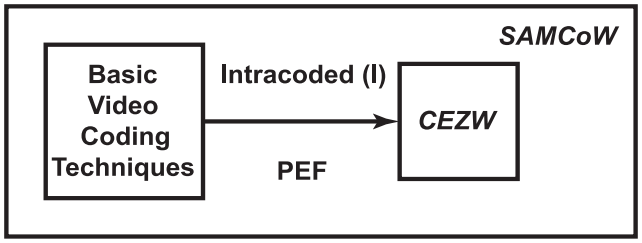


Fig. 4.1. Block diagram of *SAMCoW* showing the processing flow of intracoded (I) and predictive error frames (PEF).

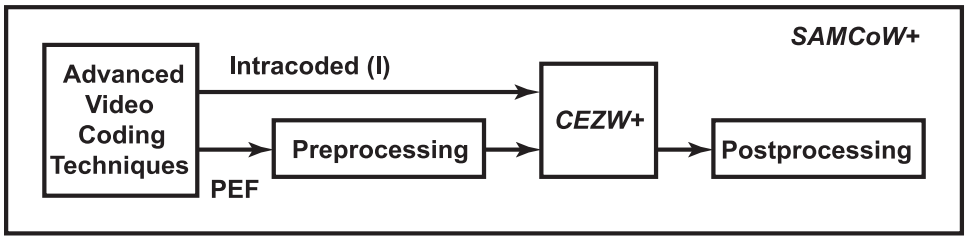


Fig. 4.2. Block diagram of *SAMCoW+* showing the processing flow of intracoded (I) and predictive error frames (PEF).

- The use of static bit allocation, determined at the beginning of the encoding process

In *SAMCoW+*, the I frames and PEFs are obtained using the advanced video coding techniques described in Section 4.3. Preprocessing and postprocessing techniques, described in Section 4.4, are also used on the PEFs to improve the coding performance. A version of *CEZW* that uses rate-distortion theory, as described in Section 4.5, replaces the original *CEZW* in *SAMCoW+*. We shall refer to this extension of *CEZW* as *CEZW+*. The block diagram of *SAMCoW+* is shown in Figure 4.2.

4.1 Performance Limitations of *SAMCoW* at Low Data Rates

In this Section, we investigate the issues that limit the performance of *SAMCoW* at low data rates, in order to obtain insight into the class of techniques that can be used to improve its performance.

4.1.1 The effect of image size

Maintaining acceptable quality in color images coded at low data rates (rates less than 0.5 bits per pixel (bpp)) is a challenge, especially in small images (176×144 pixels (QCIF) and smaller). Ringing artifacts and areas of discoloration are commonly noticeable when using wavelet-based image compression algorithms. The reason for this effect is that in the wavelet decomposition of larger images, a spatial orientation tree (SOT) spans larger smooth areas. In contrast, in smaller images, a SOT spans areas that contain more detail. When SOTs span smooth areas, more zerotree symbols are produced by *CEZW* than any other symbol. Since the symbols are entropy coded,

this results in a reduction in the data rate for a fixed distortion. However, when SOTs span areas that contain more detail, the distribution of the symbols generated by *CEZW* is more uniform, since more detail needs to be encoded. Therefore, at low data rates, the number of symbols encoded is not enough to sufficiently describe the detail in the image, due to the fact that the bit budget is exhausted before all the symbols can be encoded. Coding artifacts are therefore noticeable.

(a) (b)

(c) (d)

Fig. 4.3. Effect of image size on *CEZW*. (a) Original (512×512 pixels) (b) Encoded using *CEZW*, decoded at 0.25 bpp. (c) Cropped and resized (176×144 pixels) (d) Encoded using *CEZW*, decoded at 0.25 bpp.

CEZW, described in Section 3.1, is used in *SAMCoW* for intracoded (I) and predictive error frames. To show the effect of image size on the performance of *SAMCoW*, a 512×512 pixels, YUV 4:1:1 image is encoded using *CEZW*, and decoded at 0.25 bpp. The original and decoded images are shown in Figure 4.3 (a) and (b). The same image, cropped and scaled to 176×144 pixels (QCIF), is encoded using *CEZW*, and decoded at the same data rate (0.25 bpp). This is shown in Figure 4.3 (c) and (d). Artifacts are much more noticeable.

4.1.2 Blurring effect of the discrete wavelet transform

In a wavelet-based image compression algorithm, such as *CEZW*, the wavelet transform is obtained for the entire image, and the decomposition is performed repeatedly over the lowest subband (LL). The result is that the wavelet coefficients of the LL band at the coarsest scale have, in general, larger magnitude than the coefficients at the finer scales. This implies that when using *CEZW* at low data rates, a small number of coefficients at the finer scales will be found to be significant, thus their magnitudes are quantized to zero, effectively resulting in a blurring effect due to high frequency "wash out." In Figure 4.4, the statistics (mean and standard deviation) of the coefficients of the wavelet decomposition shown in Figure 2.21, are presented. Note that the mean ($\bar{x}$) and the standard deviation (σ) of the lowest (LL) subband, denoted as "A" in Figure 2.21, is the largest in the decomposition, indicating that more bits need to be assigned to them.

In algorithms that use the discrete cosine transform (DCT) on 8×8 blocks, such as JPEG, a similar phenomenon occurs at low data rates. The lower frequency coefficients, those who have the most energy, are encoded first, while higher frequency coefficients are quantized to zero. Therefore, blurring occurs at the block level, resulting in an overall effect of blockiness, with noticeable discontinuities at the borders of the blocks.

In Figure 4.5, a 512×512 pixels, YUV 4:1:1 image is encoded using *CEZW*, and decoded at 0.10 bpp (Figure 4.5 (b)). The same image is encoded using baseline JPEG at 0.10 bpp (Figure 4.5 (c)). The image coded using *CEZW* exhibits a low pass or

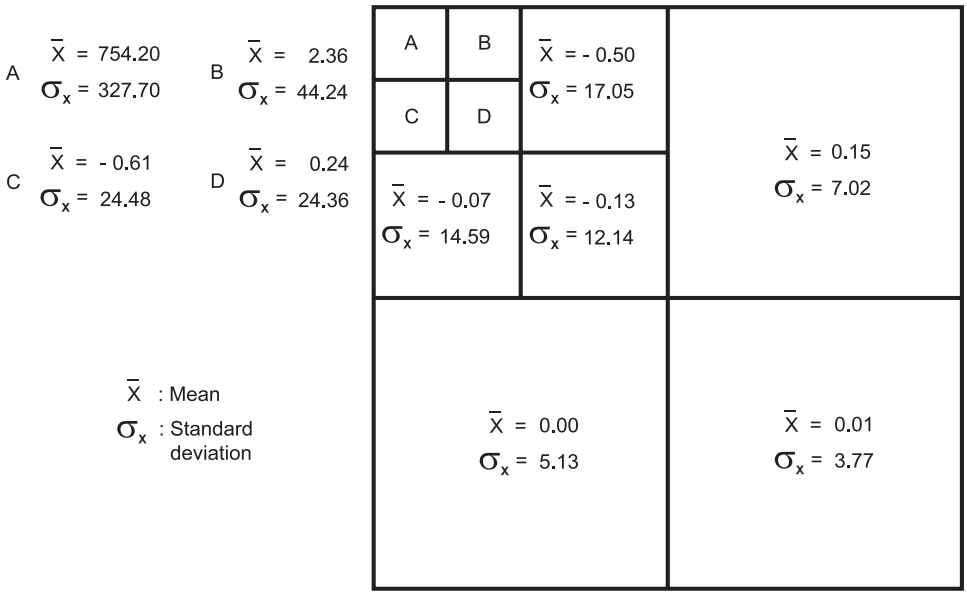


Fig. 4.4. Statistics of the coefficients of a three-level wavelet decomposition of a grayscale image (512×512 pixels). Shown are the mean ($\bar{x}$) and the standard deviation (σ). Note that the largest variations of the values of the coefficients are in the lowest subbands, as indicated by the values of σ.

blurriness effect, whereas the image coded using baseline JPEG exhibits blockiness. Blockiness is characteristic of block-based image and video compression algorithms.

4.1.3 The characteristics of predictive error frames in *SAMCoW*

Most of the research in wavelet-based image and video compression has been directed towards optimizing the coding performance on natural images [1, 2, 3, 4, 85, 86, 90, 93, 118]. Predictive error frames (PEFs), used in many video compression techniques, present a challenge for many codecs in that they are not "natural" images. Predictive error frames are obtained by subtracting the predictive frame (that is, the frame obtained using motion compensation) from the current frame. After motion compensation, the predictive frame is added to the predictive error frame to obtain the reconstructed frame at the encoder and decoder.

PEFs can be used to identify the areas of the frame where motion estimation and compensation were not able to obtain a good match between the current and reference frames. That is, we can identify the areas where the prediction error is

(a)

(b)

(c)

Fig. 4.5. Blurring effect of a wavelet-based image compression algorithm, compared against the blocking effect of an algorithm that uses the discrete cosine transform (DCT) on 8×8 blocks. (a) Original (512×512 pixels) (b) Encoded using *CEZW*, decoded at 0.10 bpp. (c) Encoded using baseline JPEG at 0.10 bpp.

large. Identifying these areas and using more bits to encode them, is crucial to the success of any video coding technique.

Two PEFs from the *foreman* sequence, extracted from *SAMCoW*, are shown in Figure 4.6. These are 176×144 pixels, YUV 4:1:1 images. The gray areas correspond to regions in the PEF where the pixel values are nearly zero, indicating small prediction error.

(a) (b)

Fig. 4.6. Predictive error frames from frames 35 and 293, respectively, of the *foreman* sequence. These are 176×144 pixels, YUV 4:1:1 images, and their pixel values have been shifted so that they are in the range $[0, 255]$.

PEFs have two characteristics that negatively affect the coding performance of *CEZW*. The first characteristic of PEFs is that they are "noisy" in the sense that many areas of the frame have pixel values that are nearly zero. This can be seen in Figure 4.6 (a), where these areas are the background. In this case, we would be more interested in encoding the facial detail because the error made by motion estimation is larger.

The contribution of the "noisy" areas to the quality of the reconstructed frame is minimal. However, due to the fact that the pixel values in these areas are not zero, the corresponding wavelet coefficients in the decomposition are also not zero. This creates a problem during encoding of the PEF, because bits are wasted on areas where the predictive error is small, and the wavelet transform is obtained of the

entire frame and not just on blocks as in H.263 and MPEG. Setting these pixels to zero would increase the performance of *CEZW*. We will describe a technique based on this approach in Section 4.4.

Wavelet coefficients whose magnitudes are small also limit the perfomance of *CEZW* because bits are wasted coding them. We are more interested in encoding coefficients with larger magnitude with higher precision, particularly in PEFs. Setting the value of these small wavelet coefficients to zero would also increase the performance of *CEZW*. We will also describe a technique based on this approach in Section 4.4.

The second characteristic of PEFs are the block artifacts caused by block-based motion estimation. During motion estimation, a good match between the current and reference frames, results in a block of pixels whose values are nearly zero. In constrast, a bad match results in larger pixel values. When a good match is followed by a bad match, it creates an artificial discontinuity between neighboring blocks, producing high frequency wavelet coefficients that will be encoded by *CEZW*, thus wasting bits used in encoding the artificial block boundary. This is largely addressed by the use of overlapped block motion compensation (OBMC), described in Section 2.3, in the original version of *SAMCoW* [2, 4, 5]. OBMC creates a smoother PEF by using a weighted sum of the motion vectors of the neighboring blocks to obtain the reconstructed frame. Some block boundaries still exist despite the use of OBMC, but we feel that they are not as important as the "noisy" regions in the PEFs. Thus, the mitigation of these block boundaries will not be addressed in this thesis.

Video coding standards such as H.263+ [56] make efficient use of block-based transform coding of the PEFs. Blocks may be skipped in regions of the PEF where the energy content is below a threshold. This is referred to as "skip" mode. Coding gains are obtained by assigning very short codewords to these skipped blocks. This results in allocating more bits to areas in the PEF with large predictive error. In *CEZW*, a special symbol, zerotree root (ZTR), is used to indicate that all the coefficients in a spatial orientation tree (SOT) are below a threshold. These are known as insignificant

coefficients (see Section 2.4). For PEFs, the use of the ZTR symbol is similar in concept to that of the SKIP mode in H.263+. However, in contrast to block-based techniques, where a block can be coded independently from others, in full-frame wavelet-based compression, the spatial orientation trees (SOT) span overlapping areas in the image. That is, a SOT cannot be used to independently code a particular region of the image, and the implementation in *CEZW* of a mode similar to the SKIP mode in H.263+ is difficult.

In Figure 4.7 (a) and (b), two PEFs from the *foreman* sequence are shown. These PEFs are encoded using *CEZW* and decoded at 0.25 bpp. The decoded frames are shown in Figure 4.7 (c) and (d). Blotchiness and ringing artifacts can be seen in the decoded frames, due to the fact that bits are wasted in encoding the nearly zero background regions. At the decoder, the reconstructed frame is obtained by adding the decoded PEF to the predictive frame produced by motion compensation. Therefore, these coding artifacts propagate to future frames when the decoded frame is used as a reference frame.

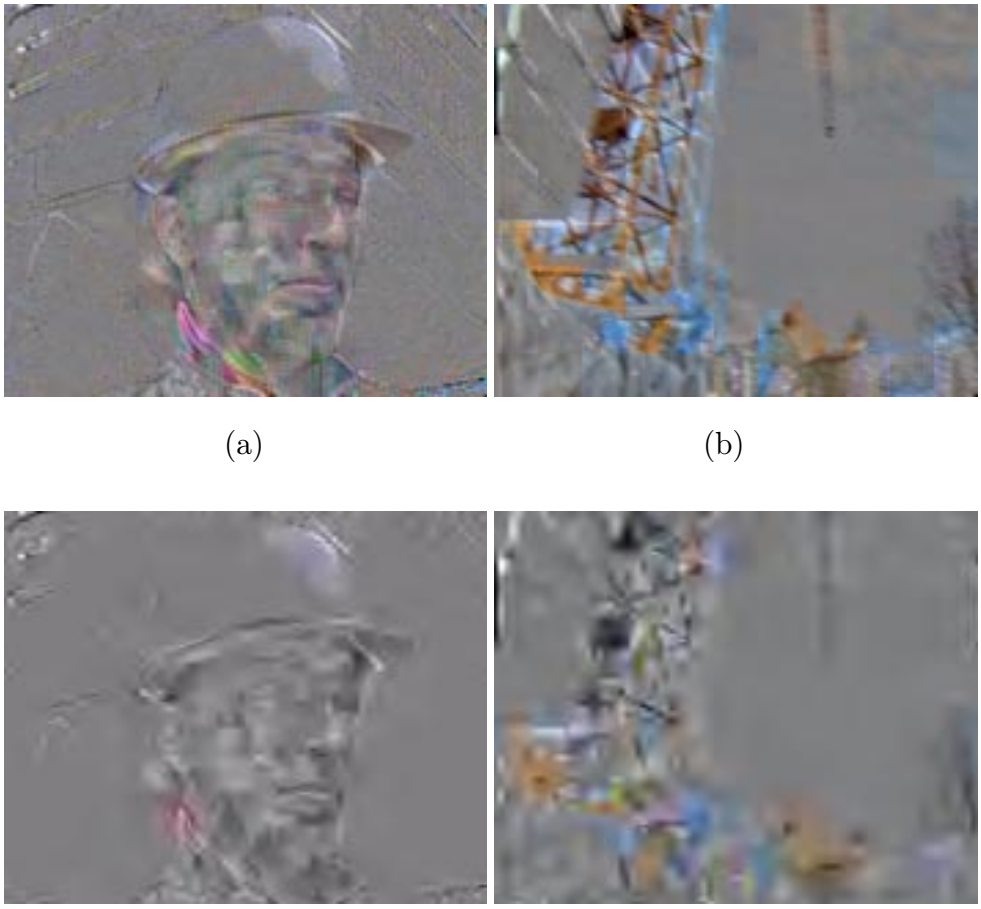
4.1.4 Use of basic video compression techniques in *SAMCoW*

SAMCoW uses only basic video compression techniques, including:

- Restricting the motion vectors to point inside the reference frame
- Integer-pixel accurate block-based motion estimation and compensation
- The use of only intracoded (I) and predictive-coded (P) frames
- The use of static bit allocation, determined at the beginning of the encoding process

In recent years, several advanced video coding techniques have been shown to provide better compression performance [21].

Restricting the motion vectors to point inside the reference frame works well for simple scenes [21]. However, for more complex scenes, those with medium and high motion where objects are entering and exiting the frame, the PEFs need to contain



(a) (b)

(c) (d)

Fig. 4.7. (a) and (b) Predictive error frames from frames 35 and 293, respectively, of the *foreman* sequence, (c) and (d) Encoded using *CEZW* and decoded at 0.25 bpp.

more detail, therefore making it more difficult to encode them at low data rates. Removing this restriction helps produce smoother PEFs, thefore improving compression efficiency.

While integer-pixel accurate motion estimation does reduce the temporal redundancy of a video sequence, it restricts the true frame-to-frame displacement of the objects in the sequence to coincide with the sampling grid of the frame [21]. If this restriction is removed by using sub-pixel-accurate motion estimation, improved prediction can be obtained [48]. Current video compression standards, such as H.263+ and MPEG-4 use half-pixel accurate motion vectors obtained by bilinear interpolation from the integer sampling grid, as described in Section 2.3, and as shown in Figure 2.14. Current efforts in other standards, such as H.26L [156], investigate the use of one-third-accurate motion vectors, as well as variable block-size motion estimation.

The use of I and P frames only causes problems when *SAMCoW* is used in video streaming applications over packet networks. This is because every frame in the stream is used as a reference for future frames. If a P frame is damaged or lost during transmission, the error cannot be recovered at the decoder until the next I frame is received. Bidirectionally predictive-coded (B) frames can provide to *SAMCoW* a form of error resilience, in that if they are damaged, the decoder can discard them without impacting other frames in the stream. In addition, when congestion occurs, B frames can be dropped from the stream, providing *SAMCoW* with a simple form of temporal scalability.

4.1.5 Use of static bit allocation in *SAMCoW*

In *SAMCoW*, a static bit allocation strategy is used to determine the bits assigned to frames in the sequence. A problem with this approach is that the scene content of the sequence varies continuously. For example, simple scenes could be followed by more complex scenes. Therefore, a static bit allocation scheme is not efficient considering that the quality of a motion compensated frame in a group of pictures (GOP) diverges from that of the original, since predictive-coded (P) frames are used

as reference for other P frames. This causes PEFs towards the end of a GOP to have larger prediction errors. In these cases, allocating more bits to frames with larger prediction error can improve the overall quality. In *SAMCoW+*, a new bit allocation scheme is used to address this problem.

4.2 Performance of *SAMCoW* Compared to H.263+

In Figures 4.9, 4.10, and 4.11, the performance of *SAMCoW* is compared against that of H.263+. We present results for the *coastguard*, *carphone* and *foreman* sequences at 24, 48, and 64 kbps, respectively, at 10 frames per second (fps). For *SAMCoW*, we used a GOP (group of pictures) size of 20, and four levels of wavelet decomposition. We used Release 0.2 of the H.263+ software, obtained from the University of British Columbia, Canada [57, 157]. Annexes D (Unrestricted Motion Vectors mode), E (Advanced Prediction mode), and F (Arithmetic Coding mode) were used, with TMN8 rate control and no frame skip. From these results, it can be seen that *SAMCoW* typically performs 3-4 dB lower than H.263+.

4.3 Advanced Coding Techniques

In this Section, we describe advanced video coding techniques that are used in *SAMCoW+* to improve its performance at low data rates. We will accomplish this by obtaining a more accurate prediction, therefore minimizing the prediction error in the PEFs. The result of this approach is that *CEZW* will be able to code the PEF more efficiently. These techniques are:

- Half-pixel accurate motion estimation and compensation
- Unrestricted motion vectors
- Bidirectionally predictive-coded (B) frames
- Dynamic bit allocation

4.3.1 Half-pixel accuracy

To obtain motion vectors with half-pixel accuracy, the best integer-pixel accurate motion vector is initially found using a search region of size 15×15 pixels. This motion

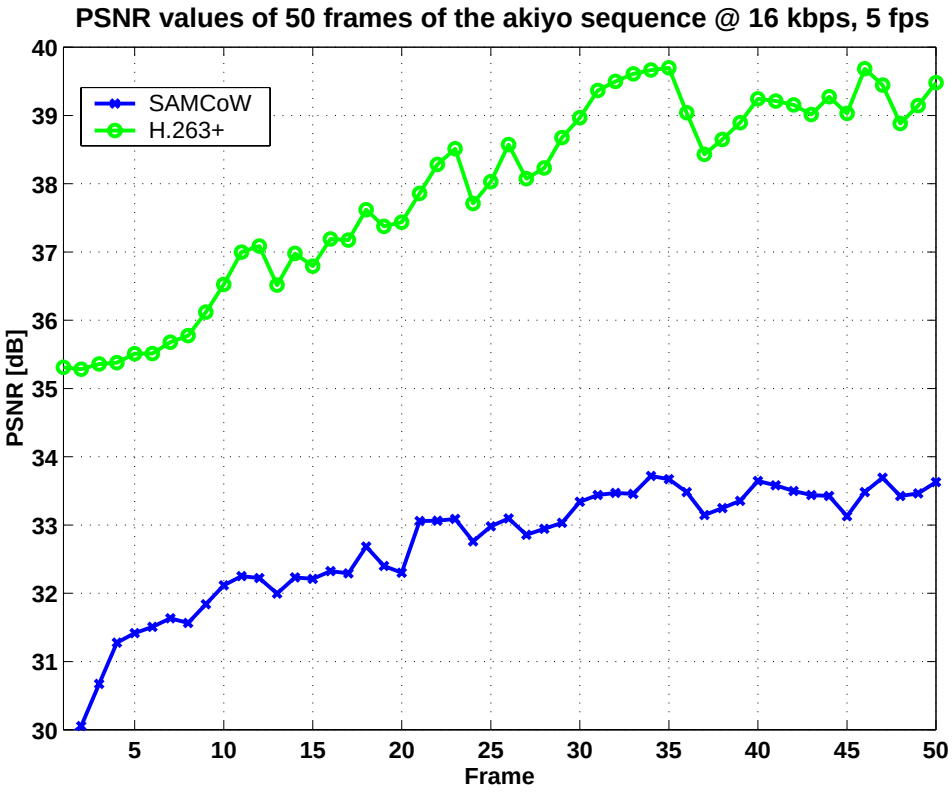


Fig. 4.8. Comparison between *SAMCoW* and H.263+: *akiyo* sequence at 16 kbps, 5 fps.

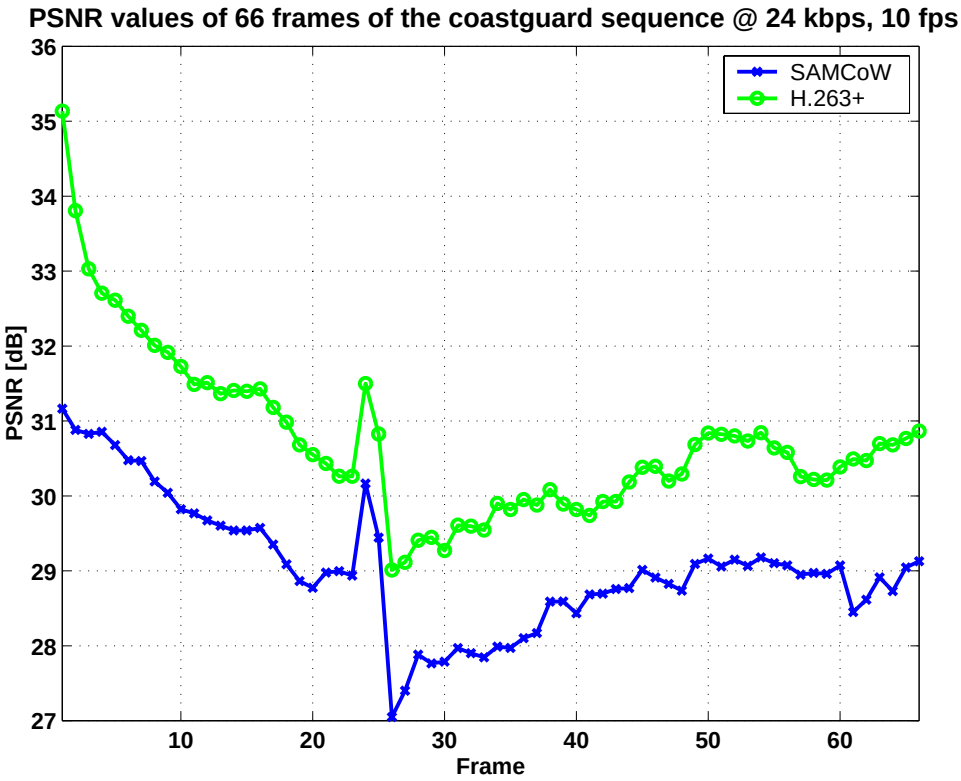


Fig. 4.9. Comparison between *SAMCoW* and H.263+: *coastguard* sequence at 24 kbps, 10 fps.

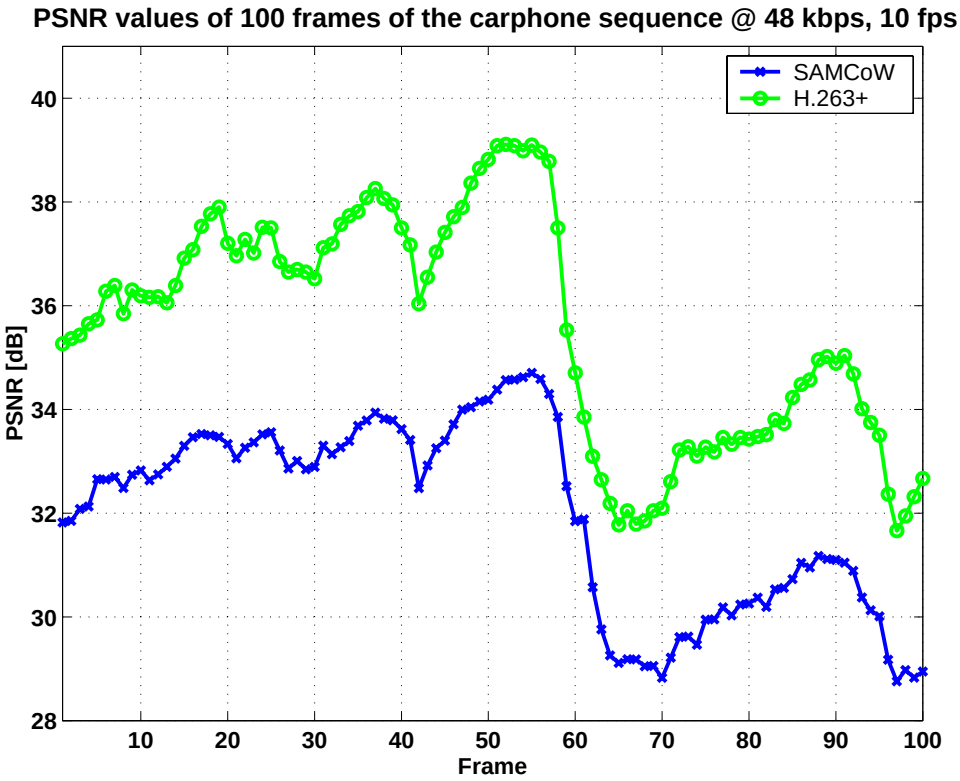


Fig. 4.10. Comparison between *SAMCoW* and H.263+: *carphone* sequence at 48 kbps, 10 fps.

