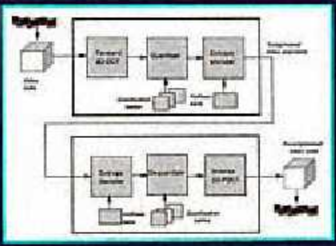
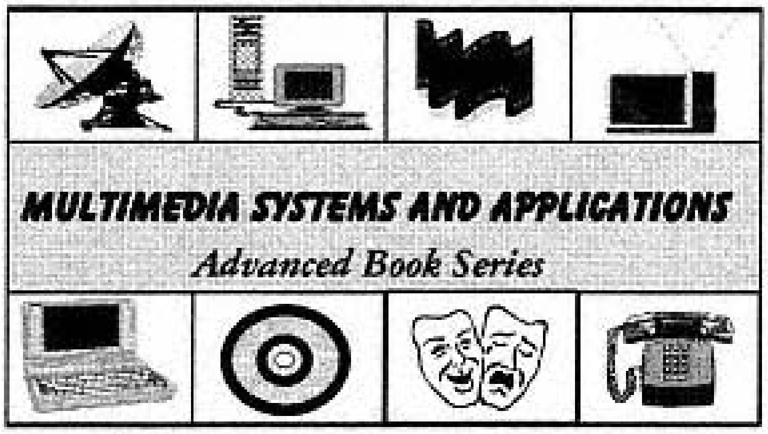
REAL-TIME VIDEO COMPRESSION

Techniques and Algorithms

Raymond Westwater
Borko Furht



Real-Time Video Compression


MULTIMEDIA SYSTEMS AND APPLICATIONS
Advanced Book Series

Real-Time Video Compression

Techniques and Algorithms

by

Raymond Westwater
Borko Furht

Florida Atlantic University

KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Printed on acid-free paper.

Printed in the United States of America

Contents

Preface

This book is on real-time video compression. Specifically, the book introduces the XYZ video compression technique, that operates in three dimensions, eliminating the overhead of motion estimation. First, video compression standards, MPEG and H.261/H.263, are described. They both use asymmetric compression algorithms, based on motion estimation. Their encoders are much more complex than decoders. The XYZ technique uses a symmetric algorithm, based on the Three-Dimensional Discrete Cosine Transform (3D-DCT). 3D-DCT was originally suggested for compression about twenty years ago, however at that time the computational complexity of the algorithm was to high, it required large buffer memory, and was not as effective as motion estimation. We have resurrected the 3D-DCT based video compression algorithm by developing several enhancements to the original algorithm. These enhancements made the algorithm feasible for real-time video compression in applications such as video-on-demand, interactive multimedia, and videoconferencing. The demonstrated results, presented in the book, suggest that the XYZ video compression technique is not only a fast algorithm, but also provides superior compression ratios and high quality of the video compared to existing standard techniques, such as MPEG and H.261/H.263. The elegance of the XYZ technique is in its simplicity, which leads to inexpensive VLSI implementation of a XYZ codec.

We would like to thank Jim Prince for conducting experiments in developing visually weighted quantizers for the XYZ algorithm, as well as a number of students from Florida Atlantic University, who participated in these experiments. We also want to thank Drs. Roy Levow, K. Genesan, and Matthew Evett, professors from Florida Atlantic University, Dr. Steve Rosenbaum from Cylex Systems, and Joshua Greenberg for constructive discussions during this project.

RAYMOND WESTWATER AND BORKO FURHT
BOCA RATON, JULY 1996.

1— The Problem of Video Compression

The problem of real-time video compression is a difficult and important one, and has inspired a great deal of research activity. This body of knowledge has been, to a substantial degree, embodied into the MPEG and H.261/H263 motion video standards. However, some important questions remain unexplored. This book describes one possible alternative to these standards that has superior compression characteristics while requiring far less computational power for its full implementation.

Since about 1989, moving digital video images have been integrated with programs. The difficulty in implementing moving digital video is the tremendous bandwidth required for the encoding of video data. For example, a quarter screen image (320 x 240 pixels) playing on an RGB video screen at full speed of 30 frames per second (fps) requires storage and transmission of 6.9 million bytes per second. This data rate is simply prohibitive, and so means of compressing digital video suitable for real-time playback are a necessary step for the widespread introduction of digital motion video applications.