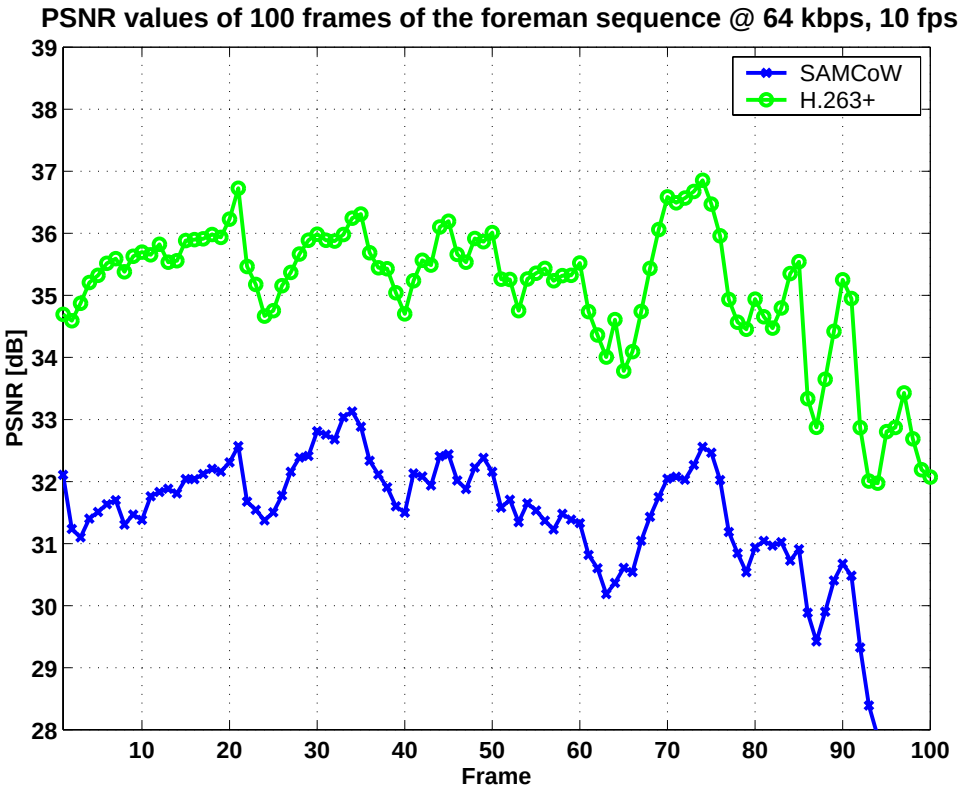


Fig. 4.11. Comparison between *SAMCoW* and H.263+: *foreman* sequence at 64 kbps, 10 fps.

vector is then used as the center of another search region, where eight half-pixel accurate positions are considered, as shown in Figure 2.14. The half-pixel representation is obtained using bilinear interpolation of the gray level values of adjacent pixels in the integer sampling grid. The error from the best integer-pixel accurate motion vector is compared to the errors from the half-pixel accurate motion vectors to obtain the best motion vector for the macroblock under consideration. Our matching metric is mean square error, although we can also use the sum of arithmetic differences. Both the luminance and the chrominance components are used to obtain the matching metric.

4.3.2 Unrestricted motion vectors

The motion vectors in *SAMCoW+* are not restricted to point within the boundaries of each reference image. When a pixel referenced by a motion vector lies outside the original reference image boundaries, the value used is that of the closest boundary pixel.

4.3.3 Bidirectionally predictive-coded (B) frames

In our implementation of B frames, we can choose to use a forward reference only, a backwards reference only, or both, depending on the content of the scene. The decision is based upon which motion vectors result in the best match for a particular macroblock. The motion vectors are encoded differentially, as shown in Figure 2.15. Note that in our implementation of B frames, we do not use intracoded blocks.

4.3.4 Dynamic bit allocation

In *SAMCoW*, bits are statically allocated. That is, at the beginning of the encoding, the target data rate for each frame is set, and is not changed throughout the encoding. In *SAMCoW+*, a dynamic bit allocation scheme is used to adjust for the varying content of the sequence. Based on the number of bits used for encoding motion vectors, and the statistics of the PEFs, we determine whether the number of bits assigned to a PEF should be increased or decreased, while maintaining the overall data rate within the target data rate. A similar process is used for intracoded (I) frames. We implemented a scheme where we maintain a running average of the number of bits used for encoding the motion vectors, plus the mean, minimum, max-

imum, and standard deviation of the pixel values in the frame. When more (less) bits are used for coding the motion vectors, we increase (decrease) the number of bits allocated to the PEF by a predetermined percentage. The same is true when the statistics of the frames change more than a predetermined threshold. Empirical evidence indicates that adjusting the bits in steps of 10% improves the performance. As the coefficients are tested for significance using the *CEZW* algorithm, they are assigned the appropriate symbols during the dominant and subordinate passes. The process terminates when the bit budget is exhausted. More elaborate schemes, such as the use of rate-distortion analysis can also be used.

4.4 Preprocessing and Postprocessing of Predictive Error Frames

In *SAMCoW*, *CEZW* is used for coding intracoded (I) frames and predictive error frames (PEF), as shown in Figure 4.1. In *SAMCoW+*, we modify the processing flow for I frames and PEFs as follows (see Figure 4.2):

- We use preprocessing and postprocessing techniques for PEFs to mitigate "noisy" regions. Preprocessing and postprocessing techniques have been used successfully in video to reduce the artifacts introduced by compression techniques [158, 159, 160, 161, 162].
  - Preprocessing: The preprocessing stage uses an spatial adaptive gain (AG) function to enhance areas of the PEF where the prediction error is large, and wavelet shrinkage [163, 164, 165] to reduce the number of wavelet coefficients with small magnitude before *CEZW* encoding.
  - Postprocessing: Wavelet-based video compression algorithms require that the transform be performed on the entire PEF and, hence, must contend with the low pass effect inherent to wavelet filtering. To counter this effect, we use an unsharp masking filter at the decoder to postprocess the decoded PEFs in *SAMCoW+*.
- We propose a modified version of *CEZW* for use with the preprocessing and postprocessing techniques. The goal of this modification is to identify the most

Fig. 4.12. Block diagram of the processing flow for preprocessing and postprocessing the PEFs.

important regions in the PEF, to allocate most of the bits. It should be noted that this approach will be replaced by the rate-distortion analysis described in Section 4.5.

A block diagram of the processing flow is shown in Figure 4.12.

4.4.1 Preprocessing stage

The goal of the preprocessing stage is to process the PEF, both in the spatial and wavelet domains, to improve the coding performance of *CEZW*. First, we would like to identify the areas of the PEF where the prediction error is large, and spend most of the bits encoding these regions. Second, we would also like to spend most of the bits on the wavelet coefficients that have large magnitude. Therefore, we do the following:

- Enhance the areas where the prediction error is large and set to zero areas where it is small. This enables *CEZW* to use more bits to encode the areas of large prediction error. We accomplish this by the use of an adaptive gain (AG) function.

- Setting to zero those wavelet coefficients whose magnitude is small enables *CEZW* to refine more the values of the larger wavelet coefficients, therefore spending more bits encoding these coefficients. We accomplish this by using wavelet shrinkage on the coefficients.

The block diagram of the preprocessing stage is shown in Figure 4.13.

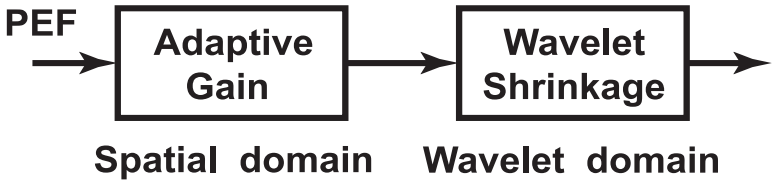


Fig. 4.13. Block diagram of the preprocessing stage.

Adaptive gain

In order to differentiate the areas of the PEF where the prediction error is large from those where the error is small, we partition the pixels in the PEF into four sets:

- The first set contains pixels whose values are the largest in the PEF. They represent areas of the PEF where the prediction error is the largest. Therefore, we would like to spend most of our bits on these areas, and thus we will not modify them.

- The second set contains pixels whose values are large but not as large as the pixels in the first set. However, the prediction error is still large, and we would like to differentiate these pixels from the rest of the pixels with smaller value. Therefore, we will increase their value using a linear function so that they are closer to the values of the pixels in the first set.

- The third set contains pixels whose values are small, but not negligible. We will not modify them because if there are enough bits allocated to the PEF, we may want to consider them in the encoding process.

- The fourth set contains pixels whose values we consider too small to make a significant contribution to the reconstruction. Therefore, we will set their values to zero.

To accomplish this differentiation between pixels in different areas of the PEF, we use an adaptive gain (AG) function that is similar to the GAG operator described

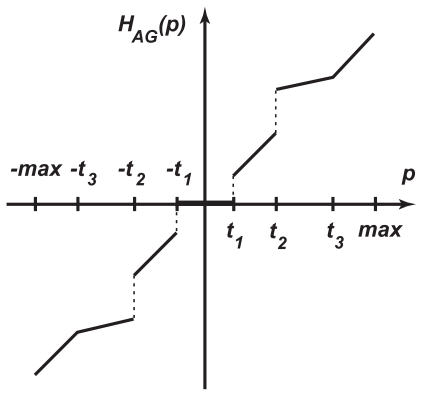


Fig. 4.14. Adaptive gain (AG) function used to enhance the features of a PEF. The horizontal axis represents the range of possible pixel values. *max* represents the largest pixel magnitude in the PEF.

in [166]. In this AG function, we define four regions that correspond to the four sets described above. The AG function is:

$$H_{AG}(p) = \begin{cases} 0 & \text{, if } 0 \leq |p| < t_1, \\ p & \text{, if } t_1 \leq |p| < t_2, \\ p + (t_3 - p)K & \text{, if } t_2 \leq |p| < t_3, \\ p & \text{, if } t_3 \leq |p| < max, \end{cases} \tag{4.1}$$

where p is a pixel value in the PEF, t_1, t_2, and t_3 are thresholds that depend on the content of the PEF, K is constant that controls how much gain each pixel in the second set receives, and max is the largest pixel value in the PEF. This function is shown in Figure 4.14. The horizontal axis of Figure 4.14 represents the range of possible pixel values. Note that the slope of the AG function for first and third regions is equal to one, that is, the pixel values are not modified.

The parameters of the AG function are set dynamically for each PEF in the sequence, in order to adapt to the varying content of the PEFs. The thresholds t_1, t_2, and t_3 are chosen based on the statistics of the PEF.

Based on empirical evidence, we have determined that the following threshold values yield good results: $K = 2.0$, $t_1 = 0.5\sigma$, $t_2 = 2.0\sigma$, and $t_3 = 3.0\sigma$, where σ is the standard deviation of the pixel values of the PEF. The choice of the above

values for the thresholds corresponds to selecting a different number of pixels in each region to modify in each PEF. The advantage of using σ is that it adapts better to the content of a video sequence. An alternative approach would be to select a fixed percentage of pixels for each region. However, we believe that adaptation is an important feature of our technique.

Wavelet shrinkage

Soft- and hard-thresholding of wavelet coefficients has been used for signal and image denoising [163, 164, 166, 167, 168]. Typical thresholding functions are shown in Figure 4.15. In [164], a uniform soft-threshold is used across scales of the decomposition, whereas in [166, 167, 168] soft-thresholding is scale-dependent, exploiting the fact that the statistics of the coefficients change at each scale.

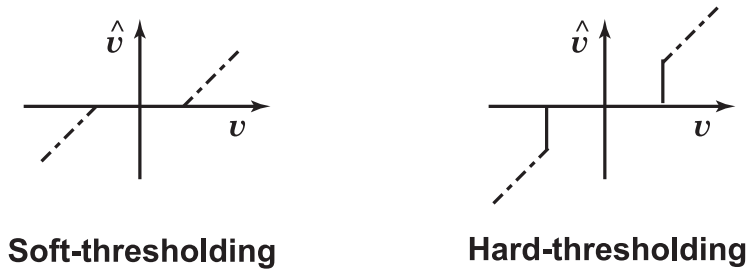


Fig. 4.15. Soft- and hard-thresholding of wavelet coefficient v

Our goal in encoding the wavelet coefficients in the PEF is to allocate bits to coefficients with the largest magnitude. By using less bits for the coefficients with the small magnitude, we are able to encode the larger coefficients with better precision, thus improving the reconstruction. To accomplish this task, we set the smaller coefficients to zero, using a soft-thresholding technique [164] with a scale adaptive threshold [166], as described below.

Let $f(m,n)$ be a predictive error frame (PEF), and $v = W_j^d[f(m,n)]$ be a wavelet coefficient in the decomposition of $f(m,n)$ at level j ($1 \leq j \leq J$) and spatial orientation d ($d \in \{HH, HL, LH, LL\}$). The new wavelet coefficient $\hat{v}$ is obtained by:

$$\hat{v} = sign(v)(|v| - t_j^d)_+ \tag{4.2}$$

where

$$sign(v) = \begin{cases} +1, & \text{if } v > 0, \\ 0, & \text{if } v = 0, \\ -1, & \text{if } v < 0, \end{cases} \tag{4.3}$$

$$(|v| - t_j^d)_+ = \begin{cases} |v| - t_j^d, & \text{if } |v| > t_j^d, \\ 0, & \text{otherwise,} \end{cases} \tag{4.4}$$

and t_j^d is a threshold. The value of t_j^d depends on the statistics of the wavelet decomposition at level j and orientation d, and is obtained as follows:

$$t_j^d = \begin{cases} (T_{max} - \alpha(j-1))\sigma_j^d, & \text{if } T_{max} - \alpha(j-1) > T_{min} \\ T_{min}\sigma_j^d, & \text{otherwise} \end{cases} \tag{4.5}$$

Here, α is a constant which determines the change in magnitude of the threshold between two consecutive levels, and T_{max} and T_{min} are maximum and minimum values of σ_j^d, the empirical standard deviation of the wavelet decomposition at the corresponding level and orientation, respectively. For our experiments, we select $\alpha = 2.0$, and $T_{max} = \sigma_J^{LL}$, and $T_{min} = \sigma_1^{HH}$ since, in general, these are the largest and smallest values of σ_j^d for a given frame.

4.4.2 Modified *CEZW*

After preprocessing, the coefficients of the wavelet decomposition of the PEFs are then encoded. In *CEZW*, several passes are made to refine the precision of the approximation. As the coefficients are examined, the symbols positive significant (POS), negative significant (NEG), isolated zero (IZ), and zerotree (ZTR) are assigned [1, 3]. A coefficient is assigned the symbol IZ when the coefficient is not significant but some of its descendants are significant with respect to a threshold. We would like to select the SOTs that include the coefficients in the decomposition with the largest magnitude. Thus, we would allocate bits for the PEFs as efficiently as possible, by encoding the largest coefficients and disregarding smaller coefficients whose contribution to the reconstruction is small. This strategy effectively skips SOTs in the wavelet decomposition. Therefore, we modify the *CEZW* algorithm as follows:

1. In the first dominant pass, we identify the coefficients that are significant (positive and negative) at the coarsest scale. We refer to these coefficients as "significant tree roots," and their descendants are part of a "significant tree." The result of this stage is that only a select number of trees are considered for further processing.

2. The *CEZW* proceeds as normal, except that in the remaining dominant passes, until the data rate is exhausted, only coefficients that belong to the "significant trees" are examined.

It is important to note that this modified *CEZW* algorithm will be replaced by the rate-distortion analysis described in Section 4.5.

4.4.3 Postprocessing stage

Due to the full-frame wavelet transform used in *CEZW*, blurriness is present at low data rates. To counter this effect, we use an unsharp masking filter at the decoder on the decoded PEFs to accentuate the high pass components of the frames, as shown in Figure 4.16.

Fig. 4.16. Unsharp masking filter.

The unsharp masking filter operation can be represented by [37]:

$$g(u,v) = f(u,v) + \lambda d_{lapl}(u,v) \tag{4.6}$$

where λ is a constant, and $d_{lapl}(u,v)$ is the discrete Laplacian defined as:

$$d_{lapl}(u,v) = f(u,v) - [f(u+1,v) + f(u-1,v) + f(u,v+1) + f(u,v-1)]/4 \tag{4.7}$$

Unsharp masking has been used to enhance edge information in images. In our experiments, we have found that the value $\lambda = 2.0$ is a good trade-off between under- and over-enhancing the edges of the image

4.5 *CEZW+*: A Rate-Distortion Approach for Scalable Image and Video Compression

In this Section, we develop a framework based on rate-distortion theory for efficiently encoding predictive error frames (PEF). This extension of *CEZW* is known as *CEZW+*. A block diagram is shown in Figure 4.17. This framework improves the modified *CEZW* algorithm presented in Section 4.4.2. Based on rate-distortion analysis, we allocate bits to the coefficients in a SOT in such a way that the largest reduction in distortion is obtained for a given data rate. We refer to this as the "best operating point" of the SOT. We maintain the embedded and rate scalable properties of our video compression algorithm. A block diagram of *CEZW+* is shown in Figure 4.18. Rate-distortion optimization has been used extensively for image and video compression [89, 93, 100, 169, 170, 171, 172, 173, 174, 175, 176].

Fig. 4.17. Block diagram of the processing flow used for PEF in *SAMCoW+*. We use *CEZW+*, an extension of *CEZW* that uses rate-distortion analysis to determine how to best allocate the bits assigned to a PEF.

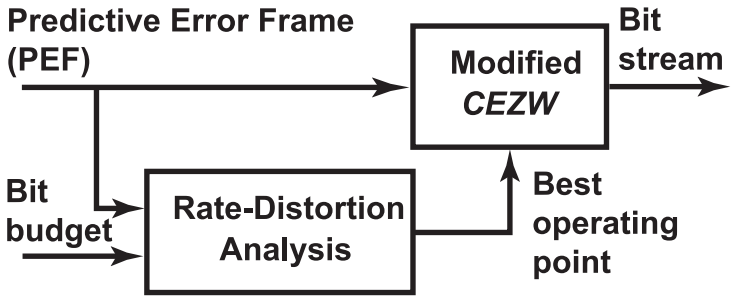


Fig. 4.18. Block diagram of our proposed approach (*CEZW+*) for encoding predictive error frames (PEFs) in *SAMCoW+*.

Although in this Section we describe the use of *CEZW+* on PEFs, our work is not restricted to PEFs. As shown in Figure 4.2, *CEZW+* is also used for intracoded

(I) frames. It is important to note that when *CEZW+* is used on I frames, neither preprocessing nor postprocessing are used these frames. That is, preprocessing and postprocessing are used on PEFs only. In [176], a rate-distortion optimized embedded coder for grayscale images is presented. The coefficients in a wavelet coefficient decomposition are encoded in decreasing order of rate-distortion slope. That is, the coefficients that yield the largest reduction of distortion, in the mean-squared error sense, are encoded first. In our work, we maintain the properties of *CEZW* while further exploiting the redundancy between color components to obtain better compression performance.

A distinction between our implementation of *CEZW+* and the *CEZW* implementation presented in [4], lies in the order in which the color components are encoded. In [4], all the bands in the Y component are encoded first, followed by those of the U and V components. In *CEZW+*, we order the color components on a per-band basis. That is, the *LL* band of the Y component is encoded first, followed by the *LL* band of U and V. Then, the *LH* band of the Y component is encoded, followed by the *LH* band of U and V, and so on. Therefore, we obtain a better balance among color components when decoding is terminated before the end of a pass – a very likely scenario.

In Section 4.4.2, we described a modified *CEZW* algorithm, where we allocate the largest number of bits to the "significant trees." A tree is said to be "significant" when the root at the coarsest scale is positive or negative significant in the first dominant pass. In the remaining dominant passes, until the bits assigned to the frame are exhausted, only coefficients that belong to the "significant trees" are examined.

In this Section, we develop a framework for improving the identification of the "significant trees" (which were described in Section 4.4) based on a rate-distortion analysis [177] of wavelet coefficients in the decomposition of the PEF. Our overall objective can be stated as follows: "Given a target data rate, minimize the overall distortion of the reconstruction of the coefficients in the wavelet domain, while preserving the embedded and rate scalable properties of the encoded bit stream."

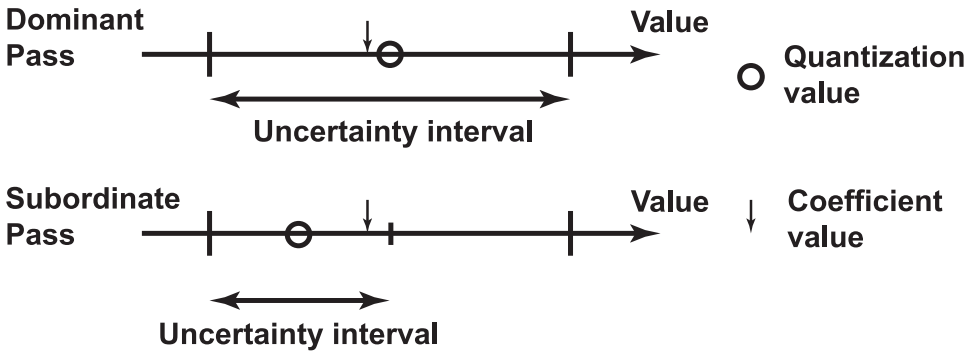


Fig. 4.19. Example where the successive refinement of a wavelet coefficient in *CEZW* does not reduce the distortion between the quantized and actual values, even though the uncertainty interval is smaller.

Our work is based on the observation that in EZW and *CEZW*, a small number of successive refinements ("passes") may not lower the distortion between the quantized and actual values of the wavelet coefficients, even though the uncertainty interval is in fact reduced. For example, consider a wavelet coefficient whose value is equal to the quantization value in the dominant pass. In the subordinate pass, a refinement of the quantized value is done, and the uncertainty interval is reduced in half. However, in this case, the distortion between the quantization value of the coefficient and its actual value increases. This example is illustrated in Figure 4.19. Although this problem is not important for high data rate applications because the number of passes is large, this is an important problem in low data rate applications. In this example, if only a dominant and a subordinate passes were done, it would have been advantageous to only perform the dominant pass.

Let W be the coefficients of the wavelet decomposition of the PEF in subbands LL, LH, HL, and HH. Each SOT in W is treated as an independent coding unit, as done in EZW [85] and *CEZW* [1, 2, 3, 4]. Therefore, the problem can be formulated as follows:

$$\min_{\forall m_j(t_i):t_i \in T} \sum_{\forall t_i \in T} d(m_j(t_i))$$

$$\text{subject to} \sum_{\forall t_i \in T} r(m_j(t_i)) \leq R_{MAX} \tag{4.8}$$

where T is the collection of all the SOTs of the wavelet decomposition W of the PEF, t_i is a SOT in T, $m_j(t_i)$ is an encoding of SOT t_i, $d(m_j(t_i))$ is the distortion incurred when encoding SOT t_i with mapping $m_j(t_i)$ at the data rate $r(m_j(t_i))$, and R_{MAX} is the maximum allowable rate at which the PEF can be encoded. The distortion metric used in our work is the squared error between the wavelet coefficient and its quantized value.

It can be shown that the optimization problem in Equation (4.8) can be solved using Lagrangian optimization as follows [178]:

$$\min_{\forall m_j(t_i): t_i \in T} \left\{ \sum_{\forall t_i \in T} \{d(m_j(t_i)) + \lambda r(m_j(t_i))\} \right\} \tag{4.9}$$

We treat each SOT as an independent coding unit, therefore Equation (4.9) can be rewritten as:

$$\sum_{\forall t_i \in T} \min \{d(m_j(t_i)) + \lambda r(m_j(t_i))\} \tag{4.10}$$

The best operating points for all coding units are obtained by varying λ and minimizing Equation (4.10) such that the sum of the resulting rates $r(m_j(t_i))$, $\forall t_i \in T$, equals the target data rate.

To obtain finer granularity in the solutions to Equation (4.10), we further subdivide the SOT into "subtrees," whose roots are coefficients in the original SOT that are in the LH, HL, and HH bands, respectively. The coefficients in the LL band are treated separately. Thus, a subtree is our basic coding unit. The rate-distortion data is gathered experimentally. We start with an empty SOT. We determine the number of bits necessary to code it, and its distortion, and store these two values. We then use the *CEZW* algorithm, and at the end of each dominant and subordinate passes, we determine the number of bits used to encode the symbols used so far in the SOT, and the current distortion. A set of {rate,distortion} pairs is obtained. We perform our rate-distortion analysis on this set for each SOT in the decomposition,

and determine the best operating point for each SOT. The optimal solution for Equation (4.10) is found using a bilinear search on λ. The best operating points are then used by *CEZW+* to encode the PEF. In contrast to [93], our approach can be seen as a "SOT growth" instead of "SOT pruning," because we first consider an "empty" subtree, and gradually add coefficients to it until we achieve the best operating point. When the best operating point is reached, a "STOP" symbol is produced to indicate that no more symbols will be used for this subtree. The coefficients already found significant are removed from the significant list.

When a "STOP" symbol is sent, no more coefficients are added to the SOT and the coefficients that have already been found significant are removed from the significant list. This is done because if they are kept in the list, they would be allocated bits during remaining subordinate passes, thus going beyond their best operating point.

4.6 Summary of Modifications to *SAMCoW*

In Table 4.1, we present a summary of the modifications to *SAMCoW*. We list the techniques used to overcome the limitations of *SAMCoW*, the limitation each one addresses, and improvement it makes to the performance of *SAMCoW*.

4.7 Experimental Results

In this Section, we present the experimental results obtained by using the modifications to *SAMCoW* described in this Chapter. For comparison purposes, we conducted three experiments which we shall call "*SAMCoW*: AVCT," "*SAMCoW*: PP," and "*SAMCoW+*." This is shown in Table 4.2. In *SAMCoW*: AVCT, we used unrestricted motion vectors, half-pixel accurate motion estimation, B frames (one B frame between each reference frame), and dynamic bit allocation. In *SAMCoW*: PP, we used the same modes as in *SAMCoW*: AVCT, in addition to using preprocessing, the modified *CEZW* algorithm, and postprocessing. Finally, in *SAMCoW+*, we used all the modes used in *SAMCoW*: PP, but replaced modified *CEZW* with *CEZW+*.

In all our experiments, we use QCIF size (176×144 pixels) YUV frames with 4:1:1 chrominance subsampling. For *CEZW* and *CEZW+*, we use four levels of wavelet

Table 4.1
Summary of the modifications to *SAMCoW*

Technique	Limitation of *SAMCoW*	Improvement
Advanced Video Coding Techniques		
Unrestricted motion vectors	Motion vectors must point inside the reference frame	Better compression performance and smoother PEFs in scenes with medium and high motion
Half-pixel accuracy	Restricts displacement of objects to coincide with the sampling grid	Improved motion prediction and smoother PEFs
B frames	Every frame is a reference for other frame	Better performance for streaming, temporal scalability.
Dynamic bit allocation	Bit budget determined at beginning of encoding	Better adaptation to the changing content of video sequence
Preprocessing and Postprocessing Techniques		
Adaptive gain	Nearly zero pixel values generate "noise" during wavelet decomposition	Remove "noise" in the spatial domain
Wavelet shrinkage	Bits wasted encoding "noise" in PEF	Remove "noise" in the wavelet domain
Modified *CEZW*	All coefficients are examined	Examine only coefficients in "significant trees"
Unsharp masking	Blurring effect of the wavelet transform	Enhance edge information
Rate-Distortion Analysis		
CEZW+	Significant coefficients are refined even if distortion is not reduced	Refine coefficients until lowest distortion is obtained

Table 4.2
Organization of the experiments using the modifications to *SAMCoW*. An × indicates that the technique is enabled for these experiments.

Technique	Experiment #1: *SAMCoW* : AVCT	Experiment #2: *SAMCoW* : PP	Experiment #3: *SAMCoW+*
Advanced Video Coding Techniques (AVCT)	×	×	×
Preprocessing, Postprocessing, and Modified *CEZW* (PP)		×	×
CEZW+ (Rate-Distortion Analysis)			×

decomposition, performed using the (9,7) Daubechies wavelet filter pair [75]. For the frame boundaries, we used whole-sample symmetric extension, as described in Section 2.4.1. Adaptive arithmetic coding is used as the entropy coder [54, 55]. We used a GOP size of 100 frames. The use of a large GOP size results in fewer frames being encoded as I frames, which helps maintain the target data rate, since P frames require less bits to encode.

We compare our results with *SAMCoW* and H.263+. We used Release 0.2 of the H.263+ software, obtained from the University of British Columbia, Canada [57, 157]. In particular, we used annexes D (Unrestricted Motion Vectors mode), E (Advanced Prediction mode), and F (Arithmetic Coding mode), with TMN8 rate control and no frame skipping[1].

For our experiments, we use the peak signal-to-noise ratio (PSNR), described in Section 2.1.2, as our "fidelity" measure. The following sequences, target data rates, and frame rates were used for our experiments: the *akiyo* sequence at 16 kbps, 5 frames per second (fps); the *coastguard* sequence at 24 kbps, 10 fps; the *carphone* sequence at 48 kbps, 10 fps; and the *foreman* sequence at 64 kbps, 10 fps.

The *akiyo* sequence contains a typical "head and shoulders" scene of an anchor person during a news broadcast. The background is static throughout the sequence, thus the main difficulty is to accurately encode the detail in the area of the face of the anchor person. The *coastguard* sequence contains scenes with high motion, in that a sudden camera pan occurs and two close objects move in opposite directions. Sudden movements of the camera are difficult because new objects enter the scene very rapidly. The way that the motion estimation reacts to this condition results in a larger prediction error in the new regions of the frame. When two objects are close in a sequence, they may be fully or partially contained in the same macroblock. If they move in different directions while still partially residing in the same macroblock, motion estimation will have problems producing an accurate motion vector.

[1]We used this combination of modes in order to have the same advanced coding options active in H.263+ as in *SAMCoW+*. We believe this is a fair comparison between both compression techniques.

In the *carphone* sequence, a subject talks to the camera while riding in a car. The background inside the car is static. However, due to the motion of the car, a portion of the background of the sequence moves rapidly, as exteriors pass by the window of the car. This sequence is challenging because one region of the frame (the one corresponding to the window) will have a larger prediction error due to the changing background. In the *foreman* sequence, a subject speaks to the camera, while the camera pans up down and to the side. As in the *coastguard* sequence, new background appears with the camera pan.

4.7.1 Experiment #1: *SAMCoW* with advanced video coding techniques (*SAMCoW* : AVCT)

In this Experiment, we present the performance of *SAMCoW* : AVCT, that is, using advanced video coding techniques. In Figures 4.20, 4.21, 4.22, and 4.23, we present the PSNR data of *SAMCoW* : AVCT for the *akiyo*, *coastguard*, *carphone*, and *foreman* sequences. The use of advanced video coding techniques improves the performance of *SAMCoW* : AVCT over *SAMCoW*, for all the test sequences, due to more accurate motion estimation and better bit allocation. This improvement can also be observed in the decoded frames.

In the *akiyo* sequence, a sequence with simple scenes, the use of half-pixel accuracy and dynamic bit allocation improves the performance of *SAMCoW* : AVCT. Half-pixel accuracy describes the motion of an object in the scene more accurately, for example, the facial features of the anchor person, such as her eyes and lips. Dynamic bit allocation provides more adaptability to sudden movements of the objects in the scene, such as when the anchor person moves her head rapidly. The use of UMV is not a factor in the encoding of this sequence since no objects enter or exit the sequence. The coding performance of *SAMCoW* : AVCT improves between the first and the last decoded frames. This is due to the static background in the sequence. More bits are used to encode the regions of the frame where the prediction error is large, such as the face region of the news anchor person. The quality of the decoded frames is improved over *SAMCoW*, although the frames still look washed out.

In the *coastguard* sequence, both unrestricted motion vectors and half-pixel accuracy improve the performance of *SAMCoW* : AVCT. In frames 22-26 of this sequence (see Figure 4.21), the sudden upwards camera pan creates problems for motion estimation and compensation. *SAMCoW* : AVCT recovers more rapidly than *SAMCoW* due to more accurate prediction. The quality of the decoded frames is also improved, in particular the rendition of the color. However, some of the detail of the objects is lost during decoding.

In the *carphone* sequence, *SAMCoW* : AVCT follows closely the performance of *SAMCoW* (see Figure 4.22). For example, near frame 40, and then again near frame 55 of the decoded sequence, *SAMCoW* : AVCT, as well as *SAMCoW* and H.263+, have problems keeping up with the motion of the objects in the scene. Such movements, in which objects appear in the scene for only a few frames, are very difficult to encode for any video compression scheme. This is due to the model assumed for motion estimation, that is, the objects in the scene have only translational motion. Therefore, when this model no longer holds, the prediction error is large and the PEFs are more difficult to encode. However, due to more accurate prediction, an improvement over *SAMCoW* was obtained by using *SAMCoW* : AVCT.

In the *foreman* sequence, the use of UMV and half-pixel accuracy improved the motion prediction. Large periods of camera pan, as seen in frames 75 and later, make it difficult to predict the new objects entering the scene (see Figure 4.23). In this case, the prediction error is large in many regions of the PEF.

In our experiments, the improvement obtained by using *SAMCoW* : AVCT over *SAMCoW* is 0.5 dB on average. A shortcoming of *SAMCoW* : AVCT is that *CEZW* is unable to allocate more bits to regions where the prediction error is large. In such situations, we expect the techniques used in *SAMCoW* : PP to perform better.

4.7.2 Experiment #2: Preprocessing and postprocessing techniques (*SAMCoW* : PP)

In this Experiment, we present the performance of *SAMCoW* : PP. For the adaptive gain function, we use: $K = 2.0$, $t_1 = 0.5\sigma$, $t_2 = 2.0\sigma$, and $t_3 = 3.0\sigma$, where σ is

the standard deviation of the pixel values of the PEF. For wavelet shrinkage, we use: $\alpha = 2.0$, and $T_{max} = \sigma_J^{LL}$, and $T_{min} = \sigma_1^{HH}$. For postprocessing, we use $\lambda = 2.0$.

First, we present the performance of the preprocessing on predictive error frames. PEFs from frames 29 and 35 of the *akiyo* and *foreman* sequences are shown in Figures 4.24 and 4.25. The PEFs after preprocessing are shown in Figures 4.24(b) and 4.25(b). After preprocessing, the regions of the predictive error frames where the prediction error is large are isolated from the regions where the error is small. Preprocessing sets to zero the regions where the prediction error is small, allowing the modified *CEZW* algorithm to allocate more bits to regions where motion estimation is less accurate.

Next, we present the performance of preprocessing, the modified *CEZW* algorithm, and postprocessing on predictive error frames. The PEFs in Figures 4.24(a) and 4.25(a) after encoding and decoding using *CEZW* at 0.25 bpp, are shown in Figures 4.24(c) and 4.25(c). Figures 4.24(d) and 4.25(d) show the PEFs in Figures 4.24(a) and 4.25(a) after preprocessing, the modified *CEZW* algorithm, and postprocessing, at the same data rate (0.25 bpp). As can be seen in Figures 4.24(d) and 4.25(d), the regions where the prediction error is large show more detail and definition, and look sharper. On the other hand, the background areas are set to zero by the preprocessing and effectively ignored, thus we are able to allocate more bits to the regions with larger prediction error. The better allocation of bits and the use of an unsharp masking filter rendered the images sharper when compared with *CEZW*.

In Figures 4.26, 4.27, 4.28, and 4.29, we present the PSNR data of *SAMCoW* : PP for the *akiyo*, *coastguard*, *carphone*, and *foreman* sequences. The use of preprocessing, the modified *CEZW* algorithm, and postprocessing improved the performance of *SAMCoW* : PP. For the *akiyo* sequence, the improvement over *SAMCoW* is noticeable in the decoded frames. We can attribute this large performance gain to the fact that we can better localize the regions of the frame where motion estimation did a poor job. In the *coastguard* sequence, we also observed that *SAMCoW* : PP improved the quality of decoded frames over *SAMCoW*, and suffers less when the camera pan

occurs around frame 25 (see Figure 4.27). This is due to the allocation of more bits to the new regions of the frame.

In the *carphone* sequence, the results of using *SAMCoW* : PP are better than *SAMCoW*, when observing the quality of the decoded frames (see 4.28). The performance drop around frame 60 still exists, although it is not as large as in *SAMCoW* : AVCT and in *SAMCoW*. There is also an improvement in the *foreman* sequence when using *SAMCoW* : PP (see Figure 4.29). This is also due to better allocation of the bits to the regions where the motion estimation is nor very effective due to camera pan.

Note the large PSNR value of the first frame in the sequences in Figures 4.26, 4.27, 4.28, and 4.29. This is due to a variation in the algorithm used to determine the number of bits allocated to the first encoded frame (an I frame). However, after an initial gain in the PSNR of the sequence, it tends to level out. Therefore, it may be advantageous to reduce the number of bits assigned to the first frame and allocate the extra bits to the PEFs in the GOP.

On average, the gain in compression performance of *SAMCoW* : PP is 1-3 dB over *SAMCoW*. It is important to note that the combination of the advanced video coding techniques and preprocessing, the modified *CEZW* algorithm, and postprocessing are the cause of the performance improvement when using *SAMCoW* : PP. We believe that although the techniques can be used independently, the combination of both yields the best possible results.

4.7.3 Experiment #3: *SAMCoW+*

In this Experiment, we present the performance of *SAMCoW+*, that is, *SAMCoW* with the use of advanced video coding techniques, preprocessing, postprocessing, and *CEZW+* (rate-distortion analysis). For the preprocessing and postprocessing stages, we use the same parameters as in Experiment #2.

We initially present the performance of *CEZW+*. In Figures 4.30(d) and 4.31(d), we show the result of using *CEZW+* on PEFs from the *akiyo* and *foreman* sequence, respectively (Figures 4.30(a) and 4.31(a)). We compare it against *CEZW* (Fig-

ures 4.30(b) and 4.31(b)) done on the original PEFs, and against the PEFs after preprocessing, the modified *CEZW* algorithm, and postprocessing (Figures 4.30(c) and 4.31(c)). More detail in the area of the face can be seen in the PEFs decoded using *CEZW+*. This is because in *CEZW+*, we are able to determine exactly what is the best operating point of each SOT in the wavelet decomposition of the image. By determining this operating point, we can stop the refinement process of the wavelet coefficients when they reach this point, therefore reducing their distortion.

Next, we present the performance of *CEZW+* when used on "natural" images. In Figure 4.32(c), *CEZW+* is used on the first frame of the *foreman* sequence (Figure 4.32(a)). The reconstruction is sharper when compared with *CEZW* (Figure 4.32(b)) done on the original image. Although we developed *CEZW+* to improve the performance of *CEZW* on PEFs, *CEZW+* is not limited to them. The same concept of determining the best operating point of a SOT in a PEF applies to "natural" images, and we are able to improve the performance of *CEZW* using rate-distortion analysis.

In Figures 4.33, 4.34, 4.35, and 4.36, we present the PSNR data of *SAMCoW+* for the *akiyo*, *coastguard*, *carphone*, and *foreman* sequences. On average, the improvement of the compression performance of *SAMCoW+* over *SAMCoW* is 1.5-3.0 dB. This is due to the better utilization of the bits assigned to the PEFs. In *akiyo*, *SAMCoW+* performs better than *SAMCoW*, yet its performance is well below that of H.263+. This is because H.263+ is better able to allocate most of the bits in the PEF to the area of the face of the news anchor person. Since the background is static, the skip mode is used heavily on this region of the frames. The performance of *SAMCoW+* is very similar to that of H.263+ in the *coastguard* sequence, because we allocate more bits to the regions of the PEF where the prediction error is large (see 4.33). In the *carphone* and *foreman* sequences, the performance of *SAMCoW+* is slightly worse than H.263+ This is mainly due to the use of the skip mode in H.263+. (see Figures 4.35 and 4.36). It is important to point out the very similar PSNR curves for all three techniques. This is because they all use block-based motion estimation and, therefore, the prediction loop produces similar results.

Table 4.3
Experimental results: Average PSNR (in dB) of the *akiyo*, *coastguard*, *carphone*, and *foreman* sequences. The average PSNR is taken over all the frames in the decoded sequence.

Sequence	*akiyo*	*coastguard*	*carphone*	*foreman*
Target data rate (kbps)	16	24	48	64
Frame rate (fps)	5	10	10	10
Number of frames	50	66	100	100
Technique	**Average PSNR (in dB)**			
SAMCoW+	34.93	30.43	34.75	33.36
SAMCoW : PP	35.82	30.11	34.29	33.15
SAMCoW : AVCT	33.08	29.35	32.63	31.75
SAMCoW	32.68	29.04	31.98	31.27
H.263+	37.89	30.74	35.61	35.11

4.7.4 Comments on the performance of *SAMCoW+*

To facilitate the comparison between the modifications to *SAMCoW* presented in this Chapter, H.263+, and *SAMCoW*, we present the PSNR of all these techniques in Figures 4.37, 4.38, 4.39, and 4.40, for the *akiyo*, *coastguard*, *carphone*, and *foreman* sequences. The average PSNR (in dB) for the decoded sequences using the modifications to *SAMCoW*, is presented in Table 4.3. The average is taken over all the frames in the decoded sequence. It can be observed that *SAMCoW* : AVCT has slightly better compression performance than *SAMCoW*, an average of 0.5 dB. This is because of the use of advanced video coding techniques.

The compression performance of *SAMCoW* : PP and *SAMCoW+* is similar, due to the use of preprocessing, postprocessing, the modified *CEZW* algorithm, and *CEZW+*. The ability to allocate bits to regions of the frames where the prediction error is larger increases the performance of both techniques over *SAMCoW* between 1.0-2.5 dB

Decoded frames from *SAMCoW*, *SAMCoW*: AVCT, *SAMCoW*: PP, *SAMCoW+*, and H.263+ are shown in Figures 4.41, 4.42, 4.43, and 4.44. The quality of the frames encoded using *SAMCoW*: AVCT is better than the frames encoded using *SAMCoW*, for all sequences. This is due to the use of advanced coding techniques. We observed more detail, smaller washed out regions, and sharper edges in all sequences. However, the quality of the frames is considerably below that of H.263+ at the same data rates. This is because *CEZW* cannot allocate more bits to the regions where the prediction error is large, such as the face of the subjects in Figures 4.41(c) and (d), 4.43(c) and (d), and 4.44(c) and (d), and the coastguard vessel and the boat in Figure 4.42(c) and (d).

A significant improvement can be noticed between *SAMCoW*: PP and both *SAMCoW*: AVCT and *SAMCoW*. Coding artifacts, such as wash out and discoloration, are reduced by the combination of better bit allocation and more accurate motion prediction. In particular, note in Figure 4.42(e) the better definition of the coastguard vessel, and sharper features of all the objects in the scene. The frame in Figure 4.43(e) is of better quality than the ones in Figure 4.43(c) and (d) because of more accurate motion prediction. However, it can be noted that motion estimation and compensation of *SAMCoW*: PP can be improved, because discoloration artifacts can be seen in the region near the neck of the subject in Figure 4.43(e).

The quality of the decoded frames from *SAMCoW+* is better than the decoded frames from *SAMCoW*. In particular, Figure 4.41(f) is much sharper than Figure 4.41(c). More detail can be observed in the rocks in Figure 4.42(f) than in Figure 4.42(c). The area of the tie and the background scene outside the window in Figure 4.43(f) is more neat than Figure 4.43(c). The face area in Figure 4.44(f) is more defined than in Figure 4.44(c).

We also observed that the quality of the decoded frames from *SAMCoW+* is comparable to the decoded frames from H.263+ in all four sequences. It is important to note that *SAMCoW+* is a continuous and fully rate scalable, embedded video compression technique, while H.263+ provides layered rate scalability. A difference

exists between using an embedded scheme such a *SAMCoW+*, or a layered scheme to obtain rate scalability. This difference is in the way motion estimation is performed. In *SAMCoW+*, the motion prediction is performed using a reference frame formed using the lowest data rate in the scalable range (R_L in Figure 3.2). This limits the performance of motion prediction at higher data rates, because the quality of the reference frame used for motion estimation is low. In contrast, in layered schemes it is possible to perform motion estimation using different reference frames for different data rates, therefore obtaining a more accurate motion prediction at higher data rates. It is important to note that the use of multiple layers introduces overhead that can affect the compression performance of layered schemes, as it is the case in H.263+ [104].

The problems that limit the compression performance of *SAMCoW+* can be summarized as follows:

- Motion estimation: Obtaining accurate motion estimation is essential to increase the compression performance of *SAMCoW+*. To address this problem, *SAMCoW+* uses overlapped block motion compensation (OBMC), unrestricted motion vectors, and half-pixel accuracy. OBMC is also used in *SAMCoW*. The use of UMV and half-pixel accuracy in *SAMCoW+* helps in obtaining predictive error frames where the prediction error is smaller, therefore improving the compression performance of *CEZW* on PEFs.

- Coding of the predictive error frames: It is necessary to allocate the bits to areas of the PEF where the performance of motion prediction is low. This is a difficult problem in full-frame wavelet-based compression because spatial orientation trees (SOT) span overlapping areas in the image. That is, a SOT cannot be used to independently code a particular region of the image. To address this problem, we introduce the use of preprocessing, postprocessing, a modified *CEZW* algorithm, and *CEZW+*. These techniques allow *CEZW* to allocate more bits to the regions of the PEF where the prediction error is large.

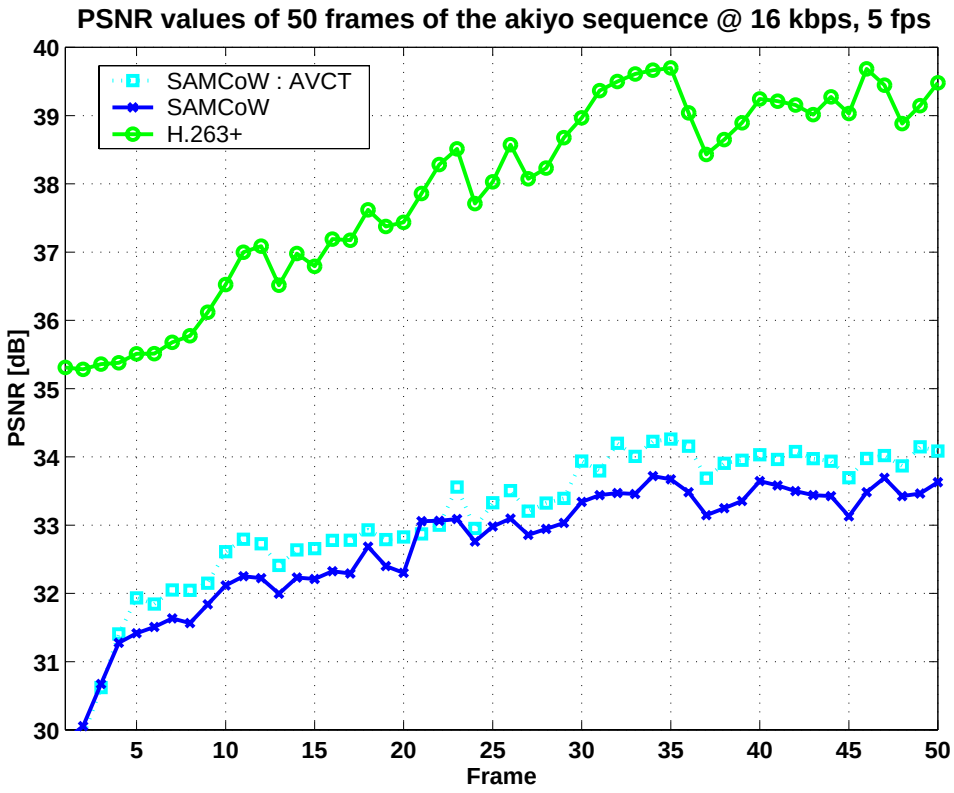


Fig. 4.20. Experiment #1: PSNR values of 50 frames of the *akiyo* sequence at 16 kbps, 5 fps.

In addition, the implementation in *CEZW* of a mode similar to skip mode in H.263+ is difficult. This mode can be used in scenes with static background.

Additional techniques may be used to furter improve *SAMCoW+*. We will describe them in Chapter 6.

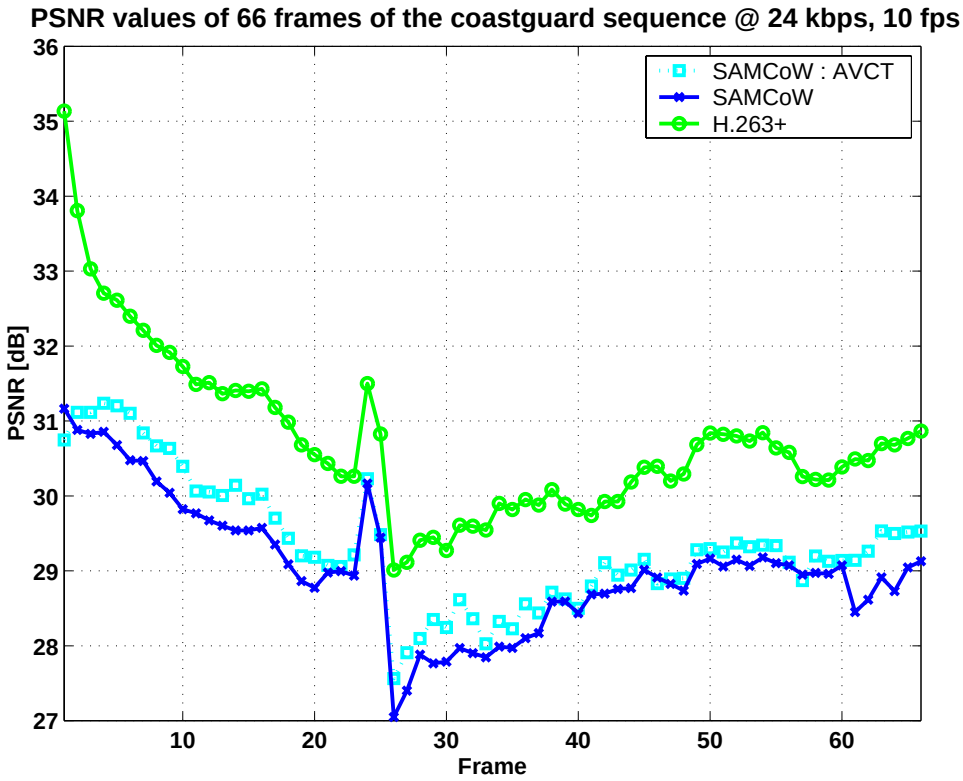


Fig. 4.21. Experiment #1: PSNR values of 66 frames of the *coastguard* sequence at 24 kbps, 10 fps.

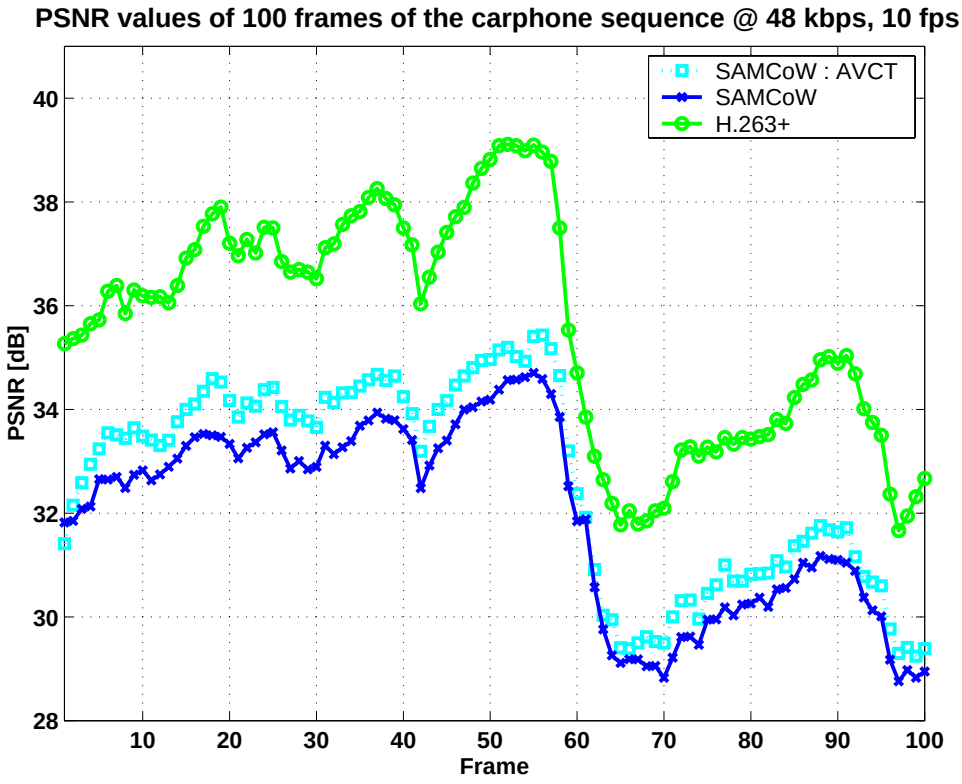


Fig. 4.22. Experiment #1: PSNR values of 100 frames of the *carphone* sequence at 48 kbps, 10 fps.