Many digital video compression algorithms have been developed and implemented. The compression ratios of these algorithms varies according to the subjective acceptable level of error, the definition of the word compression, and who is making the claim. Table 1.1 summarizes video compression algorithms, their typical compression ratios reported in the literature, and their characteristics.

Table 1.1 Overview of video compression algorithms.

Compression Algorithm	**Typical Compression Ratio**	**Characteristics**
Intel RTV/Indeo	**3:1**	A 128X240 data stream is interpolated to 256X240. Color is subsampled 4:1. A simple 16 bit codebook is used without error correction. Frame differencing is used.
Intel PLV	**12:1**	A native 256X240 stream is encoded using vector quantization and motion compensation. Compression requires specialized equipment.
IBM Photomotion	**3:1**	An optimal 8-bit color palette is determined, and run-length encoding and frame differencing are used.
Motion JPEG	**10:1**	Uses 2-D DCT to encode individual frames. Gives good real-time results with inexpensive but special-purpose equipment. This technique supports random-access since no frame differencing is used.
Fractals	**10:1**	Fractals compress natural scenes well, but require tremendous computing power.
Wavelets	**20:1**	2-D and 3-D wavelets have been used in the compression of motion video. Wavelet compression is low enough in complexity to compress entire images, and therefore does not suffer from the boundary artifacts seen in DCT-based techniques.
H.261/H263	**50:1**	Real-time compression and decompression algorithm for video telecommunications. It is based on 2-D DCT with simple motion estimation between frames.
MPEG	**30:1**	Uses 2-D DCT with motion estimation and interpolation between frames. The MPEG standard is difficult and expensive to compress, but plays back in real-time with inexpensive equipment.

An ideal video compression technique should have the following characteristics:

• Will produce levels of compression rivaling MPEG without objectionable artifacts.

• Can be played back in real time with inexpensive hardware support.

• Can degrade easily under network overload or on a slow platform.

• Can be compressed in real time with inexpensive hardware support.

1.1— Overview of Video Compression Techniques

The JPEG still picture compression standard has been extremely successful, having been implemented on virtually all platforms. This standard is fairly simple to implement, is not computationally complex, and gets 10:1 to 15:1 compression ratios without significant visual artifacts. This standard is based upon entropy encoding of quantized coefficients of the discrete cosine transformation of 8x8 blocks of pixel data.

Figure 1.1 shows the block diagram of both the JPEG compression and decompression algorithms. A single frame is subdivided into 8x8 blocks, each of which is independently processed. Each block is transformed into DCT space, resulting in an 8x8 block of DCT coefficients. These coefficients are then quantized by integer division by constants. The quantizing constant for each DCT coefficient is chosen to produce minimal visual artifacts, while maximally reducing the representational entropy of the coefficients. The quantized coefficients are then entropy coded into a compressed data stream. The reduced entropy of the quantized coefficients is reflected in the higher compression ratio of the data.

The Motion JPEG (M-JPEG) uses the JPEG compression for each frame. It provides random access to individual frames, however the compression ratios are too low (same as in JPEG), because the technique does not take advantage of the similarities between adjacent frames.

The MPEG moving compression standard is an attempt to extend DCT-based compression into moving pictures. MPEG encodes frames by estimating the motion difference between the frames, and encoding the differences into roughly JPEG format. Unfortunately, motion estimation is computationally complex, requires specialized equipment to encode, and adds considerable complexity to the algorithm. Figure 1.2 illustrates the MPEG compression algorithm for predictive frames.