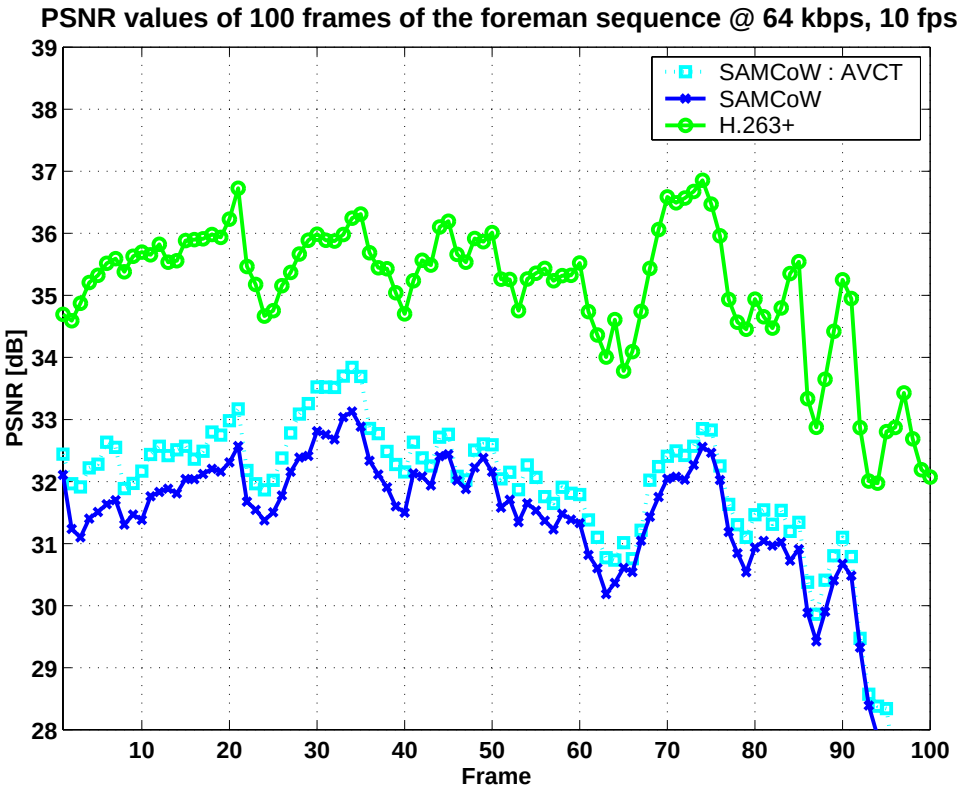
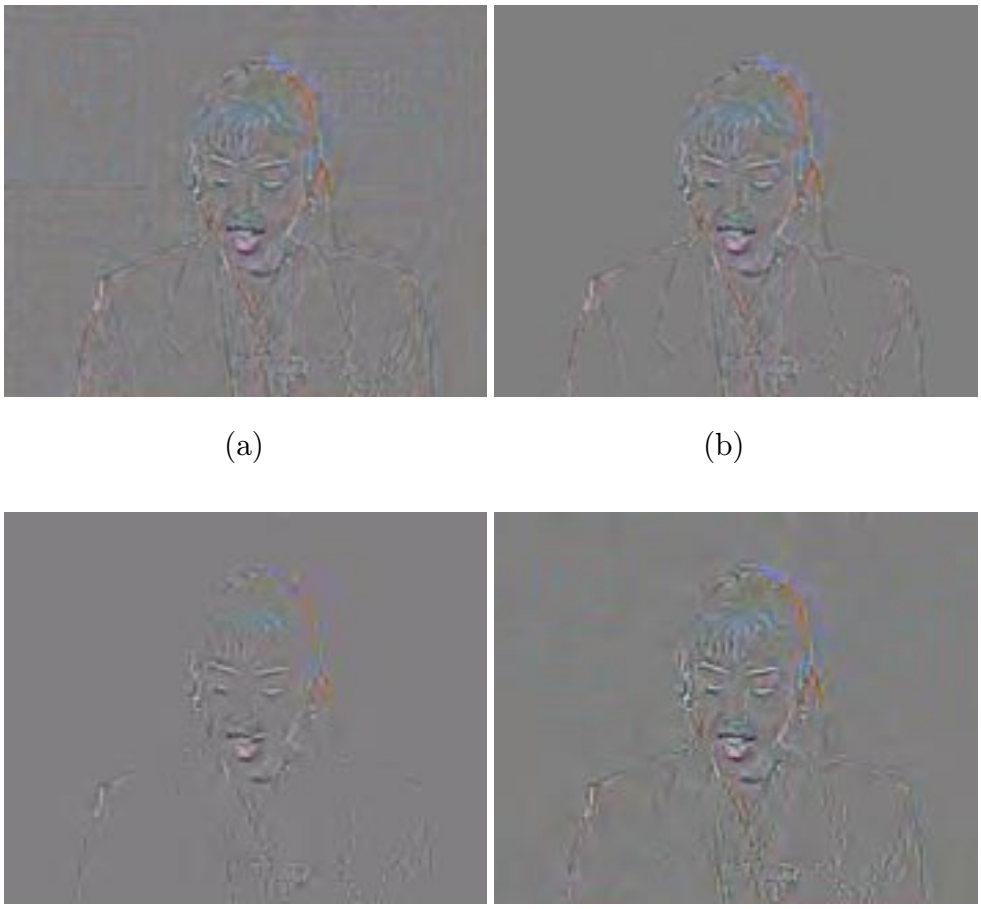


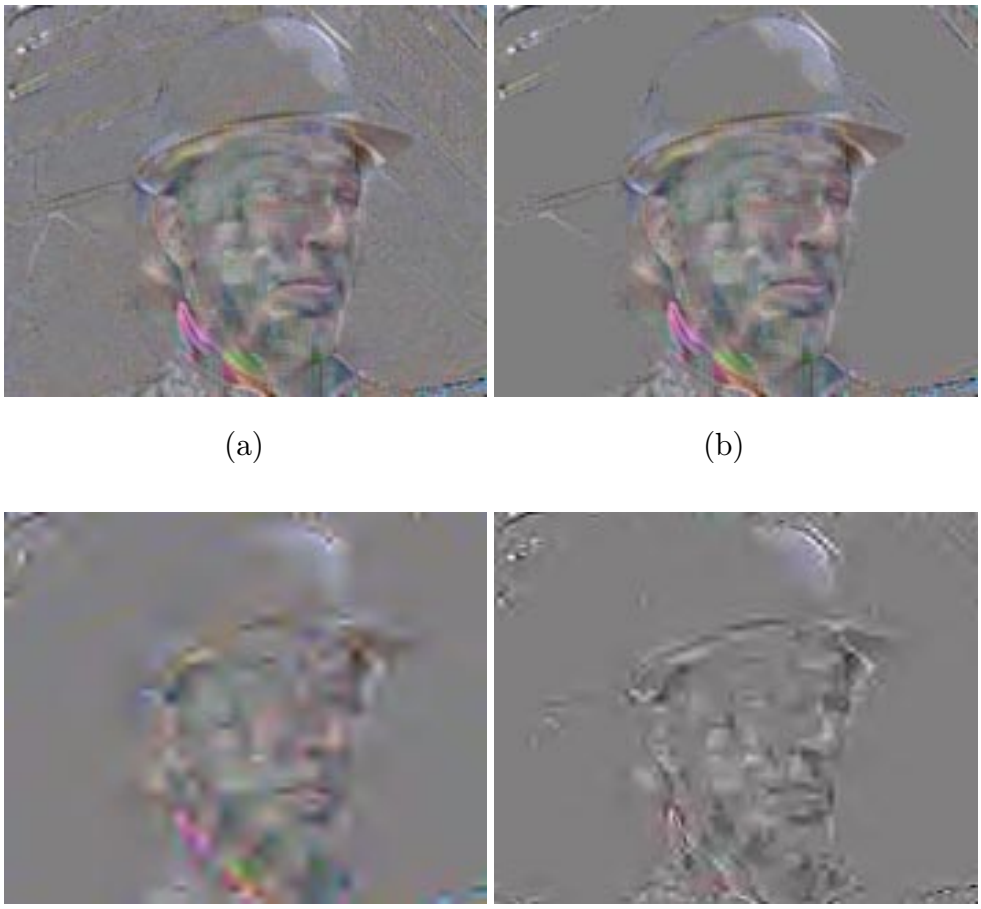
Fig. 4.23. Experiment #1: PSNR values of 100 frames of the *foreman* sequence at 64 kbps, 10 fps.



(a) (b)

(c) (d)

Fig. 4.24. Experiment #2: (a) PEF from frame 29 of the *akiyo* sequence. (b) PEF after preprocessing. (c) PEF in (a) encoded using *CEZW* and decoded at 0.25 bpp. (d) PEF in (a) after preprocessing, modified *CEZW*, and postprocessing, decoded at 0.25 bpp.



(a) (b)

(c) (d)

Fig. 4.25. Experiment #2: (a) PEF from frame 35 of the *foreman* sequence. (b) PEF after preprocessing. (c) PEF in (a) encoded using *CEZW* and decoded at 0.25 bpp. (d) PEF in (a) after preprocessing, modified *CEZW*, and postprocessing, decoded at 0.25 bpp.

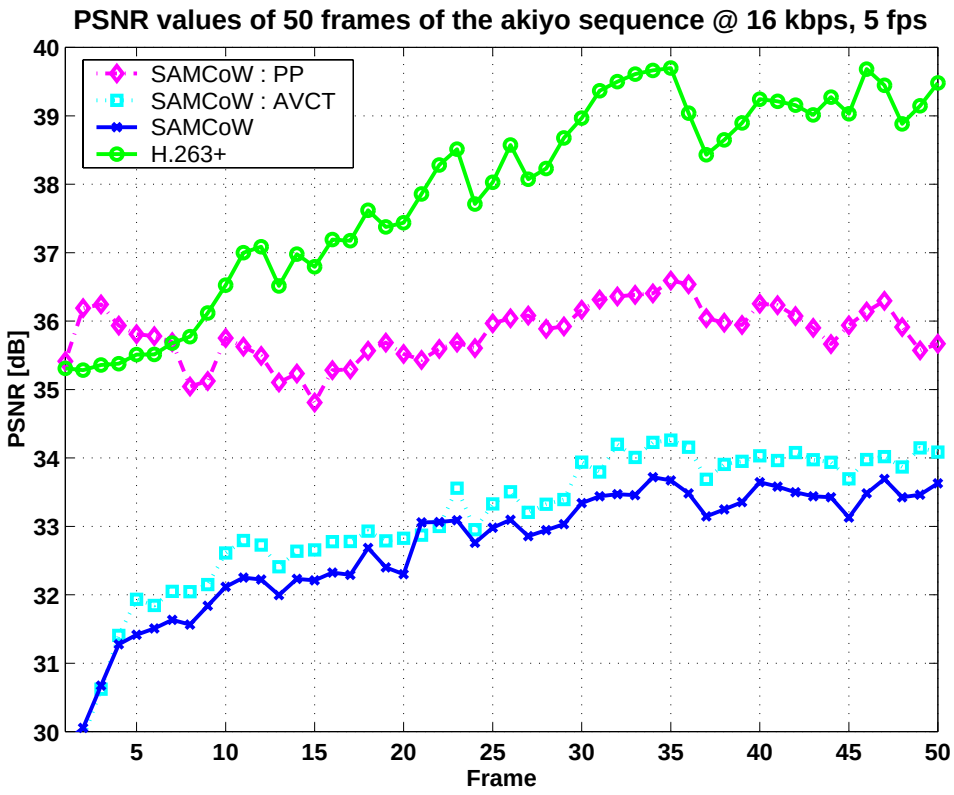


Fig. 4.26. Experiment #2: PSNR values of 50 frames of the *akiyo* sequence at 16 kbps, 5 fps.

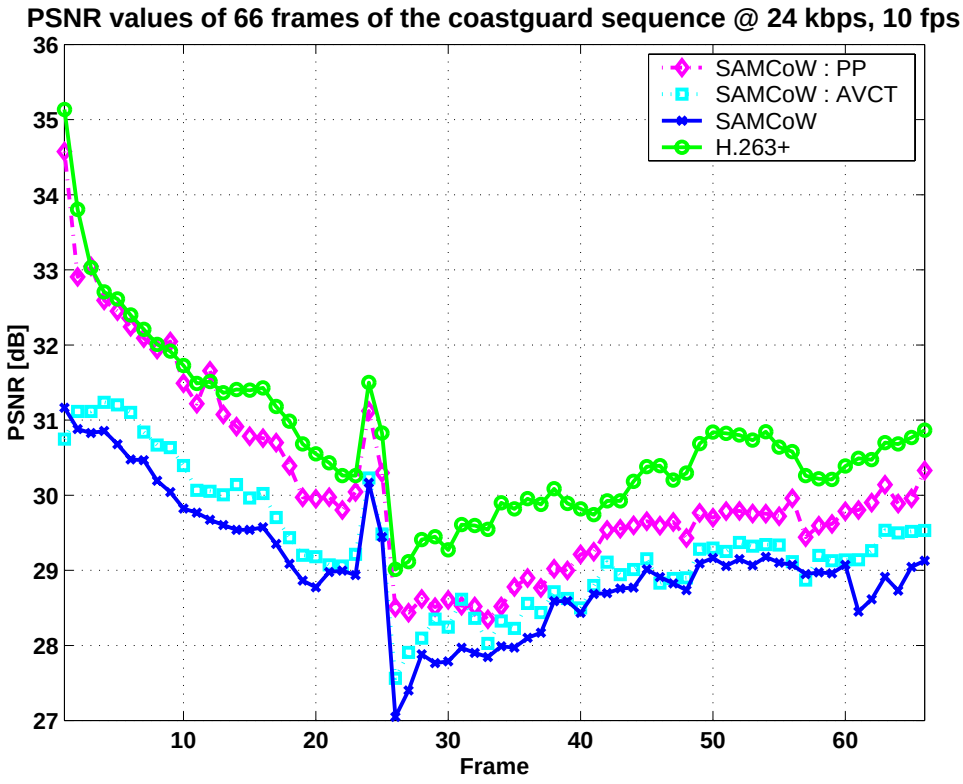


Fig. 4.27. Experiment #2: PSNR values of 66 frames of the *coastguard* sequence at 24 kbps, 10 fps.

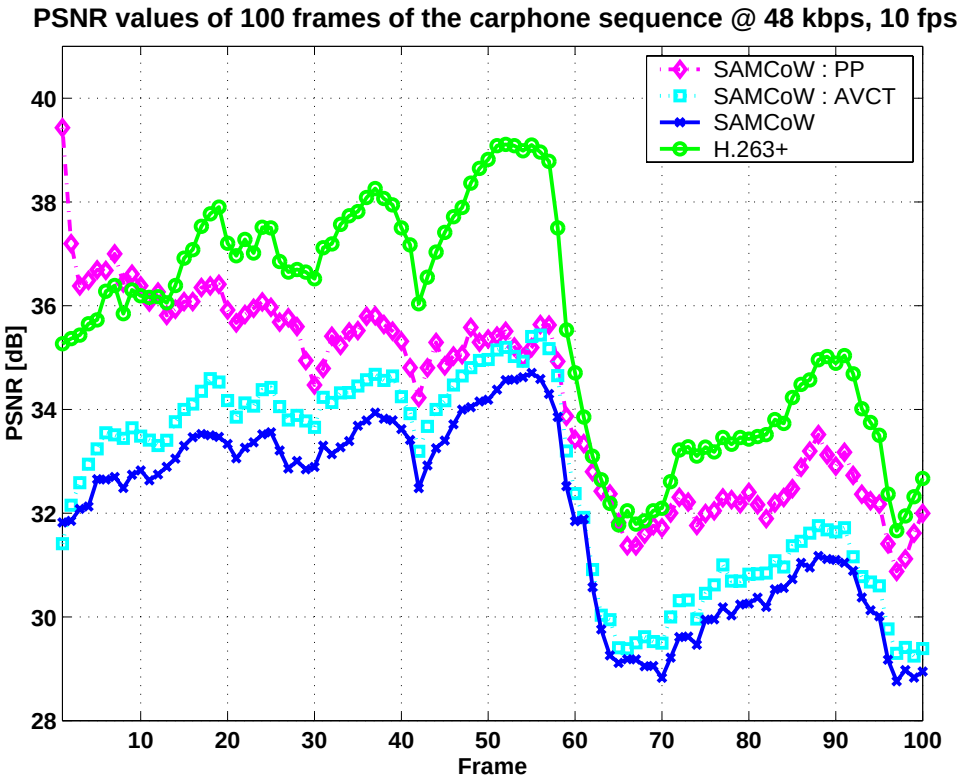


Fig. 4.28. Experiment #2: PSNR values of 100 frames of the *carphone* sequence at 48 kbps, 10 fps.

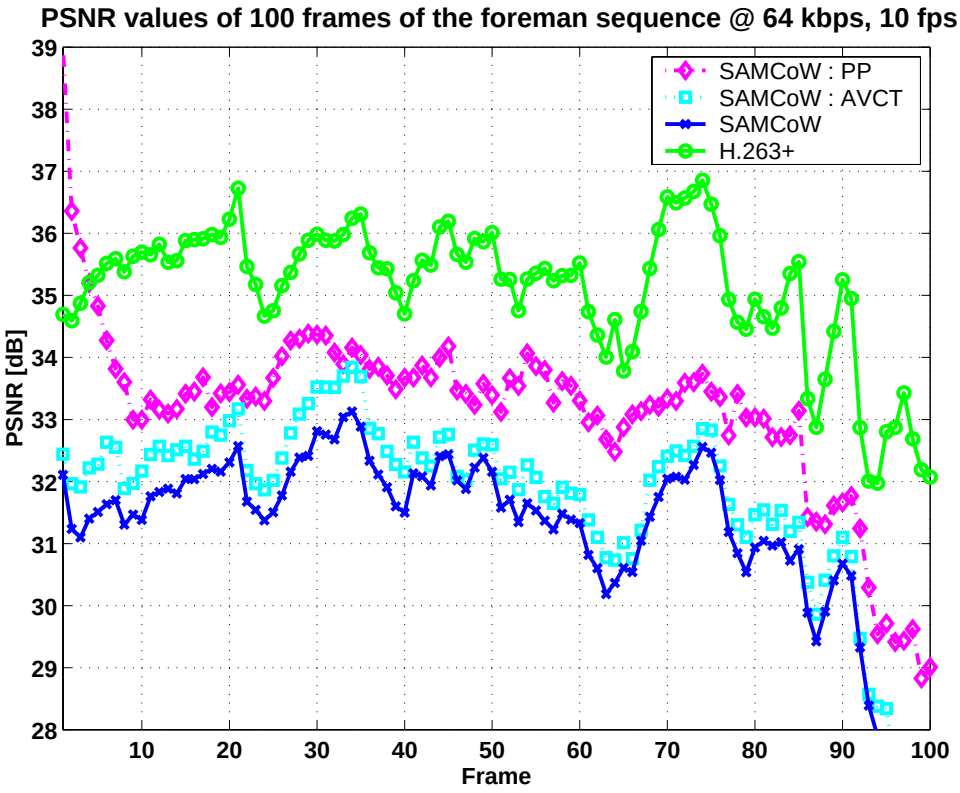
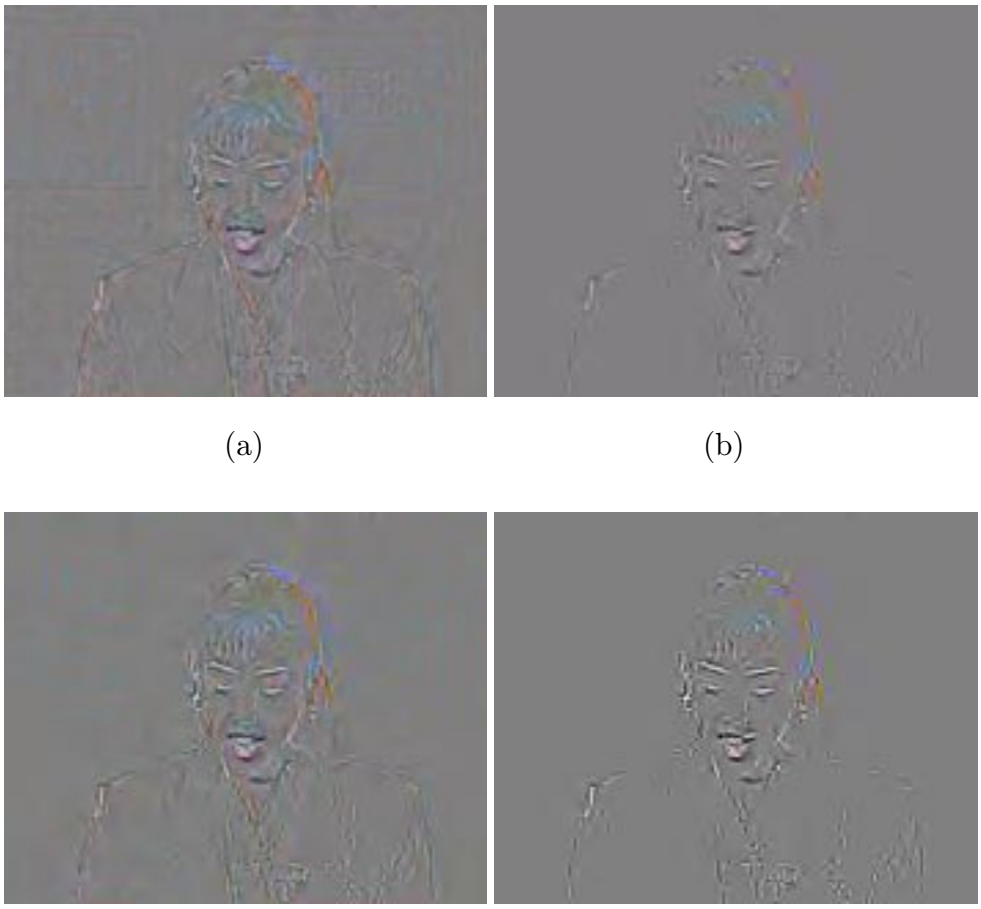


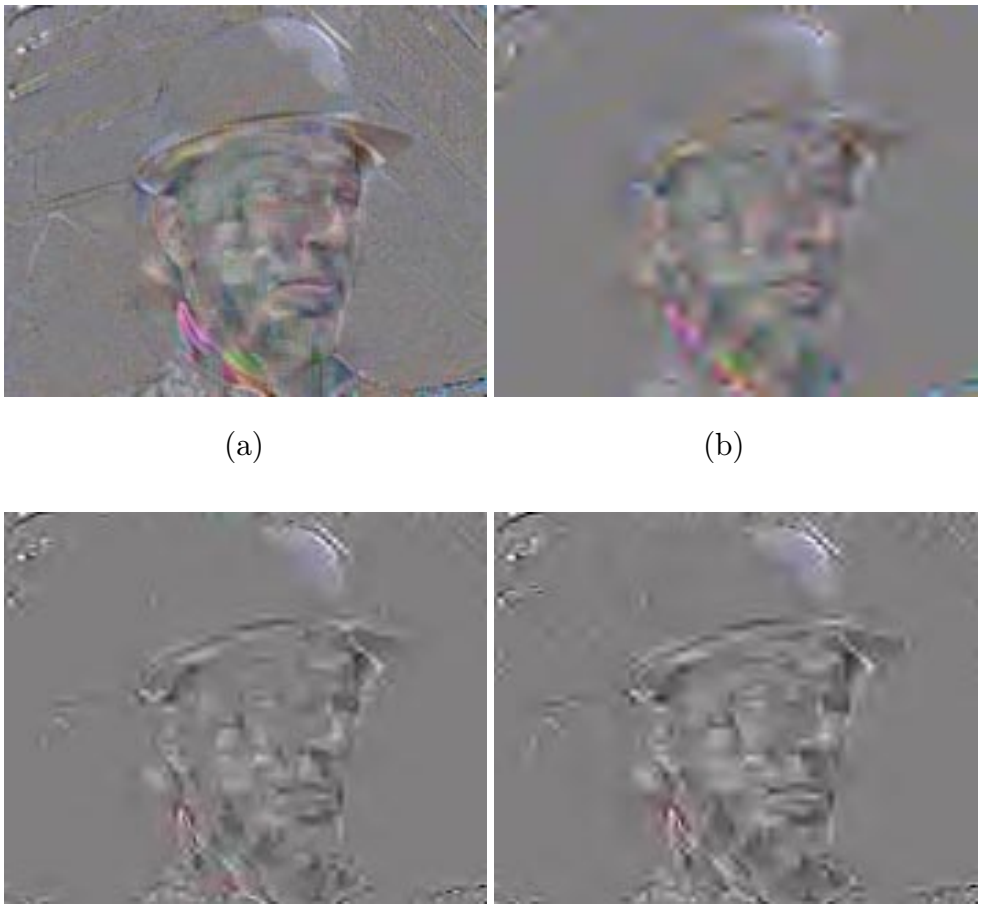
Fig. 4.29. Experiment #2: PSNR values of 100 frames of the *foreman* sequence at 64 kbps, 10 fps.



(a) (b)

(c) (d)

Fig. 4.30. Experiment #3: (a) PEF from frame 29 of the *akiyo* sequence. (b) PEF in (a) encoded using *CEZW* and decoded at 0.25 bpp. (c) PEF in (a) after preprocessing, modified *CEZW*, and postprocessing. (d) PEF in (a) encoded using *CEZW+* and decoded at 0.25 bpp.



(a) (b)

(c) (d)

Fig. 4.31. Experiment #3: (a) PEF from frame 35 of the *foreman* sequence. (b) PEF in (a) encoded using *CEZW* and decoded at 0.25 bpp. (c) PEF in (a) after preprocessing, modified *CEZW*, and postprocessing. (d) PEF in (a) encoded using *CEZW+* and decoded at 0.25 bpp.

(a)

(b) (c)

Fig. 4.32. Experiment #3: (a) The first frame from the *foreman* sequence. (b) Frame in (a) encoded using *CEZW*, and decoded at 1.0 bpp. (c) Frame in (a) encoded using *CEZW+* and decoded at 1.0 bpp.

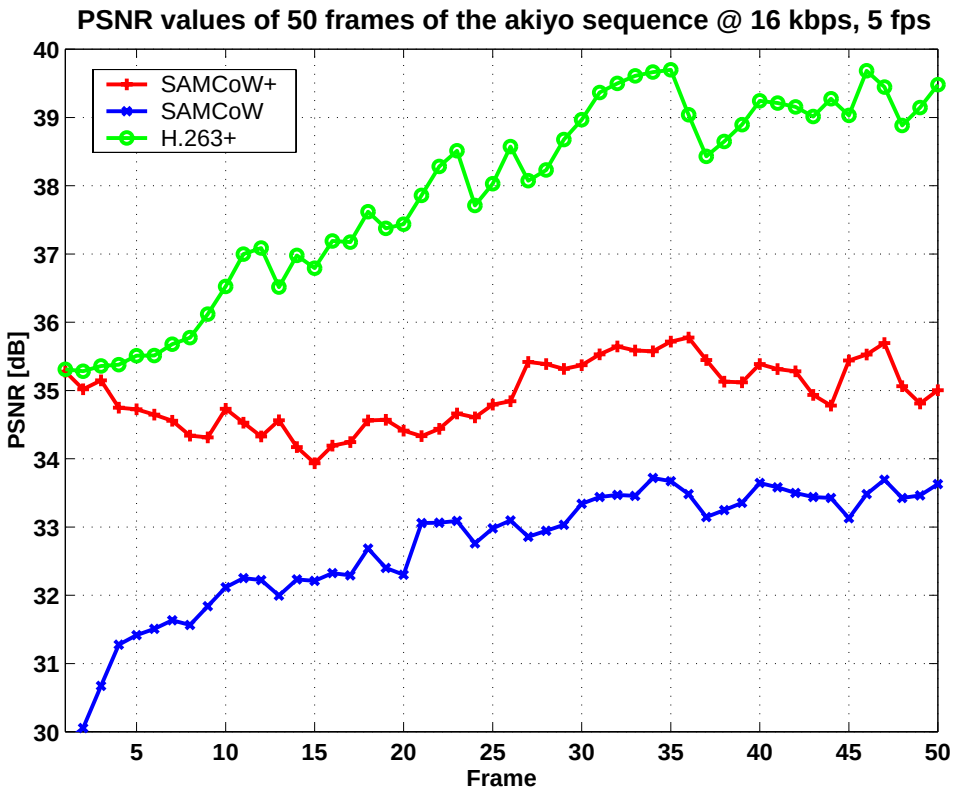


Fig. 4.33. Experiment #3: PSNR values of 50 frames of the *akiyo* sequence at 16 kbps, 5 fps.

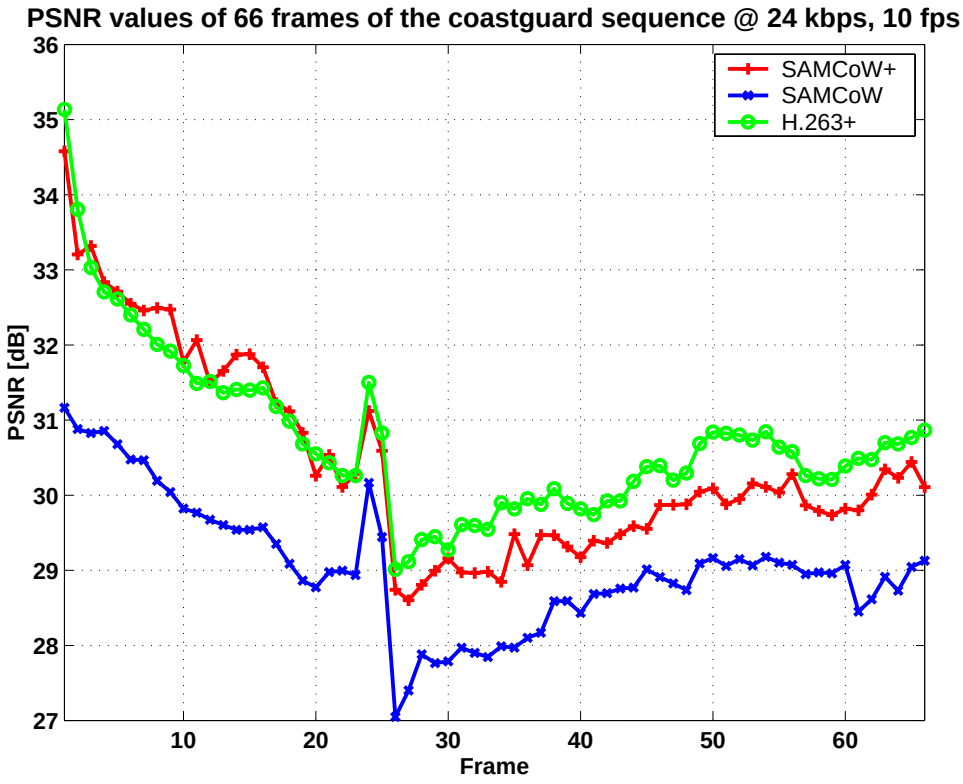


Fig. 4.34. Experiment #3: PSNR values of 66 frames of the *coastguard* sequence at 24 kbps, 10 fps.

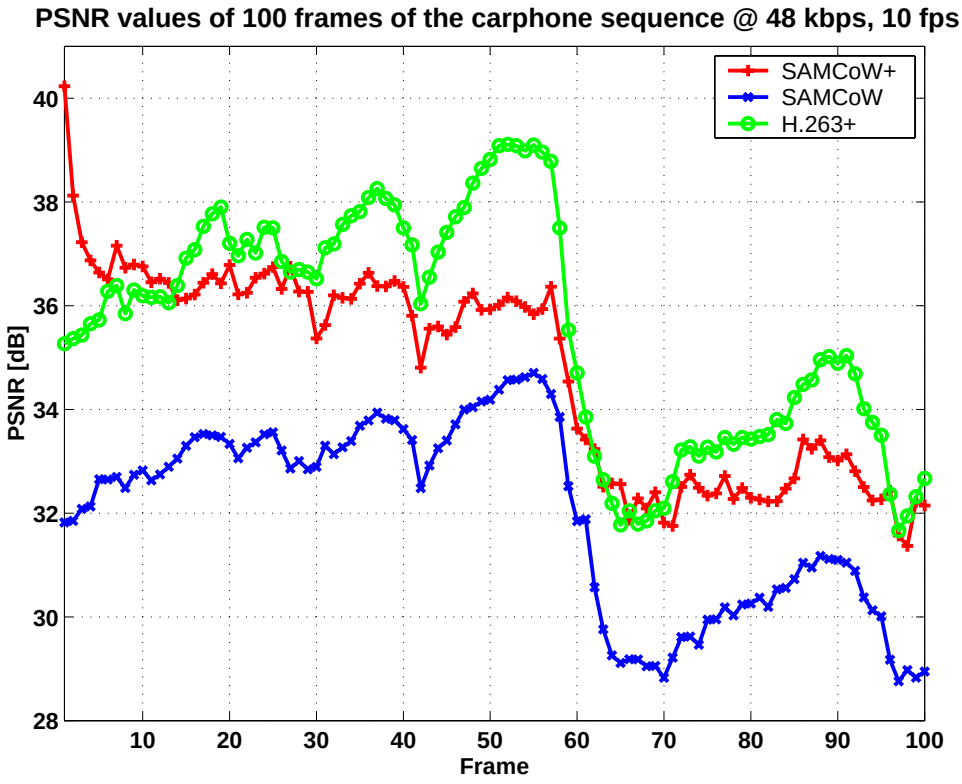


Fig. 4.35. Experiment #3: PSNR values of 100 frames of the *carphone* sequence at 48 kbps, 10 fps.

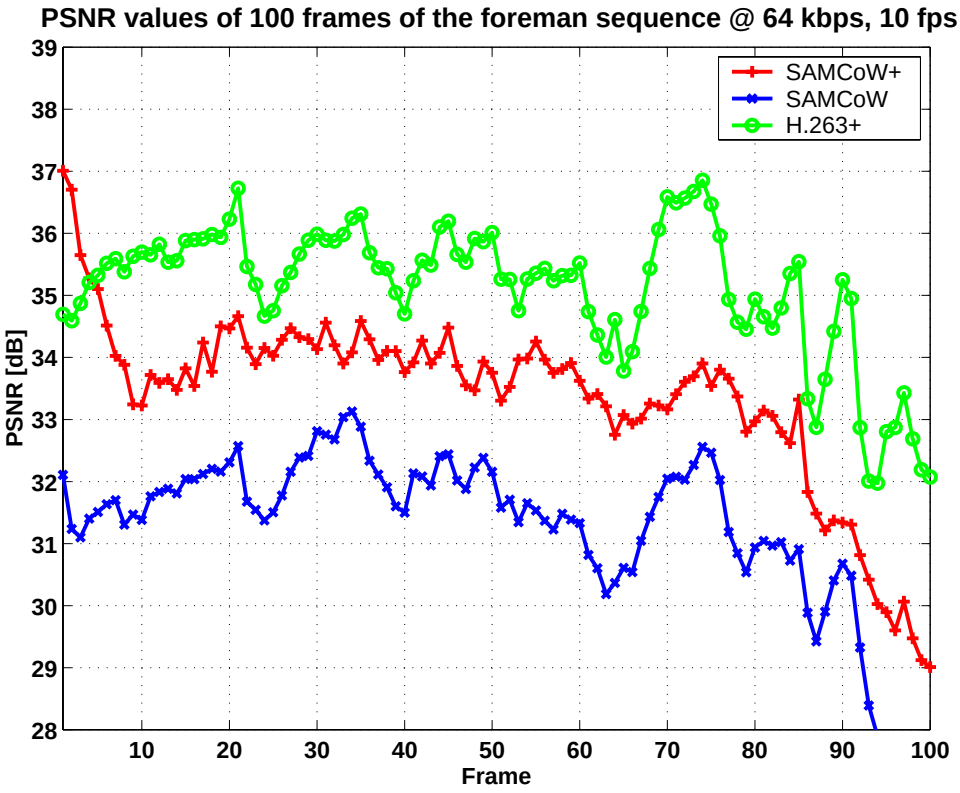


Fig. 4.36. Experiment #3: PSNR values of 100 frames of the *foreman* sequence at 64 kbps, 10 fps.

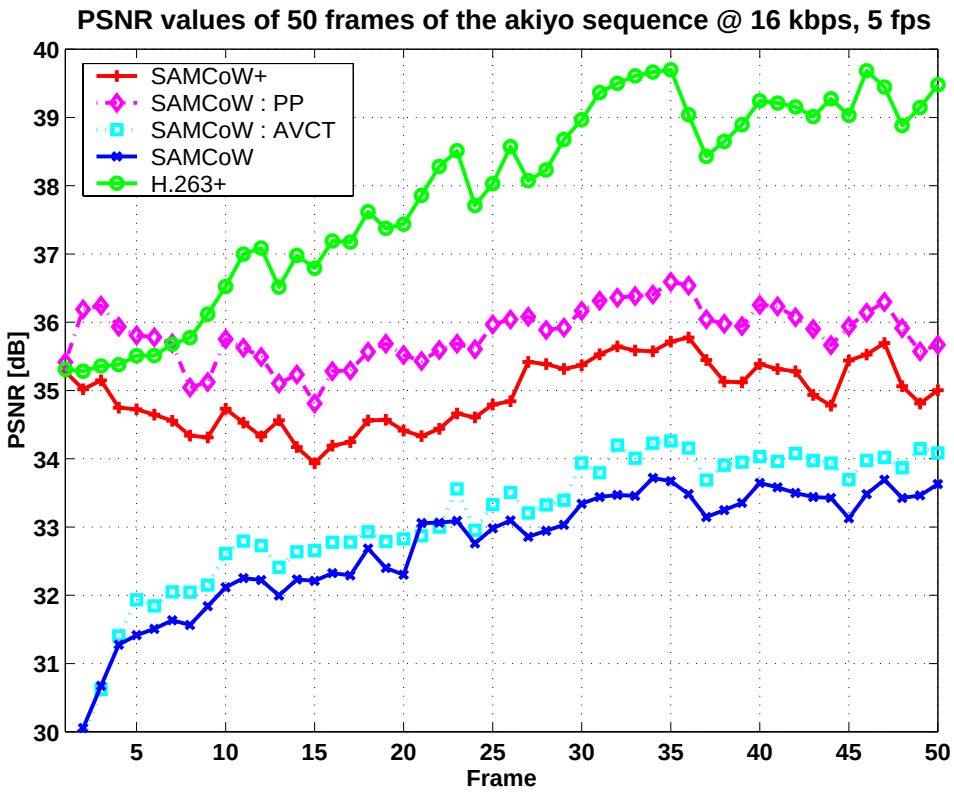


Fig. 4.37. Comparison of Experiments #1, #2, and #3: PSNR values of 50 frames of the *akiyo* sequence at 16 kbps, 5 fps.

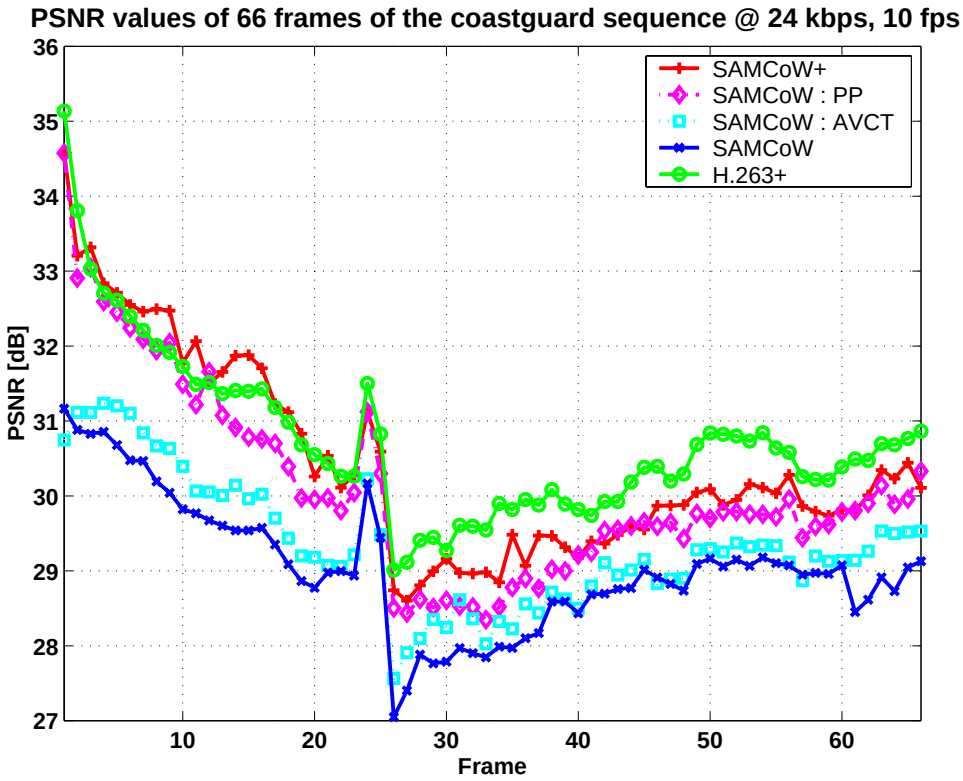


Fig. 4.38. Comparison of Experiments #1, #2, and #3: PSNR values of 66 frames of the *coastguard* sequence at 24 kbps, 10 fps.

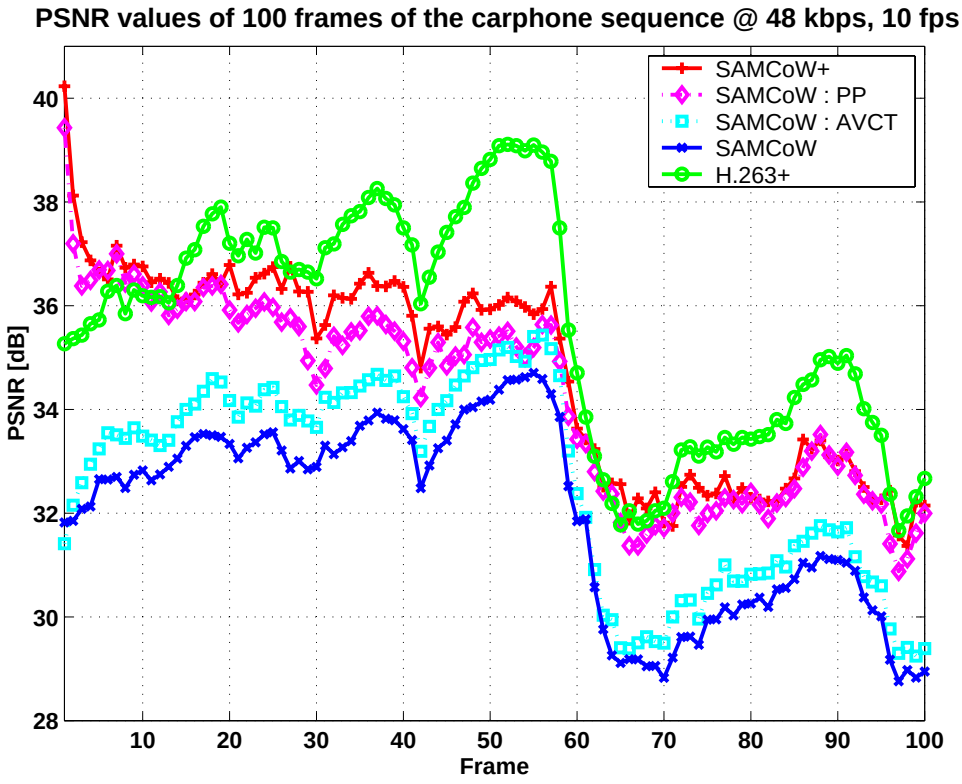


Fig. 4.39. Comparison of Experiments #1, #2, and #3: PSNR values of 100 frames of the *carphone* sequence at 48 kbps, 10 fps.

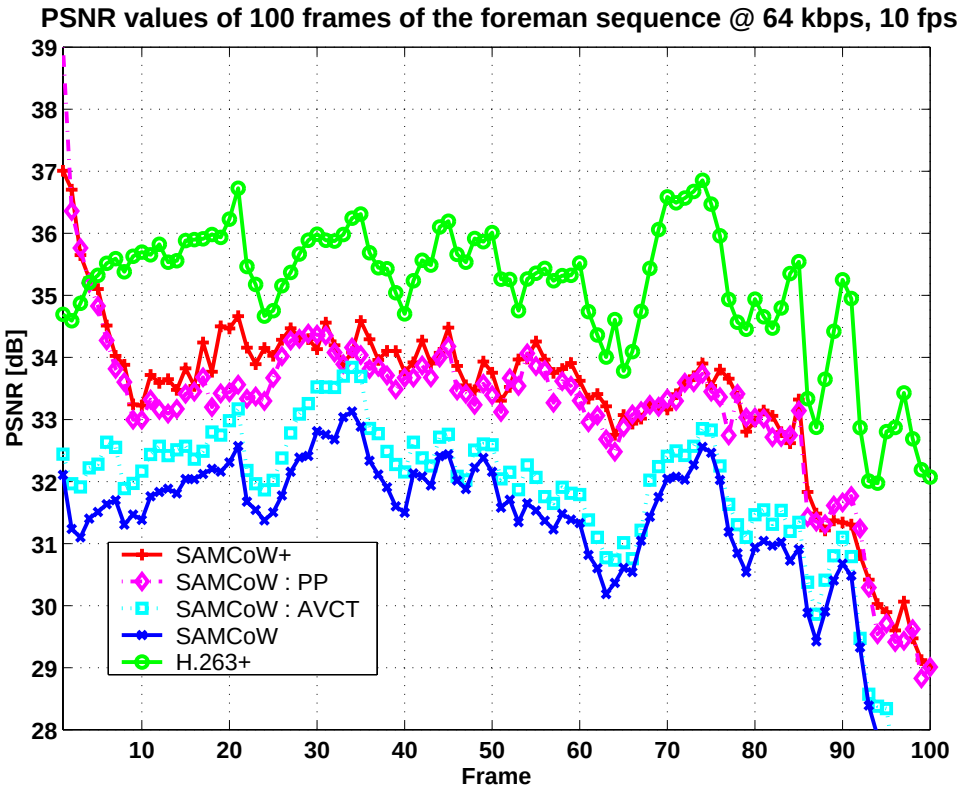


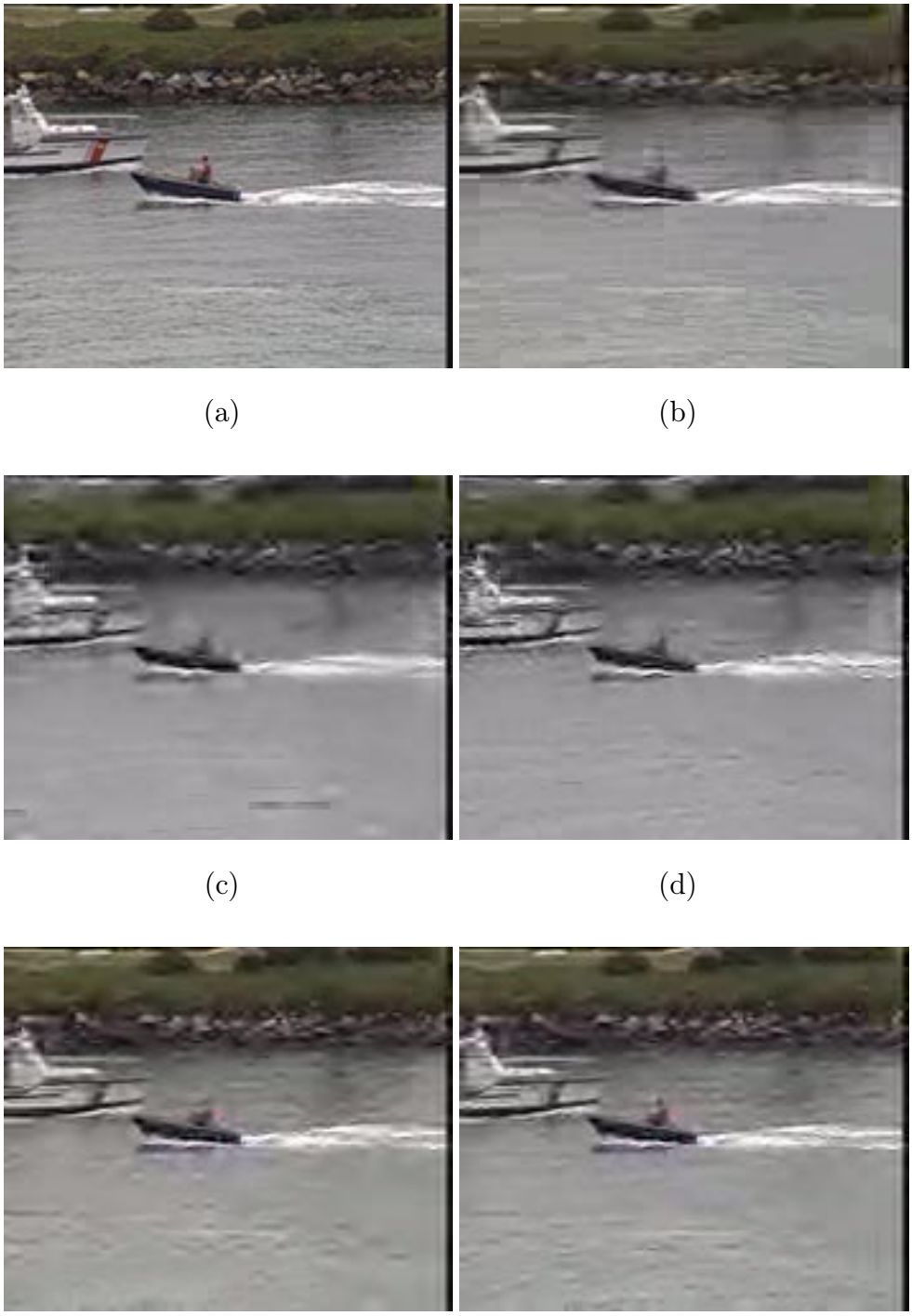
Fig. 4.40. Comparison of Experiments #1, #2, and #3: PSNR values of 100 frames of the *foreman* sequence at 64 kbps, 10 fps.

(a) (b)

(c) (d)

(e) (f)

Fig. 4.41. Comparison of Experiments #1, #2, and #3: Decoded frames of the *akiyo* sequence at 16 kbps, 5 fps. (a) Original. (b) H.263+. (c) *SAMCoW*. (d) *SAMCoW*: AVCT. (e) *SAMCoW*: PP. (f) *SAMCoW+*.



(a) (b)

(c) (d)

(e) (f)

Fig. 4.42. Comparison of Experiments #1, #2, and #3: Decoded frames of the *coastguard* sequence at 24 kbps, 10 fps. (a) Original. (b) H.263+. (c) *SAMCoW*. (d) *SAMCoW*: AVCT. (e) *SAMCoW*: PP. (f) *SAMCoW+*.

(a) (b)

(c) (d)

(e) (f)

Fig. 4.43. Comparison of Experiments #1, #2, and #3: Decoded frames of the *carphone* sequence at 48 kbps, 10 fps. (a) Original. (b) H.263+. (c) *SAMCoW*. (d) *SAMCoW*: AVCT. (e) *SAMCoW*: PP. (f) *SAMCoW+*.

(a) (b)

(c) (d)

(e) (f)

Fig. 4.44. Comparison of Experiments #1, #2, and #3: Decoded frames of the *foreman* sequence at 64 kbps, 10 fps. (a) Original. (b) H.263+. (c) *SAMCoW*. (d) *SAMCoW*: AVCT. (e) *SAMCoW*: PP. (f) *SAMCoW+*.

5. REAL-TIME IMAGE AND VIDEO PROCESSING USING DIGITAL SIGNAL PROCESSORS

The use of Digital Signal Processors (DSP) in consumer applications has increased dramatically in the past few years. This is a result of the flexibility introduced by DSP to the system design due to their programmability, upgradeability, low power consumption, and computational power. DSPs are used widely in wireless communications, DSL and cable modems, personal digital audio players, digital still cameras, IP telephony, printers, and more.

As described in Section 2.5, DSPs can be used effectively for digital image and video processing applications. We investigate problems in real-time video processing. We examine the use of digital signal processors (DSP) for image and video applications, including error concealment for digital video streams and rate scalable image and video compression. We examine new techniques that can be implemented using DSPs [12]. Our initial target processor is the Texas Instruments *TMS320C6000* (*'C6000*).

The approach we used to develop DSP software is the following:

1. A floating-point version of the algorithm is first implemented in C language on a PC. We shall refer to this as "floating-point PC code." This implementation is useful for both debugging and performance evaluation. This software is developed under Microsoft Visual C++ v6.0 and executes only on a PC (Intel x86).

2. Next, we developed fixed-point versions of the algorithm in C on a PC. We shall refer to this as "fixed-point PC code." This was done to investigate the effects of fixed-point arithmetic. This version of the software makes no attempt

to limit its resource usage (such as memory usage or optimizations for smaller or faster code) which may be necessary for implementation on a DSP. Limits due to a fixed-point implementation, such as the effect of dynamic range were studied. Again, this software was developed under Microsoft Visual C++ v6.0 and executes only on a PC (Intel x86).

3. Code we developed for fixed-point implementation on a *TMS320C6201* Digital Signal Processor (DSP) simulator is referred to as "*'C6201* DSP code." From the fixed-point PC code, we developed a version to execute on a *'C6201* DSP. This code executes on a target hardware board based on a *TMS320C6201*, as described below.

***TMS320C6201* hardware platform**

Our hardware platform for the *'C6201* consists of a Detroit board from Spectrum Signal. This board has a rev. 2.1 *'C6201* running at 200 MHz. We also have a Matrox Corona board that can digitize and display still images and video sequences. These boards communicate through the PCI bus for real-time image and video processing.

Development software

The following are the development software tools used in this project:

- Code Composer Studio, *'C6000* Compile Tools (v1.0)
- Code Composer Studio, *'C6000* EVM Tools (v1.0)

5.1 Real-Time Error Concealment in Digital Video Streams

The objective of this work was to implement the spatial and temporal reconstruction techniques described in [11, 137] in real-time. As a proof of concept, our work focused on implementing the fast, sub-optimal error concealment technique [10] using a *'C6000*-based Spectrum Signal Detroit board. Transmitting MPEG video over ATM involves the implementation of a real-time MPEG encoder and decoder and the interface to an ATM switch. We simulated this stage by randomly dropping 8×8

blocks of pixels from uncompressed frames. A block diagram of the system is shown in Figure 5.1.

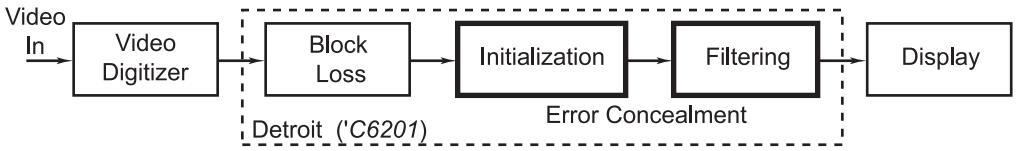


Fig. 5.1. Block diagram of the error concealment system.

5.1.1 Fast near-optimal error concealment

The error concealment technique that we implemented, is based on median filtering, and consists of two stages: initialization and filtering. During initialization, a 3×3 median filter is used to obtain an initial estimate for each of the lost pixels in a block. The pixels in the neighboring undamaged blocks are used as a reference to form the estimate, as shown in Figure 5.2. The second stage, filtering, uses a 3×3 median filter to smooth the pixels in the lost block, using the 8-nearest neighbors as shown in Figure 5.3.

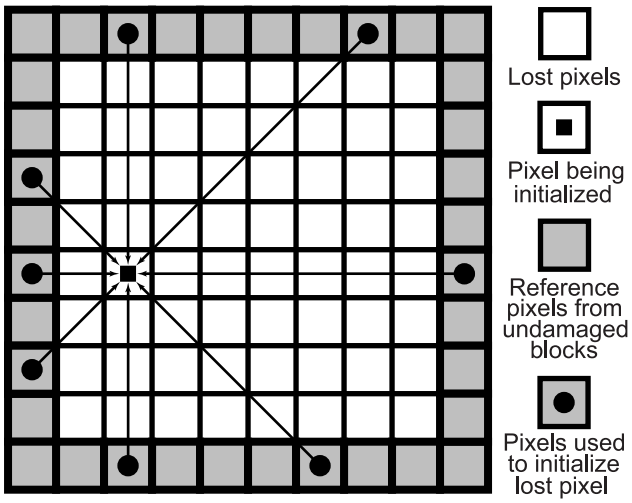


Fig. 5.2. Initialization Stage. Shown are the pixels from adjacent blocks that are used to initialize a particular lost pixel.

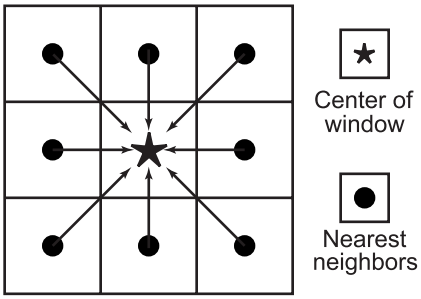


Fig. 5.3. Filtering Stage. The median filter uses 8-nearest neighbors.

5.1.2 System description

We used the *'C6201*-based Detroit board from Spectrum Signal as a computational engine. The Detroit board uses a PC as host, communicating through the PCI bus, and providing direct access to the *'C6000* external memory interface. The Detroit board has random-access memory (RAM) of two types:

- Synchronous Dynamic RAM (SDRAM): The cost of DRAM is much lower and the packaging density is higher than Static RAM (SRAM), but each memory location needs to be accessed periodically (refreshed) to preserve the information stored. Read access time refers to the time elapsed between the processor presenting the address of the location to the memory module, and the memory having the information ready for reading.

- Synchronous Burst RAM (SBRAM): Provides with shorter access time when performing sequential read/write operations. With this type of memory, the processor presents an address to the memory module, but does not need to wait for the data to be made available by memory to present a second address.

Video was digitized using a Matrox Corona board and transferred to the Detroit board, where the frames were processed. After processing is completed, frames were be transferred from the Detroit board back to the Corona board for display. To compare the results of our technique, we displayed:

- The original (undamaged) frames

- The damaged frames, showing missing blocks, or the reconstructed frames after error concealment

The system block diagram is shown in Figure 5.4.

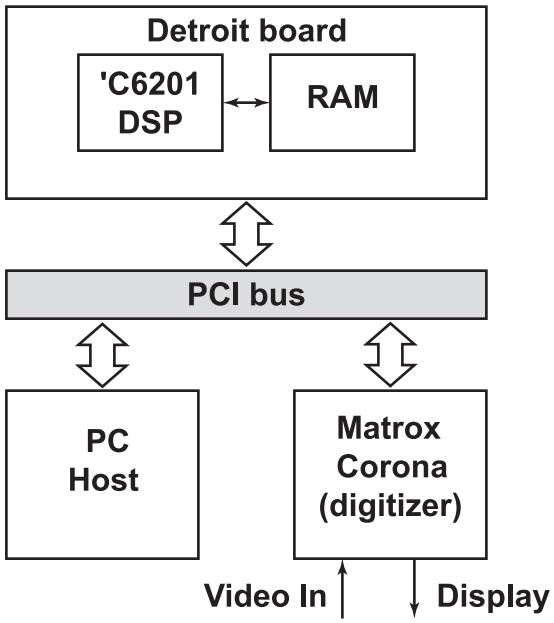


Fig. 5.4. Hardware block diagram of the error concealment system.

5.1.3 Tasks description

The tasks to be performed for error concealment are described in this Section. The tasks are:

- Task #1: Digitize frame and transfer from the Corona board to the Detroit board
- Task #2: Block Loss
- Task #3: Error Concealment
- Task #4: Frame transfer from Detroit board to Corona board for display

Flowcharts of the error concealment algorithm are shown in Appendix A.1.