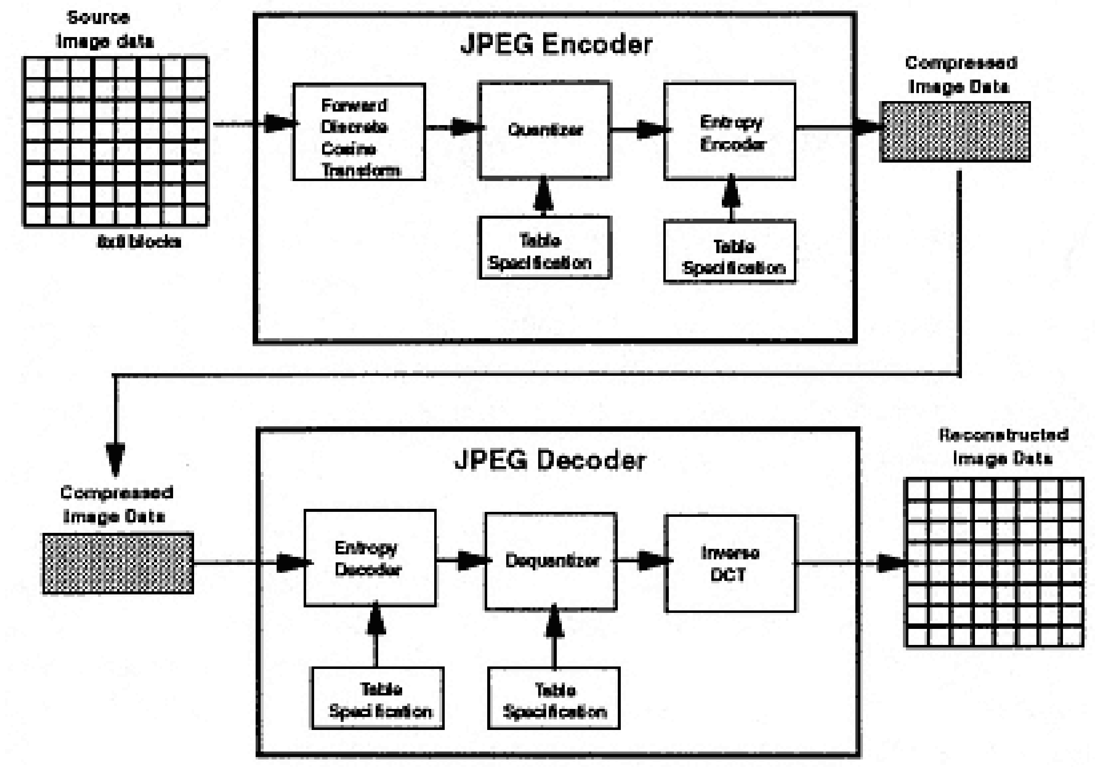


Figure 1.1
JPEG compression and decompression algorithms.

One of the most promising new technologies is wavelet-based compression [VK95]. Figure 1.3 illustrates a simple wavelet transform: subband decomposition. The image as a whole is subdivided into frequency subbands, which are then individually quantized. One of the most attractive features of this system is that it is applied to the image as a whole, thereby avoiding the edge artifacts associated with the block-based DCT compression schemes.

The wavelet transform can be applied to the time dimension as well. Experience has shown that this decomposition does not give as good compression results as motion compensation. As there are no other compression algorithms capable of such high compression ratios, MPEG is considered the existing "state-of-the-art".

The XYZ algorithm is a natural extension of the research that has been done in video compression. Much work has been done in the development of transform-based motion video compression algorithms, and in the development of quantizing factors based on the sensitivity of the human eye to various artifacts of compression.

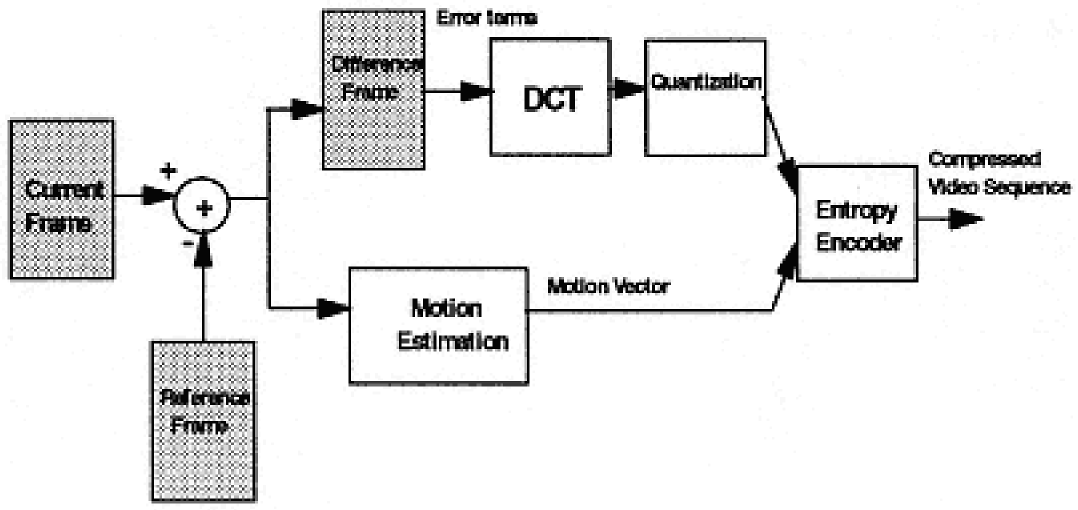


Figure 1.2
MPEG compression algorithm for predictive frames. MPEG
adds motion estimation to the JPEG model.

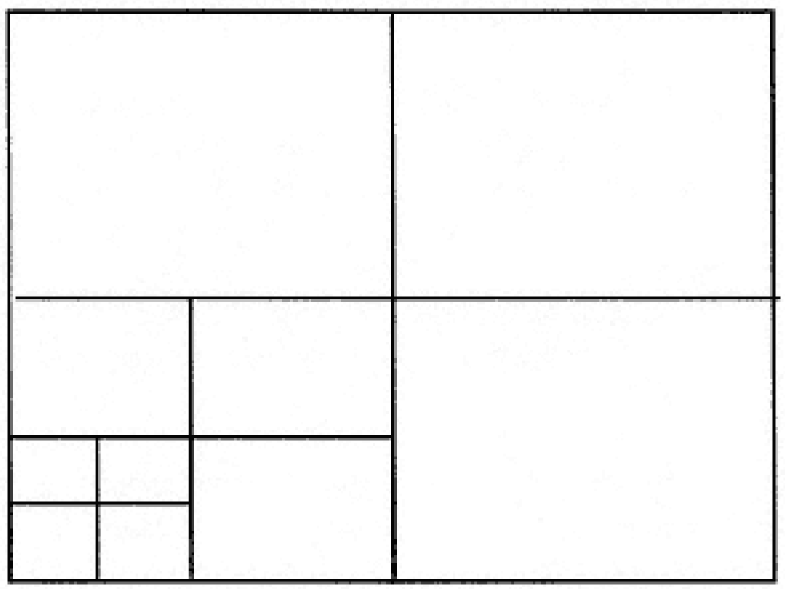


Figure 1.3
Octave-band or wavelet decomposition of a still
image into unequal subbands.

XYZ compression is an alternative extension of DCT encoding to moving pictures. Sequences of eight frames are collected into a three-dimensional block to which a three-dimensional DCT will be applied. The transformed data is then quantized.

These quantizing constants are demonstrated to cause artifacts which are minimally visible. The resulting data stream is then entropy coded. This process strongly resembles the JPEG encoding process, as illustrated in Figure 1.4.

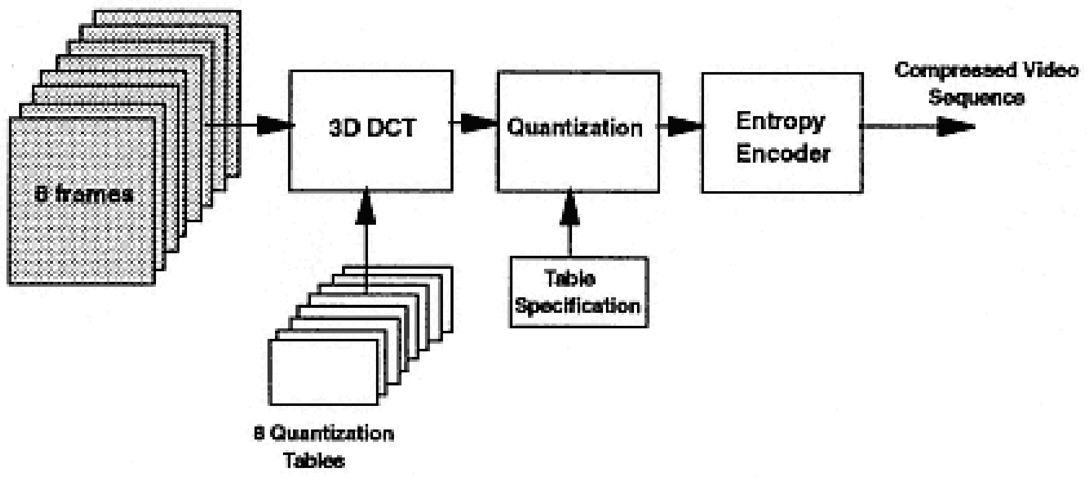


Figure 1.4
XYZ compression algorithm.