Table 5.1
CCIR-601 (YUV 4:1:1, 720×480 pixels) data rates

Frame Size	Data Rate
Y: 720×480	345,600 Bytes/frame
U: 360×240	86,400 Bytes/frame
V: 360×240	86,400 Bytes/frame
Total	518,400 Bytes/frame
At 30 frames/sec	
Input only	15,552,000 Bytes/sec (124,416,000 bits/sec)
Input and Output	31,104,000 Bytes/sec (248,832,000 bits/sec)

The video source can be a video camera, videotape, or a baseband cable TV feed. The input is digitized using a Matrox Corona board. The video format used is CCIR-601 (YUV 4:1:1, 720×480 pixels) at 30 frames/sec. We transfer the frames from the digitizer into the on-board memory via the PCI Bus. Since error concealment does not modify the frame size or the number of bytes per pixel, the data rate out of the Detroit board is the same as the input data rate (see Table 2). The Detroit board is capable of supporting up to 80 MB/s. The bandwidth required for the video I/O is shown in Table 5.1. The transfer is performed by the digitizer or using the *'C6000* DMA. The frames are stored in external memory. The *'C6000* CPU does not perform any computation.

Task #2: Block Loss

As shown in Figure A.2 in Appendix A.1, Task #2 is divided in two parts:

- Task #2.1: Determine the location of block to be damaged.

- Task #2.2: Set pixels in block to zero.

To simulate ATM cell loss, we randomly drop 8×8 blocks[1] of pixels from a frame (5400 blocks/frame) at a rate of 5%, which represents eliminating 270 blocks/frame. In previous experiments [10], we have shown that the near-optimal technique does not perform well if contiguous blocks are lost. Therefore, we impose the restriction that no adjacent blocks are dropped. In Task #2.1, the *'C6201* determines which blocks are dropped using a pseudo-random selection. The complexity of this task is low when compared to Task #3 Error concealment. Then, in Task #2.2, the *'C6201* replaced the pixel values with zero to indicate the locations of dropped blocks in external memory.

Task #3: Error concealment

Task #3 is divided in five tasks (see Figure A.3 in Appendix A.1):

- Task #3.1: Determine the location of lost blocks.
- Task #3.2: Read block plus neighbor pixels.
- Task #3.3: Initialize pixels in lost block.
- Task #3.4: Median filter block
- Task #3.5: Write block to external memory.

Since we implement the technique in real-time, we investigate if there is enough time between frames to do the processing. The processing requirements of Tasks #3.1, #3.2 and #3.5 are minimal when compared to Tasks #3.3 and #3.4.

Each missing block to be concealed needs to be processed separately, first by initializing the pixels in the block (Task #3.3), and then by filtering (Task #3.4). Both stages require a median filter; thus we use this operation twice on each pixel in the block. In a YUV 4:1:1 frame, each block1 has two components: luminance (Y) and chrominance (U or V). Hence, the number of calls to the median filtering

[1]Note: An 8×8 block of pixels includes an 8×8 block for the Y component and a 4×4 block for each of the U and V components.

routine per 8×8 block is: 2× [(8×8 calls/block for luminance component) + (2×4×4 calls/block for chrominance components)] = 2× [96 calls/block] = 192 calls/block.

In general, median filtering a 3×3 block of pixels requires the sorting of 8 pixels; then adding elements ranked 4th and 5th, dividing the sum by two, and assigning the value to the pixel in the center of the block, as shown in Figure 5.3. Texas Instruments (TI) has released a benchmark for a routine for the *'C6201* that finds the maximum value of a vector (see Appendix A.2). This *max* routine has been hand-optimized and assumes that the size of the vector is a multiple of 6. Initially, we modified this routine to accept a vector of size 8, but we replace it by our own hand-optimized routine to account for the particular requirements of our application.

The median filter could be implemented by calling the *max* function 5 times, keeping track of the 4th and 5th values returned. This routine requires $n/2 + 13$ cycles to complete. Assuming that it takes the same number of cycles for an array of size 8, each call to *max* would require $8/2 + 13 = 17$ cycles. We can add 8 extra cycles per call to allow for bookkeeping. Calling *max* five times yields 125 cycles for the median filter to complete. Again, we can add 25 extra cycles to take into consideration that data has to be put into a vector. Therefore, the median filter is expected to take about 150 cycles.

As described later in this Section, in our work we have implemented a C-callable median filter routine in assembly language, which has been benchmarked in a *'C6000* test and evaluation board (TEB). This routine took 126 cycles to complete, in hand-optimized code. We can determine the processing requirements using this number. Thus, to process each block, it would require (126 cycles/call)×(192 calls/block) = 25,192 cycles/block. Since there are 270 blocks to be processed, the number of cycles required to process a frame is approximately (25,192 cycles/block)×(270 blocks/frame) = 6,531,840 cycles/frame. The *'C6201* has a 5 nsec clock cycle, thus it takes approximately 33 msec to process each frame, allowing us to process 30 frames/sec.

5.1.4 Summary of processing, memory, and bandwidth requirements

From the previous Sections, we summarize the processing, memory and bandwidth requirements for our work in Table 5.2. The columns in Table 5.2 are:

- CPU Load: Number of cycles that the *'C6000* uses to perform a task.
- Program memory: Number of bytes in internal program memory occupied by the code used to perform a task.
- Data memory: Number of bytes in external memory (internal memory for Tasks #3.3 and #3.4) that need to be accessed during error concealment of one frame.
- Bandwidth: Amount of data moved to and from external memory (for Tasks #1 and #4) and internal memory (for Tasks #2 and #3).
- Data reads: Number of cycles used to read data from memory.
- Data writes: Number of cycles used to write data to memory.

Data shown for Tasks #1 and #4 was taken from Table 5.1. For Tasks #2 and #3, entries denoted as "low" represent numbers that are small when compared to other entries in the same column. These numbers may be considered negligible, but not zero. Following is a description on how the information presented in Table 5.2 was obtained.

- CPU load:
  - Tasks #2.3 and #2.4: (126 cycles/call to "median") × (96 calls/block) × (270 blocks/frame) × (30 frames/sec) = 97,977,600 cycles/sec.
- Program memory:
  - In our work, described later in this section, we have determined that the "median" routine, which is the main task in our work, uses approximately 500 assembly instructions. Therefore, for Tasks #3.3 and #3.4 we expect to use all the internal program memory available.

- Data memory:
  - Tasks #2.2, #3.4 and #3.5: (8×8 luminance components/block + 2×4×4 chrominance components/block) = 96 luminance and chrominance components/block. (96 components/block) × (1 byte/component) × (270 blocks/frame) = 25,920 bytes for one frame.
  - Tasks #3.2 and #3.3: Similar than Task #2.2, except that we need to account for the reference pixels too: (36 luminance components) + (2×20 chrominance components) = 76 reference components. (96 components/block + 76 reference components/block) × (1 byte/component) × (270 blocks/frame) = 46,440 bytes for one frame.
- Bandwidth:
  - Task #2.2: Using values obtained for data memory, (25,920 bytes/frame)× (30 frames/sec) = 777,600 bytes/sec, transferred from internal to external memory using DMA. All the values are zeros.
  - Task #3.2: Only reference pixels need to be read from memory. Thus, (76 reference components/block) × (1 bytes/component) × (270 blocks/frame) × (30 frames/sec) = 615,600 bytes/sec, transferred from external to internal memory using DMA.
  - Task #3.3: Same number as Task #3.2, read from internal memory by the CPU.
  - Task #3.4: (9 pixels/call to "median") × (96 calls/block) × (270 blocks/frame) × (30 frames/sec) = 6,998,400 bytes/sec read from internal memory by the CPU.
  - Task #3.5: Same number as Task #2.2, transferred from internal to external memory using DMA.
- Data reads (using values obtained for Bandwidth):

  - Task #3.2: Number of bytes read from external memory is: 615,600 bytes/sec. The actual number of cycles depends on external memory access time.
  - Task #3.3: (615,600 bytes/sec) × (1 cycle/4 bytes) = 153,900 cycles/sec, reading from internal memory by the CPU.
  - Task #3.4: (6,998,400 bytes/sec) × (1 cycle/4 bytes) = 1,749,600 cycles/sec, reading from internal memory by the CPU.

- Data writes (using values obtained for Bandwidth):
  - Task #2.2: The number of bytes written to external memory is: 777,600 bytes/sec. The actual number of cycles depends on external memory access time.
  - Task #3.3 and #3.4: Number of bytes written is the same as in Task #2.2, writing to internal memory. (777,600 bytes/sec)×(1 cycle/4 bytes) = 194,400 cycles/sec.
  - Task #3.5: Same as Task #2.2. The number of bytes written to external memory is: 777,600 bytes/sec. The actual number of cycles depends on external memory access time.

Our experiments have focused on the implementation of a median routine, which is the kernel of our work. This routine finds the median of eight integer values. A possible implementation in C is shown below:

```
short median(short a[])          /* Finds the median of 8 numbers */
{
  short m, n, min, med;
  for (m=1; m<8; ++m) {          /* Completely sort the array     */
    min = a[m];
    n   = m-1;
```

Table 5.2
Summary of requirements for error concealment system. "Mc/s" stands for Mcycles/sec.

Task #	CPU Load (Mc/s)	Program Memory (Bytes)	Data Memory (Bytes)	Band-width (MB/s)	Data Reads (Mc/s)	Data Writes (Mc/s)
1. Transfer frame from digitizer	0	0	518,400	15.552	0	0
2. Block loss						
2.1 Location	Low	Low	Low	0	0	0
2.2 Write	0	Low	29,920	0.777	0	Low
3. Error Concealment						
3.1 Location	Low	Low	Low	0	0	0
3.2 Read	Low	Low	46,440	0.615	Low	0
3.3 Initialization	97.977	512,000	46,440	0.615	0.153	0.194
3.4 Filtering	97.977	512,000	25,920	6.998	1.749	0.194
3.5 Write	Low	Low	25,920	1.036	0	Low
4. Transfer frame to display	0	0	518,400	15.552	0	0

```
    while (n>=0 && a[n]>min) {
      a[n+1] = a[n];
      n--;
    }
    a[n+1] = min;
  }                              /* Array is sorted               */
  med = a[3]+a[4];               /* Calculate the median          */
  med = med / 2;
  return(med);                   /* Return value                  */
}
```

As can be seen in the above implementation, it is not necessary to completely sort the array, since to calculate the median, only the largest (or smallest) five elements need to be known. Thus, a faster implementation would be:

```
short median(short a[])          /* Finds the median of 8 numbers */
{
  short m, n, med, index, fourth, fifth;

  for (m=1; m<=5; ++m) {
      index = maxindex(a);       /* Find the index of max value   */
                                 /* by going through entire array */
      if (m==4) fourth = a[index]; /* Save to calculate median    */
      if (m==5) fifth  = a[index];
      a[index] = 0;              /* Discard current max           */
  }
  med = fourth + fifth;          /* Calculate the median          */
  med = med/2;
  return(med);                   /* Return value                  */
}
```

This is the approach in the median routine that we used in our work. The above C program was benchmarked using the debugger/emulator with the *'C6201* test and evaluation board (TEB). The median completed in 2450 cycles (1055 cycles using compiler optimization). Earlier in this Section, we set our target number of cycles for the median routine to be about 150. It is clear that a more aggressive approach would need to be used.

We chose to use an assembly language implementation of the "max_index" routine in the above program. The "max_index" routine was obtained from the Texas Instruments website (http://www.ti.com/). This routine assumes that the number of elements in the array is a multiple of three, so we initially inserted a ninth element with a value of 0 to test the performance of this approach. The median completed in 652 cycles (407 cycles using compiler optimization), which is still above our target.

The next step was to implement the median routine, shown above, in assembly language. The "max_index" routine provided by TI works on an array. These values are loaded from memory every time the routine is executed. Since we know beforehand that the maximum needs to be found five times, we consider that it is better to have the data in registers to minimize memory accesses. Therefore, we wrote our own routine, which is referred as "max_and_zero" to reflect the fact that after the maximum is found, it needs to be cleared before the routine is called again. Our "median" routine calls "max_and_zero" five times, as the C program shown above. The code of version 1.0 of our "median" routine (with no optimization) is shown in Appendix A.2. This is a C-callable routine. The median completed in 220 cycles.

Version 1.0 of our code was hand-optimized. Instructions were rearranged to fill delay slots, and a new approach to clear the register containing the maximum value was used. This approach was used after we determined that this routine was the most time-consuming task.

The code of version 2.0 of "median" is shown in Appendix A.3. It executed in 126 cycles in our *'C6000* TEB with rev 1.0 silicon. This cycle count is better than our estimate of 150 cycles.

5.1.5 Implementation issues

We envision an implementation scenario where error concealment is a module of an MPEG decoder. Our spatial technique would be used on I frames with isolated damaged macroblocks, using information from the current frame only. When adjacent macroblocks are damaged, a modified temporal technique, such as the one described in [11], can be used. In P frames and B frames, only our temporal technique is be used. Reference frames are necessary for the temporal technique, therefore the error concealment module must have access to the frame buffer used by the motion compensation module of the decoder.

In implementing the error concealment on the *'C6201*, several issues need to be addressed. To obtain the highest performance with the spatial technique, the neighboring pixels need to be transferred into internal data memory. Both initialization and filtering stages are data intensive, thus the high penalty of accessing external memory locations must be avoided. The data and computational requirements for the temporal technique are lower than the spatial technique, because only the neighboring undamaged motion vectors are needed. Another issue is the limited amount of on-chip memory available on the *'C6201* (64 Kbytes). A frame must be partitioned into sections, and each section transferred into internal data memory as needed. Direct memory access is used for all block data transfers to increase efficiency. Data transfers must occur while computation is being performed to maximize overlapping.

5.1.6 Experimental results

Currently, our system can process 30 frames/second. We use CIF size (320×240 pixels) grayscale images. YUV images can be processed at a rate of 10 frames/seconds. The visual quality of the results is shown in Figure 5.5.

In our experiments, we have determined that a single 200MHz *'C6201* is needed for the implementation of the spatial technique. To reconstruct a missing block, each pixel is median filtered twice. Because the computational requirements of our temporal technique are lower, a single *'C6201* would be required in our implementation scenario. The operations are based on integer values. This is also the case for other

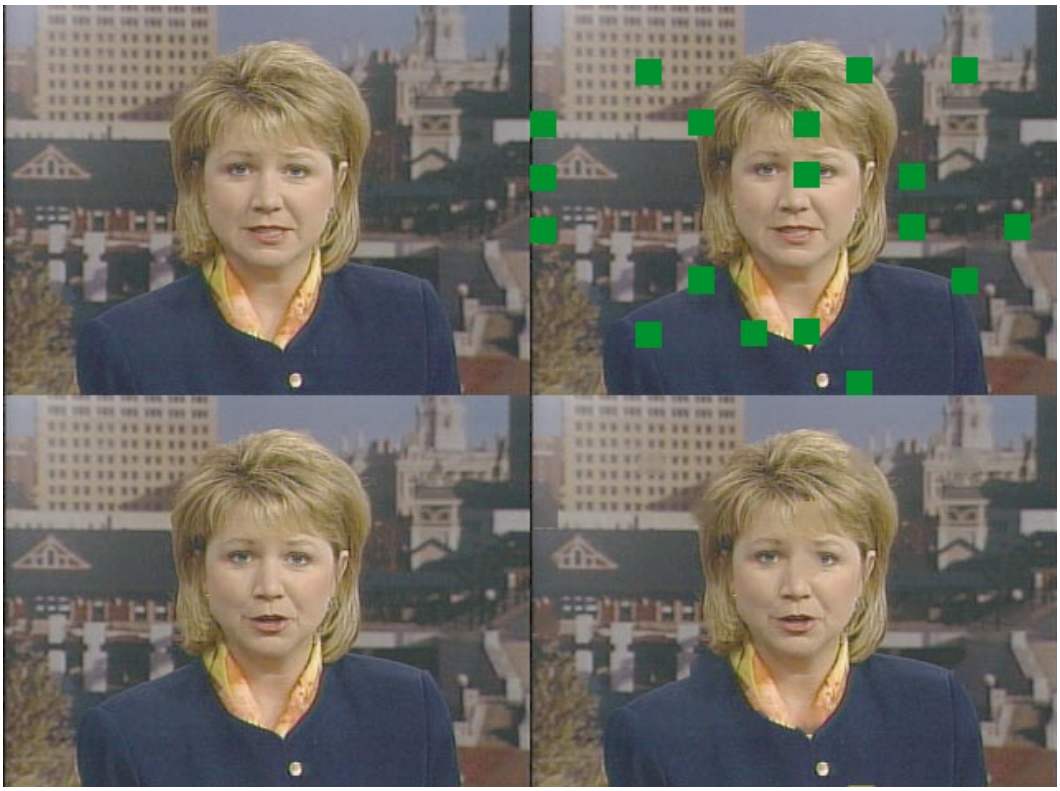


Fig. 5.5. Error concealment: Top left: Original frame. Top right: Frame with missing blocks due to channel errors. Bottom left: Original frame. Bottom right: Frame after the errors have been concealed using our technique. The spatial location of the errors is the same for both frames on the right.

tasks within an MPEG decoder, such as motion compensation [179]. Therefore, there is an advantage in using a fixed-point DSP because of the low latency of the instructions. All arithmetic and logic operations we use in the *'C6201* are single-cycle. Thus, the combination of a fast clock cycle and lower cost makes the use of the *'C6201* very attractive for this class of applications.

5.1.7 Temporal error concealment

Temporal error concealment techniques that use information from previous frames to recover errors in a current frame can also be implemented on the *'C6201*. The lost information can include motion vectors and macroblocks. A simple approach is to copy the macroblock at the same spatial location of the missing macroblock in the previous frame [180]. This is useful in sequences with simple scenes. To reduce distortion in sequences with complex scenes, more elaborate techniques need to be used. In these approaches, the lost data is recovered by estimating the missing motion vector and/or macroblock. In [10], an estimate of the missing motion vector is obtained by searching for the vector that maximizes the *a posteriori* distribution of the boundary pixels of the lost macroblock given its neighbors. Another approach is to estimate the missing motion vector by obtaining the median [11] or average [181] of the motion vectors of neighboring undamaged motion vectors.

In [11], it is shown that using the median of the neighboring undamaged motion vectors compares favorably to averaging. This is the technique chosen for our implementation on the *'C6201*. The reference frame and neighboring undamaged motion vectors are required for this implementation. Each dimension of the motion vectors from neighboring macroblocks is considered separately. The resulting motion vector is used to replace the motion vector in the damaged macroblock, as shown in Figure 5.6.

5.2 Discrete Wavelet Transform in a Fixed-Point DSP

The discrete wavelet transform (DWT) [75] is used in many image and video processing applications, such as compression, image analysis, and restoration. The JPEG2000 image compression standard [89, 90] is based on the wavelet transform, and is being used in applications such as digital still cameras and wireless communications.

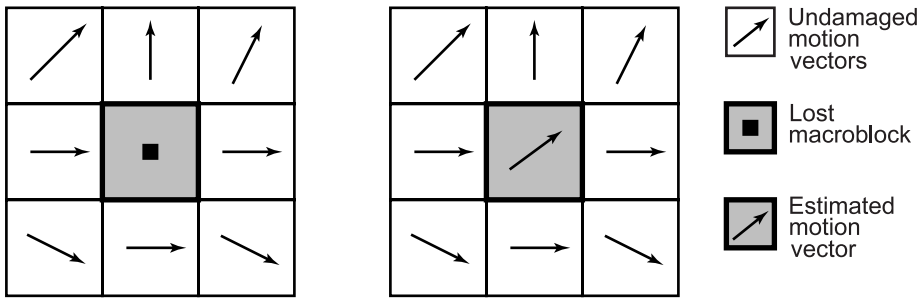


Fig. 5.6. Temporal error concealment. Left: Neighboring motion vectors. Right: Estimated motion vector.

5.2.1 Algorithm implemented

For implementing the DWT, we use the Daubechies (9,7) wavelet filter pair [75] which is a separable, perfect reconstruction filter. It has been shown that of this filter performs well for image and video processing applications [182]. We implemented the DWT as described in Section 2.4, using whole-sample symmetric extension for the boundaries of the images. Other implementations of the DWT have been proposed to reduce the computational complexity of the algorithm [183].

The coefficients of the (9,7) wavelet filter are real numbers, therefore one problem that needs to be addressed when implementing the DWT is that sufficient precision is used in order to obtain a perfect reconstruction. If not enough precision is used, the difference between the original and reconstructed images can be between 1 to 20 pixel levels. For some applications, such as image compression, this may not be a significant problem because the effect of lack of precision is similar to the effect of quantization. Furthermore, in our experiments, we perform both decomposition and reconstruction on the *'C6201*. In some scenarios, such as a digital still camera, the reconstruction stage would be performed in a different platform, most likely a PC, where a floating-point implementation would be used. Therefore, lack of precision would be less of an issue.

5.2.2 PC code implementation

In fixed-point implementations, the most important issue is dynamic range. In general, the filter coefficients are real numbers and therefore the representation must allow for the multiplication or addition of coefficients at the highest practical precision, without overflow or underflow. A function was created to estimate the appropriate number of integer and fractional bits needed for the desired precision.

A fixed-point version of the floating-point code for the wavelet transform was written for *'C6201*. A substantial number of changes are required, including using memory allocation, integer arithmetic only, avoiding overflow and underflow, and careful planning of the data flow to and from memory. The code is written in such a way that the *'C6201* compiler is able to exploit the parallelism of the application and produce efficient code. The size of a word in the *'C6201* is 16 bits. The representation of the filter coefficients used is 16 bits with 15 fractional bits and one sign bit. Wavelet coefficients is 16 bits with 9 integer bits, 6 fractional bits and one sign bit. This allows to perform at least 4 levels of decomposition of the (9,7) filter pair without causing overflow/underflow. To make maximum use of the 9 integer bits of the wavelet coefficients, the pixel values of an image are multiplied by 64 (left-shifted by 6 bits) and the mean of the image is subtracted before the filtering operations to maximize the dynamic range. Once the image is reconstructed, the mean is added back and the pixels are right-shifted by 6 bits. During filtering, after the multiplication of a wavelet coefficient and a filter coefficient is done, the 32-bit product is temporarily stored in an accumulator. To minimize the loss of dynamic range, the result is rounded before conversion to a 16-bit representation.

5.2.3 DSP code implementation

The DSP implementation of the wavelet code is based on the fixed-point PC version. In the *'C6201*, we ran the fixed-point version of the DWT in both the simulator and the Detroit board, obtaining near-perfect reconstruction for grayscale images using up to 4 levels of decomposition. The maximum pixel difference was 1 gray level.

Table 5.3
Timing results for wavelet decomposition and reconstruction on the *'C6201* assuming a clock rate of 200 MHz.

Image Size (width×height)	Decomposition (cycles / seconds)	Reconstruction (cycles / seconds)	Total (cycles / seconds)
176×144	4.00 / 0.02	6.76 / 0.03	11.90 / 0.06
256×256	10.46 / 0.05	17.41 / 0.09	29.45 / 0.15
512×512	40.77 / 0.20	68.15 / 0.34	114.88 / 0.57
1024×1024	163.08 / 0.81	265.10 / 1.32	482.34 / 2.42

Timing results, using the *'C6201* simulator, for a 4 level decomposition and reconstruction at a clock speed of 200 MHz are shown in Table 5.3 for images of various sizes.

We also implemented the wavelet decomposition and reconstruction on the Detroit system. The system block diagram is the same as the one shown in 5.4. We capture a frame using the Matrox Corona digitizer. The frame is transferred to the Detroit board, where decomposition and reconstruction is performed. After reconstruction, the frame is transferred back to the Corona for display. An original frame, its wavelet reconstruction, and the difference frame between the original and the reconstruction are shown in Figures 5.7, 5.8, and 5.9, respectively. Four levels of decomposition were used in this example.

Fig. 5.7. Frame before wavelet decomposition.

Fig. 5.8. Frame after wavelet decomposition and reconstruction on the *'C6201.*

Fig. 5.9. Difference image between the original frame and the frame after wavelet decomposition and reconstruction. The difference is only one gray level.

6. CONCLUSIONS AND FUTURE RESEARCH

In this thesis, we have investigated the performance of *SAMCoW* at low data rates (64 kbps and lower), and identified its limitations. These include: small image size, the characteristics of the PEFs in *SAMCoW*, the blurring effect of the wavelet transform, and the use of basic video coding techniques, such as restricted motion vectors, integer-pixel accurate motion estimation, the use of I and P frames only, and static bit allocation.

We have investigated several extensions to *SAMCoW* designed to overcome these limitations. These extensions are known as *SAMCoW+*.

First, we investigated the use of advanced video coding techniques. The goal is to obtain a more accurate prediction, therefore improving the coding performance of *CEZW* on predictive error frames. These techniques are [6]: Half-pixel accurate motion estimation and compensation, unrestricted motion vectors, bidirectionally predictive-coded (B) frames, and dynamic bit allocation.

Second, we used preprocessing and postprocessing techniques to improve the coding performance of *CEZW* on predictive error frames (PEF) [7, 8]. Our goal is to allocate bits to areas of the PEF where the predictive error is large. Preprocessing and postprocessing techniques are attractive because they do not add overhead to the bit stream. The trade-off is introducing more complexity to the algorithm.

Third, we developed a framework for encoding PEFs based on rate-distortion optimization [9]. The goal is to obtain the best operating point (in the rate-distortion sense) for each spatial orientation tree (SOT) in the wavelet decomposition. This framework is incorporated into an extension of *CEZW* known as *CEZW+*. *CEZW+* is used for I frames and PEFs in *SAMCoW+*.

CEZW+

An interesting topic for future research is a study of the performance of *CEZW+* on "natural" images. *CEZW+* was developed for PEFs, but it is also used for I frames in *SAMCoW+*. In our rate-distortion analysis, we use the mean-squared error (MSE). A topic for future research is the study of other distortion metrics. Minimizing the MSE of the SOTs does not guarantee a better reconstruction of the pixel values of the image. Thus, a metric that relates the coefficients in the decomposition and the reconstructed pixel values would yield improved results.

Improvements to *SAMCoW+*

Several improvements can be made to *SAMCoW+*.

- Reducing the computational complexity of motion estimation. Currently, we use full-search motion estimation. Although full-search yields the best possible match, it does increase the computational complexity substantially. Investigating the use of less complex motion estimation techniques would allow the use of larger search areas to improve the motion prediction, thus improving the performance of *SAMCoW+*. In addition, encoding the resulting motion field may not yield the most compact bit stream. Several techniques based on rate-distortion analysis of motion estimation have been proposed [175], and their use in *SAMCoW+* could increase its performance.

- Adding other scalability modes to *SAMCoW+* would significantly increase the range of application where *SAMCoW+* can be used. A sucessful scalable video compression technique is one where a variety of modes can be selected and combined to better serve different applications. B frames introduce a simple form of temporal scalability. Other modes, such as SNR and spatial scalability, are also important and can be investigated.

- Maintaining the embedded properties of *SAMCoW+* is important, but may not be the best way to accomplish the objective of having multiple scalability

modes. Therefore, investigating the advantages and disadvantages of an embedded versus a layered scalable compression technique is an important topic for future research.

- The use of other advanced video coding techniques can also improve the performance of *SAMCoW+* in a variety of applications. Techniques such as reference picture selection for video streaming applications, and the use of four motion vectors per macroblock to obtain a more accurate motion prediction, have been demonstrated in video compression standards, and can be adapted to *SAMCoW+*.

- A characteristic of the PEFs, blockiness due to block-based motion estimation, needs to be studied in more detail. Despite the use of overlapped block motion compensation (OBMC) at the macroblock level, blocking artifacts still can be observed in the PEFs. A possible solution to this problem is the use of of four motion vectors per macroblock, to obtain a more accurate motion prediction, along with OBMC at the block level. Alternatively, using postprocessing techniques that remove blocking artifacts can also be investigated.

- The use of error correction and concealment techniques in wavelet-based image and video compression techniques is also an important area for future research. The use of these techniques would greatly decrease the sensitivity of *SAMCoW+* to noise.

- The use of more sophisticated bit allocation schemes and rate control can improve the performance of *SAMCoW+* in scenarios where traffic in the network forces the system to widely vary the target data rate. This is because when using multiple scalability modes the number of variables that must be taken into consideration to maintain the quality of the sequence increases.