This algorithm is built upon a considerable body of published work. The three-dimensional DCT has been used to encode errors after motion estimation has been performed [RP77], and true three-dimensional DCT-based compression algorithms have been developed where the quantizers were based upon minimization of introduced mean square error [NA77]. These algorithms have fallen into disfavor because they were considered to require excessive computation, required too much buffer memory, and were not as effective as motion estimation. This book refutes these arguments.

Work in visibility of artifacts produced by quantization has also been done [CR90]. Visibility of two-dimensional quantization artifacts has been thoroughly explored for the DCT transforms space. The XYZ algorithm extends this work to quantization of three-dimensional DCT coefficients.

1.2— Applications of Compressed Video

Video compression techniques made feasible a number of applications. Four distinct applications of the compressed video can be summarized as: (a) consumer broadcast television, (b) consumer playback, (c) desktop video, and (d) videoconferencing.

Consumer broadcast television, which includes digital video delivery to homes, typically requires a small number of high-quality compressors and a large number of low-cost decompressors. Expected compression ratio is about 50:1.

Consumer playback applications, such as CD-ROM libraries and interactive games, also require a small number of compressors and a large number of low-cost decompressors. The required compression ratio is about 100:1.

Desktop video, which includes systems for authoring and editing video presentations, is a symmetrical application requiring the same number of encoders and decoders. The expected compression ratio is in the range from 5:1 to 50:1.

Videoconferencing applications also require the same number of encoders and decoders, and the expected compression ratio is about 100:1.

Table 1.2 Applications of the compressed video and current video compression standards.

Application	**Bandwidth**	**Standard**	**Size**	**Frame Rate [frames/sec]**
Analog Videophone	5-10 Kbps	none	170x128	2-5
Low Bitrate Video Conferencing	26-64 Kbps	H.263	128x96 176x144	15-30
Basic Video Telephony	64-128 Kbps	H.261	176x144 352x288	10-20
Video Conferencing	>= 384 Kbps	H.261	352x288	15-30
Interactive Multimedia	1-2 Mbps	MPEG-1	352x240	15-30
Digital TV - NTSC	3-10 Mbps	MPEG-2	720x480	30
High Definition Television	15-80 Mbps	MPEG-2	1200x800	30-60

Table 1.2 summarizes applications of the compressed video, by specifying current standards used in various applications, the required bandwidth, and typical frame sizes and frame rates.

1.3— Image and Video Formats

A digital image represents a two-dimensional array of samples, where each sample is called a pixel. Precision determines how many levels of intensity can be represented, and is expressed as the number of bits/sample. According to precision, images can be classified into: (a) binary images, represented by 1 bit/sample, (b) computer graphics, represented by 4 bits/sample, (c) grayscale images, represented by 8 bits/sample, and color images, represented with 16, 24 or more bits/sample.

According to the trichromatic theory, the sensation of color is produced by selectively exciting three classes of receptors in the eye. In a RGB color representation system, shown in Figure 1.5, a color is produced by adding three primary colors: red, green, and blue (RGB). The straight line, where R=G=B, specifies the gray values ranging from black to white.

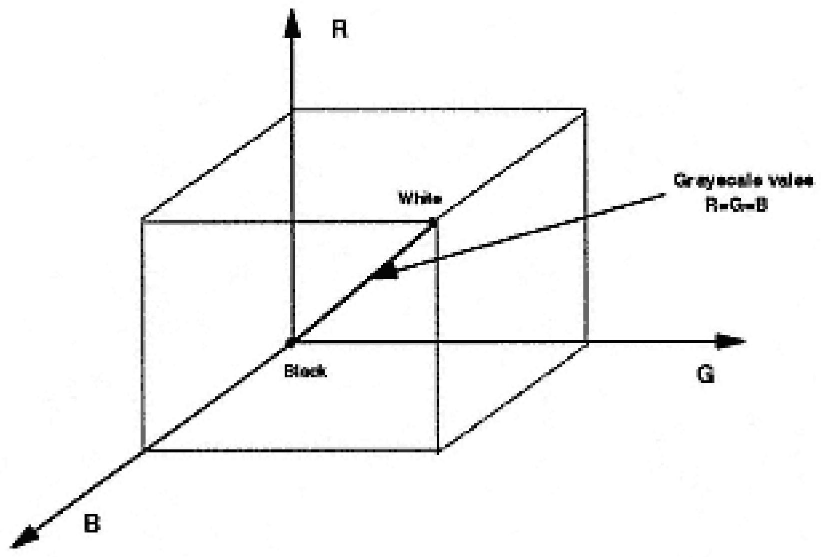


Figure 1.5
The RGB representation of color images.