The use of *SAMCoW+* for video streaming over packet and wireless networks

Another important topic for future research is the use of *SAMCoW+* for video streaming on packet networks such as the Internet, an application for which the use of a rate scalable codec is highly desirable. Several aspects of *SAMCoW+* need to be studied before its successful use in video streaming applications.

- Investigating the performance of *CEZW+* for still image transmission over packet and wireless networks is a fundamental step towards understanding the issues that affect wavelet-based image and video compression techniques for this class of applications.

- Wavelet-based rate scalable video compression techniques such as *SAMCoW* are very susceptible to transmission errors. This is due in part to the use of arithmetic coding to reduce the entropy of the bit stream. However, wavelet-based algorithms that use SOTs as their basic coding unit, such as *CEZW+* in *SAMCoW+*, can loose synchronization if a transmission error occur. Therefore, introducing error resilience techniques for *SAMCoW+* is a very important topic for future research.

- The packetization scheme used in *SAMCoW+* needs to be designed having in mind the type of network and transport protocols that will be used for delivery. This will facilitate the integration of *SAMCoW+* into a multimedia delivery system.

Error resilience

Many techniques have been used for error resilience, such as the use of resynchronization markers and reversible variable length codes (RVLC). Resynchronization markers are symbols inserted in the bit stream that help the decoder recover from transmission errors. The use of these techniques in *SAMCoW+* is an interesting area for future research.

Important problems exist in the areas of video delivery over packet and wireless networks, and error resilience for video transmission. The challenges in these areas reside in the fact that the design of new video compression schemes must take into account factors not commonly associated with video compression, such as multiple access networks, and packetization schemes.

APPENDIX

A.1 Flowcharts of the error concealment algorithm

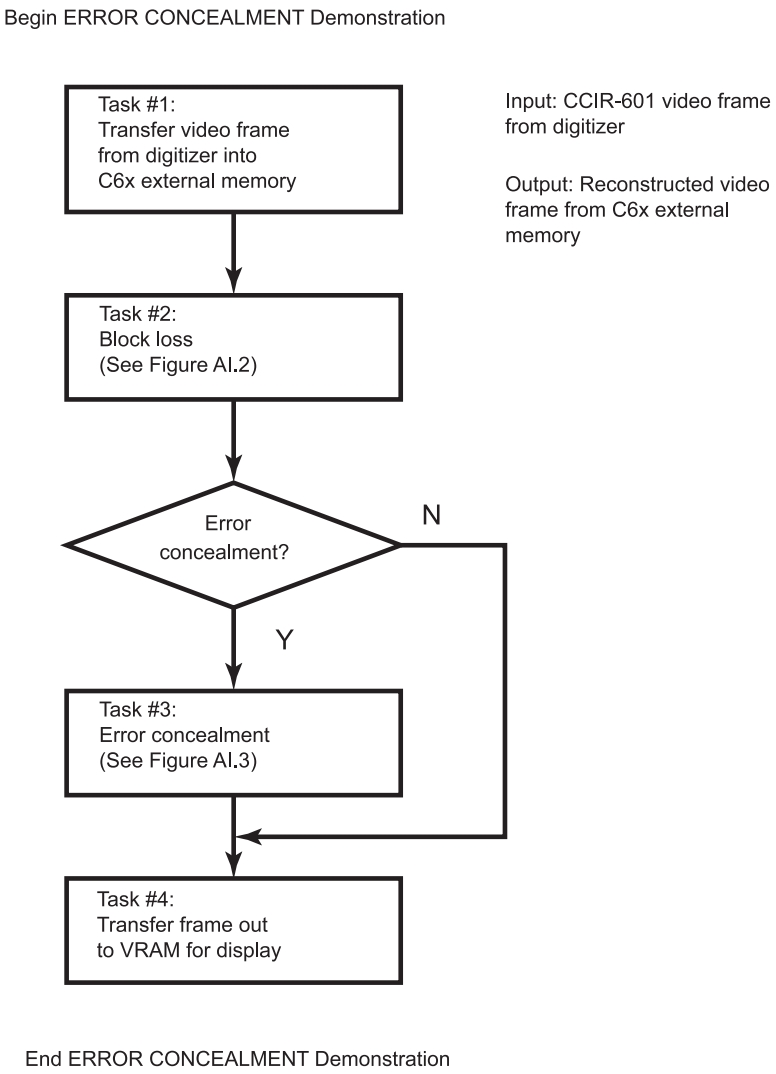


Fig. A.1. Flowchart of the error concealment algorithm.

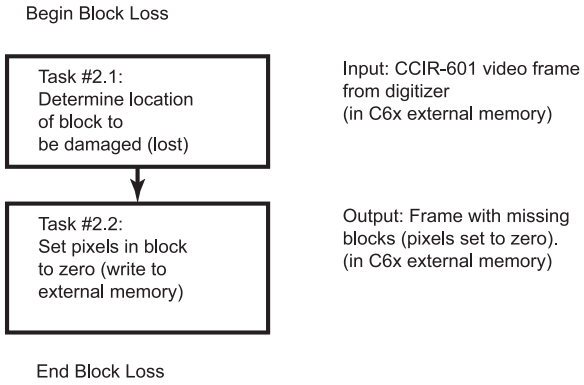


Fig. A.2. Flowchart of task #2: Block loss.

A.2 Assembly code for median filtering (no optimization)

```
;=================================================================
;
; ROUTINE:      median   Version 1  (Optimized)
;
; USAGE:        This is a C callable and can be called as:
;
;               short median(short val1, short val2,
;                            short val3, short val4,
;                            short val5, short val6,
;                            short val7, short val8)
;
;               The input values are in registers a4, b4, a6, b6,
;               a8, b8, a10, and b10 according to the calling
;               convention of the compiler.
;
; DESCRIPTION:  Finds the median value of 8 data points passed
;               and returns the median value.
;               Uses subroutine "max_and_zero" (found below).
```

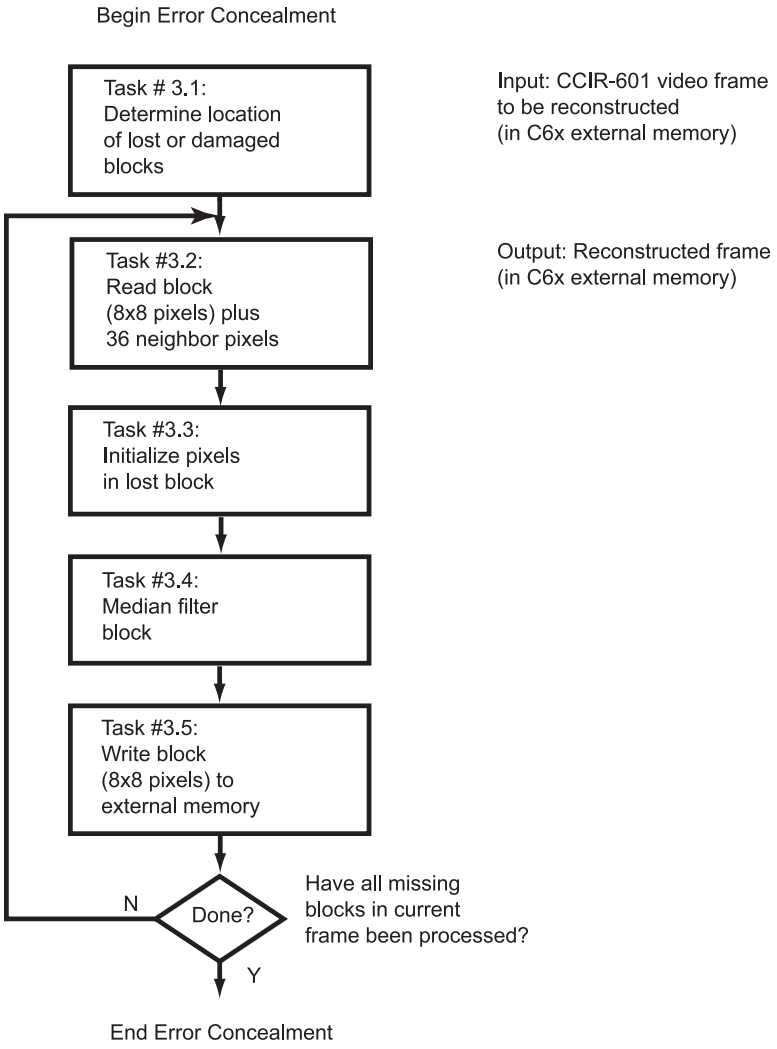


Fig. A.3. Flowchart of task #3: Error concealment.

```
;               It calls "max_and_zero" five times, saving fourth
;               and fifth largest values, which are used
;               to calculate median.
;
;               Some optimization performed.
;
; Eduardo Asbun      Purdue University         March 16, 1998
;================================================================
```

```
          .title         "Routine:    median    March 16, 1998"
          .def        _median
          .sect          "init"
_median:
          b       .s1     max_and_zero
          mvk     .s2     median_2,b5   ; save return address
          mvkh    .s2     median_2,b5
          mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
                                        ;   to max value
 ||       mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
                                        ;   to max value
          cmpgt   .L1     a4,a9,a1      ; a4 ? max_a
 ||       cmpgt   .L2     b4,b9,b1      ; b4 ? max_b
   [ a1] mv      .d1     a4,a9         ; Save new max_a
 ||[ b1] mv      .d2     b4,b9         ; Save new max_b
 ||[ a1] mvk     .s1     0x4,a7        ; Point to new max_a
 ||[ b1] mvk     .s2     0x4,b7        ; Point to new max_b

median_2:
          b       .s1     max_and_zero
          mvk     .s2     median_3,b5   ; save return address
          mvkh    .s2     median_3,b5
          mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
                                        ;   to max value
 ||       mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
```

```
                                        ;to max value
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x4,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x4,b7         ; Point to new max_b

median_3:
         b       .s1     max_and_zero
         mvk     .s2     median_4,b5    ; save return address
         mvkh    .s2     median_4,b5
         mvk     .s1     0x0,a9         ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
                                        ;   to max value
 ||      mvk     .s2     0x0,b9         ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
                                        ;   to max value
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x4,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x4,b7         ; Point to new max_b

median_4:
         b       .s1     max_and_zero
         mvk     .s2     median_5,b5    ; save return address
         mvkh    .s2     median_5,b5
```

```
         mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                       ; a10 is pointer
                                       ;   to max value
 ||      mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                       ; b10 is pointer
                                       ;   to max value
         cmpgt   .L1     a4,a9,a1      ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1      ; b4 ? max_b
   [ a1] mv      .d1     a4,a9         ; Save new max_a
 ||[ b1] mv      .d2     b4,b9         ; Save new max_b
 ||[ a1] mvk     .s1     0x4,a7        ; Point to new max_a
 ||[ b1] mvk     .s2     0x4,b7        ; Point to new max_b

median_5:

         b       .s1     max_and_zero
 ||      mv      .d1     a5,a3         ; save fourth largest
                                       ;   value (Moved!!!)
         mvk     .s2     median_6,b5   ; save return value
         mvkh    .s2     median_6,b5
         mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                       ; a10 is pointer
                                       ;   to max value
 ||      mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                       ; b10 is pointer
                                       ;   to max value
         cmpgt   .L1     a4,a9,a1      ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1      ; b4 ? max_b
   [ a1] mv      .d1     a4,a9         ; Save new max_a
```

```
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x4,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x4,b7         ; Point to new max_b

median_6:                               ; fifth largest value in a4

median_done:
         b       .s2    b3             ; Return to C calling routine
         add     .s1    a3,a5,a2       ; add fourth and fifth
                                       ;   largest values (Moved!!!)
         shr     .s1    a2,1,a4        ; calculate median
         nop            3
;=============================================================
;=============================================================
;  Routine:  "max_and_zero"
;
;    Finds the maximum value ("global maximum")
;      of 2 sets of 4 points.
;
;    Values are in these registers:
;      set "a":   a4, a6, a8, a10
;      set "b":   b4, b6, b8, b10
;
;    Sets register containing global maximum value to zero.
;    Returns maximum in a4.
;
;  Registers used:
;    a1,  b1             Used for conditionals
;    a5                  Return value (global maximum)
```

```
;    b5                  Return address
;    a4,  a6, a8, a10    Values for set "a"
;    b4,  b6, b8, b10    Values for set "b"
;    a7,  b7             pointers to maximum value in sets
;                          "a" and "b"
;    a9,  b9             max_a, max_b
;=====================================================================
        .title          "Routine:  max_and_zero    Mar 16, 1998"
max_and_zero:

;=====================================================================
; Part I. Find maximum of each set ("a" and "b")
;=====================================================================
find_max:                                ; Moved some instructions
                                         ;   to "median" !!!
         cmpgt   .L1    a6,a9,a1         ; a6 ? max_a
 ||      cmpgt   .L2    b6,b9,b1         ; b6 ? max_b
   [ a1] mv      .d1    a6,a9            ; Save new max_a
 ||[ b1] mv      .d2    b6,b9            ; Save new max_b
 ||[ a1] mvk     .s1    0x6,a7           ; Point to new max_a
 ||[ b1] mvk     .s2    0x6,b7           ; Point to new max_b

         cmpgt   .L1    a8,a9,a1         ; a8 ? max_a
 ||      cmpgt   .L2    b8,b9,b1         ; b8 ? max_b
   [ a1] mv      .d1    a8,a9            ; Save new max_a
 ||[ b1] mv      .d2    b8,b9            ; Save new max_b
 ||[ a1] mvk     .s1    0x8,a7           ; Point to new max_a
 ||[ b1] mvk     .s2    0x8,b7           ; Point to new max_b
```

```
         cmpgt   .L1    a10,a9,a1      ; a10 ? max_a
 ||      cmpgt   .L2    b10,b9,b1      ; b10 ? max_b
   [ a1] mv      .d1    a10,a9         ; Save new max_a
 ||[ b1] mv      .d2    b10,b9         ; Save new max_b
 ||[ a1] mvk     .s1    10,a7          ; Point to new max_a
 ||[ b1] mvk     .s2    10,b7          ; Point to new max_b

;=================================================================
; Part II. Zero out global maximum
;=================================================================
         shr     .s1    a2,1,a4        ; calculate median
zero:                                  ; Zero out register that
                                       ;   contains max
         cmpgt   .L1x   a9,b9,a1       ; Is "max_a" or "max_b"
                                       ;   the global max?
   [!a1] b       .s1    max_in_b       ; "max_b" is global max
         cmplt   .L1    a7,0x8,a1      ; is max in (a4, a6) or
                                       ;    (a8, a10)? (Moved!!!)
 ||      mv      .s1    a9,a5          ; Return value is
                                       ;   global max
         nop             4             ; delay slots
max_in_a:                              ; Global max in "a" set
                                       ; Some code moved
                                       ;   from here!!!
   [!a1] b       .s1     max_a810      ; Max is in (a8, a10)
         nop             4             ;   Delay slots
;;;      b       .s1     done          ;   Done
         b       .s2     b5            ;   Return to calling routine
                                       ;     (median)  (Moved!!!)
```

```
max_a46:                                ; Max is in (a4, a6)
         cmplt   .L1     a7,0x6,a1      ;   is max  in a4 or a6 ?
   [ a1] zero    .s1     a4             ;   max was in a4
 ||[!a1] zero    .L1     a6             ;   max was in a6
         nop             3              ;   remaining delay slots
max_a810:                               ; Max is in (a8, a10)
         cmplt   .L1     a7,10,a1       ;   is max  in a8 or a10 ?
   [ a1] zero    .s1     a8             ;   max was in a8
 ||[!a1] zero    .L1     a10            ;   max was in a10
         nop             3              ;   remaining delay slots
max_in_b:                               ; Global max in "b" set
         cmplt   .L2     b7,0x8,b1      ; is max in (b4, b6)
                                        ;    or (b8, b10)?
 ||      mv      .s1x    b9,a5          ; Return value is global max

   [!b1] b       .s2     max_b810       ; Max is in (b8, b10)
         nop             4              ;   Delay slots
;;;      b       .s1     done           ;   Done
         b       .s2     b5             ;   Return to calling routine
                                        ;     (median)  (Moved!!!)

max_b46:                                ; Max is in (b4, b6)
         cmplt   .L2     b7,0x6,b1      ;   is max  in b4 or b6 ?
   [ b1] zero    .s2     b4             ;   max was in b4
 ||[!b1] zero    .L2     b6             ;   max was in b6
         nop             3              ;   remaining delay slots

max_b810:                               ; Max is in (b8, b10)
         cmplt   .L2     b7,10,b1       ;   is max  in b8 or b10 ?
```

```
   [ b1] zero    .s2     b8            ;   max was in b8
 ||[!b1] zero    .L2     b10           ;   max was in b10
         nop             3             ;   remaining delay slots
done:                                  ; Code moved!!!
```

A.3 Assembly code for median filtering (optimized)

```
;=================================================================
; ROUTINE:      median    Version 2   (Optimized)
;
; USAGE:        This is a C callable and can be called as:
;
;               short median(short val1, short val2,
;                            short val3, short val4,
;                            short val5, short val6,
;                            short val7, short val8);
;
;               The input values are in registers a4, b4, a6, b6,
;               a8, b8, a10, b10 according to the calling
;               convention of the compiler.
;
; DESCRIPTION:  Finds the median value of 8 data points passed
;               and returns the median value.
;
;               Uses subroutine "max_and_zero" (found below).
;               It calls "max_and_zero" five times, saving fourth
;               and fifth largest values, which are used to
;               calculate median.
;
;               Optimization has been done.
;
```

```
; Eduardo Asbun      Purdue University         March 16, 1998
;====================================================================
         .title         "Routine:    median    March 16, 1998"
         .def        _median
         .sect          "init"
_median:

         b       .s1     max_and_zero
         mvk     .s2     median_2,b5    ; save return address
         mvkh    .s2     median_2,b5
         mvk     .s1     0x0,a9         ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
                                        ;   to max value
 ||      mvk     .s2     0x0,b9         ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
                                        ; to max value
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x0,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x10,b7        ; Point to new max_b

median_2:
         b       .s1     max_and_zero
         mvk     .s2     median_3,b5    ; save return address
         mvkh    .s2     median_3,b5
         mvk     .s1     0x0,a9         ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
```

```
                                        ;   to max value
 ||      mvk     .s2     0x0,b9         ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
                                        ;   to max value
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x0,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x10,b7        ; Point to new max_b

median_3:
         b       .s1     max_and_zero
         mvk     .s2     median_4,b5    ; save return address
         mvkh    .s2     median_4,b5
         mvk     .s1     0x0,a9         ; Initialize "max_a"  (a9)
                                        ; a10 is pointer
                                        ;   to max value
 ||      mvk     .s2     0x0,b9         ; Initialize "max_b"  (b9)
                                        ; b10 is pointer
                                        ;   to max value
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x0,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x10,b7        ; Point to new max_b

median_4:
```

```
         b       .s1     max_and_zero
         mvk     .s2     median_5,b5   ; save return address
         mvkh    .s2     median_5,b5
         mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                       ; a10 is pointer
                                       ;   to max value
 ||      mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                       ; b10 is pointer
                                       ;   to max value
         cmpgt   .L1     a4,a9,a1      ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1      ; b4 ? max_b
   [ a1] mv      .d1     a4,a9         ; Save new max_a
 ||[ b1] mv      .d2     b4,b9         ; Save new max_b
 ||[ a1] mvk     .s1     0x0,a7        ; Point to new max_a
 ||[ b1] mvk     .s2     0x10,b7       ; Point to new max_b

median_5:

         b       .s1     max_and_zero
 ||      mv      .d1     a5,a3         ; save fourth largest
                                       ;   value  (Moved!!!)
         mvk     .s2     median_6,b5   ; save return value
         mvkh    .s2     median_6,b5
         mvk     .s1     0x0,a9        ; Initialize "max_a"  (a9)
                                       ; a10 is pointer
                                       ;   to max value
 ||      mvk     .s2     0x0,b9        ; Initialize "max_b"  (b9)
                                       ; b10 is pointer
                                       ;   to max value
```

```
         cmpgt   .L1     a4,a9,a1       ; a4 ? max_a
 ||      cmpgt   .L2     b4,b9,b1       ; b4 ? max_b
   [ a1] mv      .d1     a4,a9          ; Save new max_a
 ||[ b1] mv      .d2     b4,b9          ; Save new max_b
 ||[ a1] mvk     .s1     0x0,a7         ; Point to new max_a
 ||[ b1] mvk     .s2     0x10,b7        ; Point to new max_b

median_6:                               ; fifth largest value in a4

median_done:
         b        .s2    b3             ; Return to C calling routine
         add      .s1    a3,a5,a2       ; add fourth and fifth
                                        ;   largest values (Moved!!!)
         shr      .s1    a2,1,a4        ; calculate median
     nop         3

;=====================================================================
;=====================================================================
;=====================================================================
;  Routine:  "max_and_zero"
;
;    Finds the maximum value ("global maximum") of 2 sets of
;      4 points.
;
;    Values are in these registers:
;      set "a":   a4, a6, a8, a10
;      set "b":   b4, b6, b8, b10
;
;    Sets register containing global maximum value to zero.
```

```
;    Returns maximum in a4.
;
;  Registers used:
;    a1,  b1             Used for conditionals
;    b2                  Pointer to table of "zero" instructions
;    a5                  Return value (global maximum)
;    b5                  Return address
;    a4,  a6, a8, a10    Values for set "a"
;    b4,  b6, b8, b10    Values for set "b"
;    a7,  b7             pointers to maximum value in sets "a"
;                          and "b"
;    a9,  b9             max_a, max_b
;=================================================================
        .title          "Routine:  max_and_zero     Mar 16, 1998"
max_and_zero:
;=================================================================
; Part I. Find maximum of each set ("a" and "b")
;
;   The pointer to the current maximum is the offset of the
;   instruction entry that will be used to clear the register
;   holding the global maximum in Part II.
;   Each instruction is 4 bytes long, so offset is 4xk, where
;   "k" is the entry number in the table.
;=================================================================
|
find_max:
                    ; Moved some instructions to "median" !!!

         cmpgt   .L1     a6,a9,a1       ; a6 ? max_a
```

```
||       cmpgt   .L2     b6,b9,b1       ; b6 ? max_b
  [ a1] mv      .d1     a6,a9          ; Save new max_a
||[ b1] mv      .d2     b6,b9          ; Save new max_b
||[ a1] mvk     .s1     0x4,a7         ; Point to new max_a
||[ b1] mvk     .s2     0x14,b7        ; Point to new max_b

         cmpgt   .L1     a8,a9,a1       ; a8 ? max_a
||       cmpgt   .L2     b8,b9,b1       ; b8 ? max_b
  [ a1] mv      .d1     a8,a9          ; Save new max_a
||[ b1] mv      .d2     b8,b9          ; Save new max_b
||[ a1] mvk     .s1     0x8,a7         ; Point to new max_a
||[ b1] mvk     .s2     0x18,b7        ; Point to new max_b

         cmpgt   .L1     a10,a9,a1      ; a10 ? max_a
||       cmpgt   .L2     b10,b9,b1      ; b10 ? max_b
  [ a1] mv      .d1     a10,a9         ; Save new max_a
||[ b1] mv      .d2     b10,b9         ; Save new max_b
||[ a1] mvk     .s1     0xc,a7         ; Point to new max_a
||[ b1] mvk     .s2     0x1c,b7        ; Point to new max_b

;=============================================================
; Part II. Zero out global maximum
;
;   The register containing global maximum is identified and
;   cleared using a table, from which one instruction is executed
;   in the last delay slot of the branch that returns to the
;   calling routine.
;
;=============================================================
```

```
zero:                                   ; Zero out register that
                                        ; contains max

         cmpgt   .L1x    a9,b9,a1       ; Is "max_a" or "max_b" the
                                        ; global max?
   [ a1] mv      .s1     a9,a5          ; Return value is global
                                        ;   max (max_a)
 ||      mvk     .s2     zero_out_reg,b2
                                        ; This is address of table
                                        ;   below  (Moved!!!)
   [!a1] mv      .s1x    b9,a5          ; Return value is global max
                                        ;   (max_b)
 ||      mvkh    .s2     zero_out_reg,b2
                                        ; This is address of table
                                        ;   below  (Moved!!!)

   [ a1] add     .L2x    b2,a7,b2       ; Add correct offset
   [!a1] add     .L2     b2,b7,b2       ; Add correct offset
         b       .s2     b2             ; Jump to appropriate
                                        ;   instruction in table
         b       .s2     b5             ; Return to median
         nop             4              ; Wait to execute...

; This is a table from which only *one* instruction
; will be executed.
zero_out_reg:
         zero    .L1     a4             ;   max was in a4
                                        ;     (Offset =  0)
         zero    .L1     a6             ;   max was in a6
```

```
                                ;     (Offset =  4)
zero    .L1     a8              ;   max was in a8
                                ;     (Offset =  8)
zero    .L1     a10             ;   max was in a10
                                ;     (Offset = 12)
zero    .L2     b4              ;   max was in b4
                                ;     (Offset = 16)
zero    .L2     b6              ;   max was in b6
                                ;     (Offset = 20)
zero    .L2     b8              ;   max was in b8
                                ;     (Offset = 24)
zero    .L2     b10             ;   max was in b10
                                ;     (Offset = 28)
```

LIST OF REFERENCES

[1] K. Shen and E. J. Delp, "Color image compression using an embedded rate scalable approach," *Proceedings of the IEEE International Conference on Image Processing*, vol. III, pp. 34–37, Santa Barbara, California, October 26–29, 1997.

[2] K. Shen, *A Study of Real Time and Rate Scalable Image and Video Compression*. Ph.D. thesis, School of Electrical and Computer Engineering, Purdue University, December 1997.

[3] M. Saenz, P. Salama, K. Shen, and E. J. Delp, "An evaluation of color embedded wavelet image compression techniques," *SPIE Conference on Visual Communications and Image Processing '99*, pp. 282–293, San Jose, California, January 25–27, 1999.

[4] K. Shen and E. J. Delp, "Wavelet based rate scalable video compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 109–122, February 1999.

[5] E. J. Delp, P. Salama, E. Asbun, M. Saenz, and K. Shen, "Rate scalable image and video compression techniques," *Proceedings of the 42nd Midwest Symposium on Circuits and Systems*, Las Cruces, New Mexico, August 8–11, 1999.

[6] E. Asbun, P. Salama, K. Shen, and E. J. Delp, "Very low bit rate wavelet-based scalable video compression," *Proceedings of the IEEE International Conference on Image Processing*, pp. 948–952, Chicago, Illinois, October 4–7, 1998.

[7] E. Asbun, P. Salama, and E. J. Delp, "Encoding of predictive error frames in rate scalable video codecs using wavelet shrinkage," *Proceedings of the IEEE International Conference on Image Processing*, pp. 832–836, Kobe, Japan, October 25–28, 1999.

[8] E. Asbun, P. Salama, and E. J. Delp, "Preprocessing and postprocessing techniques for encoding predictive error frames in rate scalable video codecs," *Proceedings of the 1999 International Workshop on Very Low Bitrate Video Coding*, pp. 148–151, Kyoto, Japan, October 29–30, 1999.

[9] E. Asbun, P. Salama, and E. J. Delp, "A rate-distortion approach to wavelet-based encoding of predictive error frames," *Proceedings of the IEEE International Conference on Image Processing*, pp. 832–836, Vancouver, British Columbia, September 10–13, 2000.

[10] P. Salama, N. Shroff, and E. J. Delp, "A fast suboptimal approach to error concealment in encoded video streams," *Proceedings of the International Conference on Image Processing*, vol. II, pp. 101–104, Santa Barbara, California, October 26–29, 1997.

[11] P. Salama, N. Schroff, and E. J. Delp, "Error concealment in MPEG video streams over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 1129–1144, June 2000.

[12] E. Asbun and E. J. Delp, "Real-time error concealment in compressed digital video streams," *Proceedings of the Picture Coding Symposium '99*, pp. 345–348, Portland, Oregon, April 21–23, 1999.

[13] C. A. Poynton, *A Technical Introduction to Digital Video*. New York: Wiley, 1996.

[14] K. Jack, *Video Demystified*. Solana Beach, California: HighText Publications, second Ed., 1996.

[15] Optical Society of America, *The Science of Color*. Washington, D.C.: Optical Society of America, 1953.

[16] G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*. New York: Wiley, Second Ed., 1982.

[17] R. W. G. Hunt, *The reproduction of Colour*. England: Fountain Press, 1987.

[18] R. M. Boynton, *Human Color Vision*. Washington, D.C.: Optical Society of America, 1992.

[19] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*. New York: Chapman and Hall, 1996.

[20] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*. New York: Chapman and Hall, 1997.

[21] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards*. Norwell, Massachusetts: Kluwer Academic Publishers, Second Ed., 1997.

[22] L. Torres and E. J. Delp, "New trends in image and video compression," *Proceedings of the X European Signal Processing Conference (EUSIPCO)*, Tampere, Finland, September 5–8, 2000.

[23] International Organization for Standardization, *ISO/IEC 11172-2, Information Technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s*, May 1993. (MPEG-1 Video).

[24] International Organization for Standardization, *ISO/IEC 13818-2, Information Technology – Generic coding of moving pictures and associated audio information*, 1994. (MPEG-2 Video).

[25] International Organization for Standardization, *ISO/IEC 14496-2, Information Technology – Coding of Audio-Visual Objects: Video*, October 1998. (MPEG-4 Version 1, Part 2: Final Draft International Standard).

[26] International Telecommunication Union (ITU-T), *ITU-T Recommendation H.261: Line transmission of non-telephone signals. Video codec for audiovisual services at* $p \times 64$ *kbits*, March 1993.

[27] International Telecommunication Union (ITU-T), *ITU-T Recommendation H.263: Video Coding for Low Bitrate Communication*, March 1996.

[28] J. Taylor, *DVD Demystified*. New York, New York: McGraw-Hill, 1998.

[29] L. Torres and E. M. Kunt, *Video Coding: The Second Generation Approach*. Norwell, Massachusetts: Kluwer Academic Publishers, 1996.

[30] L. Chiariglione, "MPEG and multimedia communications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 5–18, February 1997.

[31] International Organization for Standardization, *ISO/IEC JTC1/SC29/WG11 N2995, MPEG-4 Overview*, October 1999. (Melbourne Version).

[32] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 19–21, February 1997.

[33] International Telecommunication Union (ITU-T), *ITU-T Recommendation H.320: Line transmission of non-telephone signals. Narrowband visual telephone systems and terminal equipment*, March 1993.

[34] International Telecommunication Union (ITU-T), *ITU-T Recommendation H.324: Terminal for low bitrate multimedia communication*, November 1995.

[35] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. 23, no. 23, pp. 90–93, January 1974.

[36] K. R. Rao and P. Yip, *Discrete Cosine Transform - Algorithms, Advantages, Applications*. New York: Academic Press, 1990.

[37] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

[38] S. Watanabe, "Karhunen-Loeve expansion and factor analysis. theoretical remarks and applications," *Transactions of the Fourth Prague Conference on Information Theory, Statistics, Decision Functions, and Random Processes*, pp. 635–660, Prague, Czechoslovakia, 1965.

[39] V. R. Algazi and D. J. Sakrison, "On the optimality of Karhunen-Loeve expansion," *IEEE Transactions on Information Theory*, vol. IT-15, no. 3, pp. 319–321, March 1969.

[40] W. H. Chen, C. H. Smith, , and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Transactions on Communications*, vol. COM-25, no. 9, pp. 1004–1009, September 1977.

[41] E. Feig and E. Linzer, "Discrete cosine transform algorithms for image data compression," *Proceedings of Electronic Imaging '90 East*, pp. 84–87, Boston, Massachusetts, 1990.

[42] S. Cucchi and M. Fratti, "A novel architecture for VLSI implementation of the 2-D DCT/IDCT," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 693–696, San Francisco, California, March 23–26, 1992.

[43] K. Aono, M. Toyokura, T. Araki, A. Ohtani, H. Kodama, and K. Okamoto, "A video digital signal processor with a vector-pipeline architecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1886–1894, December 1992.

[44] C. E. Shannon, *The Mathematical Theory of Communication*. Urbana, Illinois: The University of Illinois Press, 1949.

[45] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[46] B. Girod, E. Steinbach, and N. Farber, "Performance of the H.263 video compression standard," *Journal of VLSI Signal Processing*, vol. 17, no. 2–3, pp. 101–111, November 1997.

[47] W. Pennebaker, "Motion vector search strategies for MPEG-1," Technical Report ESC96-002, Encoding Science Concepts, Inc., February 1996.

[48] B. Girod, "Motion-compensating prediction with fractional-pel accuracy," *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 604–612, April 1993.

[49] J. Ribas-Corbera and D. L. Neuhoff, "Optimizing block size in motion-compensated video coding," *Journal of Electronic Imaging*, vol. 7, no. 1, pp. 155–165, January 1998.

[50] Z. Xiong, K. Ramchandran, M. T. Orchard, and Y.-Q. Zhang, "A comparative study of DCT- and wavelet-based image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 5, pp. 692–695, August 1999.

[51] M. Wien and W. Niehsen, "Space-frequency adaptive coding of motion compensated frame differences," *Proceedings of the 1999 Picture Coding Symposium*, pp. 65–68, Portland, Oregon, April 21–23, 1999.

[52] G. M. Schuster and A. K. Katsaggelos, "A theory for the optimal bit allocation between displacement vector field and displaced frame difference," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 9, pp. 1739–1751, December 1997.

[53] G. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, November 1998.

[54] I. H. Witten, R. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, June 1987.

[55] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[56] International Telecommunication Union (ITU-T), *ITU-T Recommendation H.263 Version 2: Video Coding for Low Bit Rate Communication*, February 1998. (H.263+).

[57] G. Côté, B. Erol, M. Gallant, and F. Kossentini, "H.263+: Video coding at low bit rates," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 849–866, November 1998.

[58] M. Vetterli, "Multi-dimensional sub-band coding: some theory and algorithms," *Signal Processing*, vol. 6, pp. 97–112, April 1984.

[59] J. W. Woods and S. D. O'Neil, "Subband coding of images," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 5, pp. 1278–1288, October 1986.

[60] J. W. Woods, *Subband Image Coding*. Norwell, Massachusetts: Kluwer Academic Publishers, 1991.

[61] M. R. Cinvalar and A. Puri, "Scalable video coding in frequency domain," *Proceedings of SPIE - Visual Communications and Image Processing*, pp. 1124–1134, Boston, Massachusetts, November 1992.

[62] Y.-Q. Zhang and S. Zafar, "Motion-compensated wavelet transform coding for color video compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 3, pp. 285–296, September 1992.

[63] F. Bosveld, *Hierarchical Video Compression using SBC*. Ph.D. thesis, Technical University Delft, The Netherlands, September 1996.

[64] P. H. Westerink, *Subband Coding of Images*. Ph.D. thesis, Technical University Delft, The Netherlands, October 1989.

[65] C. I. Podilchuk, N. S. Jayant, and N. Farvardin, "Three-dimensional subband coding of video," *IEEE Transactions of Image Processing*, vol. 4, no. 2, pp. 125–139, February 1995.

[66] B.-J. Kim and W. A. Pearlman, "An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPITH)," *Proceedings of the IEEE Data Compression Conference*, pp. 251–260, Snowbird, Utah, March 25–27, 1997.

[67] K. Ramchandran and M. Vetterli, "Rate-distortion optimal fast thresholding with complete JPEG/MPEG decoder compatibility," *IEEE Transactions of Image Processing*, vol. 3, no. 5, pp. 700–704, September 1994.

[68] M. Crouse and K. Ramchandran, "Joint thresholding and quantizer selection for transform image coding: Entropy constrained analysis and applications to baseline JPEG," *IEEE Transactions on Image Processing*, vol. 6, no. 2, pp. 285–297, February 1997.

[69] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics (SIAM), 1992.

[70] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[71] S. Mallat, *A Wavelet Tour of Signal Processing*. New York: Academic Press, 1998.

[72] C. S. Burrus and R. A. Gopinath, *Introduction to Wavelets and Wavelet Transforms: A Primer*. Upper Saddle River, New Jersey: Prentice Hall, 1998.

[73] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, New Jersey: Prentice Hall, 1993.

[74] G. M. Davis and A. Nosratinia, "Wavelet-based image coding: an overview," in *Applied and Computational Control, Signals, and Circuits*, B.N. Datta, Editor, Birkhauser, 1999.

[75] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 205–220, April 1992.

[76] R. A. DeVore, B. Jawerth, and B. J. Lucier, "Image compression through wavelet transform coding," *IEEE Transactions on Information Theory*, vol. 38, pp. 719–746, March 1992.

[77] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[78] N. J. Fliege, *Multirate Digital Signal Processing.* New York: Wiley, 1994.

[79] R. R. Coifman and M. V. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 713–718, March 1992.

[80] R. R. Coifman, Y. Meyer, and M. V. Wickerhauser, "Wavelet analysis and signal processing," in *Wavelets and their Applications*, Boston: B. Ruskai et al, Editors, Jones and Barlett, 1992.

[81] S. A. Martucci, "Signal extension and noncausal filtering for subband coding of images," *Proceedings of SPIE - Visual Communications and Image Processing*, pp. 137–148, Boston, Massachusetts, November 1991.

[82] C. M. Brislawn, "Preservation of subband symmetry in multirate signal coding," *IEEE Transactions on Signal Processing*, vol. 43, no. 12, pp. 3046–3050, December 1995.

[83] C. M. Brislawn, "Classification of nonexpansive symmetric extension transforms for multirate filter banks," *Applied and Computational Harmonic Analysis*, vol. 3, no. 4, pp. 337–357, October 1996.

[84] C. M. Brislawn, "Symmetric extension transforms," in *Wavelet Image and Video Compression*, P. N. Topiwala, Editor, Kluwer Academic Publishers, 1998.

[85] J. M. Shapiro, "Embedded image coding using zerotrees of wavelets coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, December 1993.

[86] A. Said and W. A. Pearlman, "New, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[87] S. LoPresto, K. Ramchandran, and M. Orchard, "Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework," *Proceedings of 1997 Data Compression Conference*, pp. 221–230, Snowbird, Utah, March 25–27, 1997.

[88] E. Ordentlich, M. Weinberger, and G. Seroussi, "A low-complexity modeling approach for embedded coding of wavelet coefficient," *Proceedings of 1998 Data Compression Conference*, Snowbird, Utah, March 30–April 1, 1998.

[89] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[90] ISO/IEC, *ITU-T Rec. T.800 CD15444-1, V1.0*, December 2000. (JPEG2000).

[91] I. Sodagar, H.-J. Lee, P. Hatrack, and Y.-Q. Zhang, "Scalable wavelet coding for synthetic/natural hybrid images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 2, pp. 244–254, March 1999.

[92] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang, "A zerotree wavelet video coder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 109–118, February 1997.

[93] Z. Xiong, K. Ramchandran, and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Transactions on Image Processing*, vol. 6, no. 5, pp. 677–693, May 1997.

[94] J.-R. Ohm, "Three-dimensional subband coding with motion compensation," *IEEE Transactions of Image Processing*, vol. 3, no. 5, pp. 559–571, September 1994.

[95] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 572–588, September 1994.

[96] B.-J. Kim and W. A. Pearlman, "Low-delay embedded 3-D wavelet color video coding with SPIHT," *Proceedings of SPIE - Visual Communications and Image Processing*, pp. 955–964, San Jose, California, January 28–30, 1998.

[97] A. Wang, Z. Xiong, P. A. Chou, and S. Mehrotra, "Three-dimensional wavelet coding of video with global motion compensation," *Proceedings of the IEEE Data Compression Conference*, pp. 404–413, Snowbird, Utah, March 29–31, 1999.

[98] A. Smolic, T. Sikora, and J.-R. Ohm, "Long-term global motion estimation and its application for sprite coding, content description, and segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 8, December 1999.

[99] S.-J. Choi and J. W. Woods, "Motion-compensated 3-D subband coding of video," *IEEE Transactions on Image Processing*, vol. 8, no. 2, pp. 155–167, February 1999.

[100] F. Kossentini, W. C. Chung, and M. J. T. Smith, "Rate-distortion-constrained subband video coding," *IEEE Transactions on Image Processing*, vol. 8, no. 2, pp. 145–154, February 1999.

[101] J. Lee and B. W. Dickinson, "Subband video coding with scene-adaptive hierarchical motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 3, pp. 459–466, April 1999.

[102] J. Vass, B.-B. Chai, K. Palaniappan, and X. Zhuang, "Significance-linked connected component analysis for very low bit-rate wavelet video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 4, pp. 630–647, June 1999.

[103] D. Marpe and H. L. Cycon, "Very low bit-rate video coding using wavelet based techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 85–94, February 1999.

[104] L. Yang, F. C. M. Martins, and T. R. Gardos, "Improving H.263+ scalability performance for very low bit rate applications," *Proceedings of SPIE - Visual Communications and Image Processing*, pp. 768–779, San Jose, California, January 25–27, 1999.

[105] B. Girod and N. Farber, "Feedback-based error control for mobile video transmission," *Proceedings of the IEEE*, vol. 87, no. 10, pp. 1707–1723, October 1999.

[106] J. D. Villasenor, Y.-Q. Zhang, and J. Wen, "Robust video coding algorithms and systems," *Proceedings of the IEEE*, vol. 87, no. 10, pp. 1724–1733, October 1999.

[107] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[108] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 623–656, October 1948.

[109] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[110] S. D. Servetto and K. Nahrstedt, "Video streaming over the public Internet: Multiple description codes and adaptive transport protocols," *Proceedings of the IEEE International Conference on Image Processing*, pp. 85–89, Kobe, Japan, October 25–28, 1999.

[111] S. D. Servetto, *Compression and reliable transmission of digital image and video signals*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1999.

[112] W.-T. Tan and A. Zakhor, "Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, June 1999.

[113] M. Gallant and F. Kossentini, "Robust and efficient layered H.263 Internet video based on rate-distortion optimized joint source/channel coding," *Submitted to IEEE Packet Video Workshop 2000*, Sardinia, Italy, 2000.

[114] V. A. Vaishampayan, "Design of multiple description scalar quantizers," *IEEE Transactions on Information Theory*, vol. IT-39, no. 3, pp. 821–834, March 1993.

[115] V. A. Vaishampayan, "Application of multiple description codes to image and video transmission over lossy networks," *Proceedings of the 7th International Packet Video Workshop*, Brisbane, Australia, 1996.

[116] S. Servetto, K. Ramchandran, V. A. Vaishampayan, and K. Nahrstedt, "Multiple description wavelet based image coding," *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 813–826, May 2000.

[117] S. Servetto, K. Ramchandran, and M. Orchard, "Image coding based on a morphological representation of wavelet data," *IEEE Transactions on Image Processing*, vol. 8, no. 9, pp. 1161–1174, September 1999.

[118] C. Chrysafis and A. Ortega, "Efficient context-based entropy coding for lossy wavelet image compression," *Proceedings of the IEEE Data Compression Conference*, pp. 241–250, Snowbird, Utah, March 25–27, 1997.

[119] J. H. McClellan, R. W. Schafer, and M. A. Yoder, *DSP First: A Multimedia Approach*. Upper Saddle River, New Jersey: Prentice Hall, 1998.

[120] K. Guttag, R. J. Gove, and J. R. V. Aken, "A single-chip multiprocessor for multimedia: the MVP," *IEEE Computer Graphics and Applications*, vol. 126, no. 6, pp. 53–64, November 1992.

[121] W. Lee and Y. Kim, "MPEG-2 video decoding on programmable processors: computational and architectural requirements," *Proceedings of SPIE*, vol. CR60, pp. pp 265–287, 1995.

[122] H. Fujiwara, M. L. Liou, M.-T. Sun, K.-M. Yang, M. Maruyama, K. Shomura, and K. Ohyama, "An all-ASIC implementation of a low bit-rate video codec," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 123–134, June 1992.

[123] S. K. Rao, M. Hatamian, M. T. Uyttendaele, S. Narayan, J. H. O'Neill, and G. A. Uvieghara, "A real-time p*64/MPEG video encoder chip," *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, pp. 32–33, San Francisco, California, February 24–26, 1993.

[124] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals*. Piscataway, New Jersey: IEEE Press, 1997.

[125] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PCs," *Communications of the ACM*, vol. 40, no. 1, pp. 25–38, January 1997.

[126] H. V. Sorensen and J. Chen, *A Digital Signal Processing Laboratory using the TMS320C30*. Upper Saddle River, New Jersey: Prentice Hall, 1997.

[127] K. Konstantinides, "VLIW architectures for media processing," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 16–19, March 1998. (in The Past, Present and Future of Image and Multidimensional Signal Processing).

[128] R. J. Gove, "MVP: A highly-integrated video compression chip," *Proceedings of the IEEE Data Compression Conference*, pp. 215–224, Snowbird, Utah, March 28-31, 1994.

[129] W. Lin, K. H. Goh, B. J. Tye, G. A. Powell, T. Ohya, and S. Adachi, "Real time H.263 video codec using parallel DSP," *Proceedings of the IEEE International Conference on Image Processing*, vol. II, pp. 586–589, Santa Barbara, California, October 26-29, 1997.

[130] M. Bolton, R. Boulton, J. Martin, S. Ng, and S. Turner, "A complete single-chip implementation of the JPEG image compression standard," *Proceedings of the IEEE Custom Integrated Circuits Conference*, p. 4, San Diego, California, May 12-15, 1991.

[131] C. Chrysafis and A. Ortega, "Line base, reduced memory, wavelet image compression," *Proceedings of the IEEE Data Compression Conference*, pp. 398–407, Snowbird, Utah, March 30 – April 1, 1998.

[132] Texas Instruments, *TMS320C8x System-Level Synopsis*. Dallas, Texas, April 1995.

[133] Texas Instruments, *TMS320C6000 Technical Brief*. Dallas, Texas, February 1999. (Literature Number: SPRU197D).

[134] N. Seshan, "High VelociTI processing," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 21–58, March 1998.

[135] D. A. Patterson and J. L. Hennesy, *Computer Architecture: A Quantitative Approach*. Palo Alto, CA: Morgan Kaufmann, Second Ed., 1996.

[136] P. Faraboschi, G. Desoli, and J. Fisher, "The latest word in digital and media processing," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 59–85, March 1998.

[137] P. Salama, N. B. Shroff, and E. J. Delp, "Error concealment in encoded video streams," in *Image Recovery Techniques for Image Compression Applications*, N. P. Galatsanos and A. K. Katsaggelos, Editors, Kluwer Academic Publishers, 1998.

[138] R. Talluri, "Error-resilient video coding in the ISO MPEG-4 standard," *IEEE Communications Magazine*, vol. 2, no. 6, pp. 112–119, June 1998.

[139] S. Wenger, G. Knorr, J. Ott, and F. Kossentini, "Error resilience support in H.263+," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 867–877, November 1998.

[140] N. Farber, B. Girod, and J. Villasenor, "Extensions of ITU-T recommendation H.324 for error-resilient video transmission," *IEEE Communications Magazine*, vol. 2, no. 6, pp. 120–128, June 1998.

[141] J. Liang and R. Talluri, "Tools for robust image and video coding in JPEG2000 and MPEG4 standards," *Proceedings of SPIE - Visual Communications and Image Processing*, pp. 40–51, San Jose, California, January 25–27, 1999.

[142] Y. Wang, S. Wenger, J. Wen, and A. K. Katsaggelos, "Error resilient video coding techniques," *IEEE Signal Processing Magazine*, vol. 17, no. 4, pp. 61–82, July 2000.

[143] Y. Wang and Q.-F. Zhu, "Error control and concealment for video communication: A review," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

[144] P. Salama, N. Shroff, and E. J. Delp, "Error concealment in embedded zerotree wavelet codecs," *Proceedings of the 1998 International Workshop on Very Low Bitrate Video Coding*, pp. 200–203, Urbana, Illinois, October 8–9, 1998.

[145] P. Salama, N. Shroff, E. J. Coyle, and E. J. Delp, "Error concealment techniques for encoded video streams," *Proceedings of the International Conference on Image Processing*, vol. I, pp. 9–12, Washington, DC, October 23–26, 1995.

[146] P. Salama, N. Shroff, and E. J. Delp, "A bayesian approach to error concealment in encoded video streams," *Proceedings of the International Conference on Image Processing*, vol. II, pp. 49–52, Lausanne, Switzerland, September 16–19, 1996.

[147] U. Black, *ATM: Foundation for broadband networks*. Upper Saddle River, New Jersey: Prentice-Hall, 1995.

[148] L. G. Cuthbert and J.-C. Sapanel, *ATM: The Broadband Telecommunications Solution*. IEE Telecommunications Series 29, IEE, 1993.

[149] W. Grimson, "An implementation of a computational theory of visual surface interpolation," *Computer Vision, Graphics, Image Processing*, vol. 22, no. 2, pp. 39–69, April 1983.

[150] W. Kwok and H. Sun, "Multidirectional interpolation for spatial error concealment," *IEEE Transactions on Consumer Electronics*, vol. 3, no. 39, pp. 455–460, August 1993.

[151] M. Ghanbari and V. Seferidis, "Cell-loss concealment in ATM video codecs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 3, pp. 238–247, June 1993.

[152] K. Shen and E. J. Delp, "A control scheme for a data rate scalable video codec," *Proceedings of the IEEE International Conference on Image Processing*, vol. II, pp. 69–72, Lausanne, Switzerland, September 16–19, 1996.

[153] M. L. Comer, K. Shen, and E. J. Delp, "Rate-scalable video coding using a zerotree wavelet approach," *Proceedings of the Ninth Image and Multidimensional Digital Signal Processing Workshop*, pp. 162–163, Belize City, Belize, March 3-6, 1996.

[154] A. N. Netravali and C. B. Rubinstein, "Luminance adaptive coding of chrominance signals," *IEEE Transactions on Communications*, vol. COM-27, no. 4, pp. 703–710, April 1979.

[155] J. O. Limb and C. B. Rubinstein, "Plateau coding of the chrominance component of color picture signals," *IEEE Transactions on Communications*, vol. COM-22, no. 3, pp. 812–820, June 1974.

[156] T. Chen, "Video coding and multimedia communications standards for Internet," *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, pp. 607–610, New York, New York, May 31–June 3, 1998.

[157] University of British Columbia, *H.263 / H.263+ Research Library, Release 0.2*, December 1999. (http://spmg.ece.ubc.ca/).

[158] M.-Y. Shen and C.-C. J. Kuo, "Artifact reduction in low bit rate wavelet coding with robust nonlinear filtering," *Proceedings of the 1998 IEEE SecMnd Workshop on Multimedia Signal Processing*, pp. 480–485, Redondo Beach, California, December 7–9, 1998.

[159] S. Oguz, Y. Hu, and T. Nguyen, "Image coding ringing artifact reduction using morphological postfiltering," *Proceedings of the 1998 IEEE Second Workshop on Multimedia Signal Processing*, pp. 628–634, Redondo Beach, California, December 7–9, 1998.

[160] T. O'Rourke and R. Stevenson, "Improved image decompression for reduced transform coding artifacts," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 6, pp. 490–499, December 1995.

[161] A. Nosratinia, "Embedded post-processing for enhancement of compressed images," *Proceedings of the IEEE Data Compression Conference*, pp. 62–71, Snowbird, Utah, March 29–31, 1999.

[162] J. G. Apostolopoulos and N. S. Jayant, "Postprocessing for very low bit-rate video compression," *IEEE Transactions on Image Processing*, vol. 8, no. 8, pp. 1125–1129, August 1999.

[163] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation via wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, March 1994.

[164] D. L. Donoho, "De-noising by soft-thresholding," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613–627, May 1995.

[165] D. L. Donoho and I. M. Johnstone, "Minimax estimation via wavelet shrinkage," *The Annals of Statistics*, vol. 26, no. 3, June 1998.

[166] X. Zong, A. Laine, and E. Geiser, "Speckle reduction and contrast enhancement of echocardiograms via multiscale nonlinear processing," *IEEE Transactions on Medical Imaging*, vol. 17, no. 4, pp. 532–540, August 1998.

[167] S. G. Chang, B. Yu, and M. Vetterli, "Image denoising via lossy compression and wavelet thresholding," *Proceedings of the IEEE International Conference on Image Processing*, vol. I, pp. 604–607, Santa Barbara, California, October 26–29, 1997.

[168] S. G. Chang and M. Vetterli, "Spatial adaptive wavelet thresholding for image denoising," *Proceedings of the IEEE International Conference on Image Processing*, vol. II, pp. 374–377, Santa Barbara, California, October 26–29, 1997.

[169] R. Gray, *Source Coding Theory*. Norwell, Massachusetts: Kluwer Academic Publishers, 1990.

[170] K. Ramchandran, A. Ortega, and M. Vetterli, "Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 533–545, September 1994.

[171] Y. Yang and S. S. Hemami, "A rate-distortion optimized motion compensated wavelet video coder," *Proceedings of the 1999 Picture Coding Symposium*, pp. 61–64, Portland, Oregon, April 21–23, 1999.

[172] M. J. Ruf and J. W. Modestino, "Operational rate-distortion performance for joint source and channel coding of images," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 305–320, March 1999.

[173] T. Wiegand, M. Lightstone, D. Mukherjee, T. G. Campbell, and S. K. Mitra, "Rate-distortion optimized mode selection for very low bit rate video coding and the emerging H.263 standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 2, pp. 182–190, April 1996.

[174] M. Gallant, G. Côté, and F. Kossentini, "An efficient computation-constrained block-based motion estimation algorithm for low bit rate video coding," *IEEE Transactions on Image Processing*, vol. 8, no. 12, pp. 1816–1823, December 1999.

[175] M. C. Chen and A. N. Wilson, "Rate-distortion optimal motion estimation algorithms for motion-compensated transform video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 2, pp. 147–158, April 1998.

[176] J. Li and S. Lei, "An embedded still image coder with rate-distortion optimization," *IEEE Transactions on Image Processing*, vol. 8, no. 7, pp. 913–924, July 1999.

[177] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23–50, November 1998.

[178] H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Research*, vol. 11, pp. 399–417, 1963.

[179] E. Asbun and C. Chen, "On the implementation of MPEG-4 motion compensation using the TMS320C62x," Technical Report SPRA586, Texas Instruments, August 1999.

[180] H. Sun, J. W. Zdepski, W. Kwok, and D. Raychaudhuri, "Error concealment algorithms for robust decoding of mpeg compressed video," *Signal Processing: Image Communication*, vol. 10, no. 4, pp. 249–268, September 1997.

[181] M. Wada, "Selective recovery of video packet loss using error concealment," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 5, pp. 807–814, June 1989.

[182] J. D. Villasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression," *IEEE Transactions of Image Processing*, vol. 4, no. 8, pp. 1053–1060, August 1995.

[183] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 245–267, 1998.

VITA

Eduardo Asbun received the Electronics Engineer degree from the Metropolitan University (UAM-A), Mexico City, Mexico, in 1990. He received the Master of Science in Electrical Engineering (M.S.E.E) degree in 1992, and the degree of Doctor of Philosophy (Ph.D.) in Electrical and Computer Engineering in 2000, both from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, U.S.A.

From 1990 to 1991, he was a Systems Engineer with IBM-Mexico. From 1993 to 2000, he was a Research Assistant with the School of Electrical and Computer Engineering, and a Teaching Assistant with the School of Electrical and Computer Engineering and the Department of Mathematics, Purdue University. He joined the Video and Image Processing Laboratory at Purdue University in 1996. In the summer of 1998, he was with the DSP Applications Group of Texas Instruments, Houston, Texas. In the summer of 1999, he was with the Media Technologies Laboratory of the DSP Solutions Research and Development Center of Texas Instruments, Dallas, Texas. Currently, he is with the Broadband Communications Sector of Motorola (formerly General Instrument), San Diego, California.

He was a Fulbright Scholar from 1991 to 1996. He was awarded the 1997/1998 Maria Elena Canto Neuberger Memorial Award from the Society of Hispanic Professional Engineers (SHPE) and the School of Electrical and Computer Engineering, Purdue University. In 2000, he received the Raymond Davis Scholarship from the Society for Imaging Science and Technology (IS&T). He has been a member of the Institute of Electrical and Electronics Engineers (IEEE) since 1991, and a member of the Society for Imaging Science and Technology (IS&T) since 2000.