Another representation of color images, YUV representation, describes luminance and chrominance components of an image. The luminance component provides a grayscale version of the image, while two chrominance components give additional information that converts the grayscale image to a color image. The YUV representation is more natural for image and video compression. The exact transformation from RGB to YUV representation, specified by the CCIR 601 standard, is given by the following equations:

$$Y = 0.299R + 0.587G + 0.114B \tag{1.1}$$

$$U = 0.564(B - Y) \tag{1.2}$$

$$V = 0.713(B - Y) \tag{1.3}$$

where Y is the luminance component, and U and V are two chrominance components.

An approximate RGB to YUV transformation is given as:

$$Y = 0.3R + 0.6G + 0.1B \tag{1.4}$$

$$U = B - Y \tag{1.5}$$

$$V = R - Y \tag{1.6}$$

This transformation has a nice feature that, when R+G+B, then Y=R=G=B, and U=V=0. In this case, the image is a grayscale image.

Color conversion from RGB to YUV requires several multiplications, which can be computationally expensive. An approximation, proposed in [W+94], can be calculated by performing bit shifts and adds instead multiplication operations. This approximation is defines as:

$$Y = \frac{R}{4} + \frac{G}{2} + \frac{B}{2} \tag{1.7}$$

$$U = \frac{B - Y}{2} \tag{1.8}$$

$$V = \frac{R - Y}{2} \tag{1.9}$$

This approximation also gives a simplified YUV to RGB transformation, expressed by:

$$R = Y + 2V \tag{1.10}$$

$$G = Y - (U + V) \tag{1.11}$$

$$B = Y + 2U \tag{1.12}$$

Another color format, referred to as YCbCr format, is intensively used for image compression. In YCbCr format, Y is the same as in a YUV system, however U and V components are scaled and zero-shifted to produce Cb and Cr, respectively, as follows:

$$Cb = \frac{U}{2} + 0.5 \tag{1.13}$$

$$Cr = \frac{V}{1.6} + 0.5 \tag{1.14}$$

In this way, chrominance components Cb and Cr are always in the range [0,1].

Computer Video Formats

Resolutions of an image system refers to its capability to reproduce fine detail. Higher resolution requires more complex imaging systems to represent these images in real time. In computer systems, resolution is characterized with number of pixels. Table 1.3 summarizes popular computer video formats, and related storage requirements.

Television Formats

In television systems, resolution refers to the number of line pairs resolved on the face of the display screen, expressed in cycles per picture height, or cycles per picture width. For example, the NTSC broadcast system in North America and Japan, denoted as 525/59.94, has about 483 picture lines.

The HDTV system will approximately double the number of lines of current broadcast television at approximately the same field rate. For example, a 1050x960 HDTV system will have 960 total lines. Spatial and temporal characteristics of conventional television systems (such as NTSC, SECAM, and PAL), and high-definition TV systems (HDTV) are presented in Tables 1.4 and 1.5, respectively [BF91].

Table 1.3 Characteristics of various computer video formats.

Computer Video Format	**Resolution (pixels)**	**Colors (bits)**	**Storage Capacity Per Image**
CGA - Color Graphics Adapter	320x200	4 (2 bits)	128,000 bits= 16 KB
EGA - Enhanced Graphics Adapter	640x350	16 (4 bits)	896,000 bits= 112 KB
VGA - Video Graphics Adapter	640x480	256 (8 bits)	2,457,600 bits= 307.2 KB
88514/A Display Adapter Mode	1024x768	256 (8 bits)	6,291,456 bits= 786.432 KB
XGA - Extended Graphics Array (a)	640x480	65,000 (24 bits)	6,291,456 bits= 786.432 KB
XGA - Extended Graphics Array (b)	1024x768	256 (8 bits)	6,291,456 bits =786.432 KB
SVGA - Super VGA	1024x768	65,000 (24 bits)	2.36 MB

Table 1.4 Spatial characteristics of television systems [BF91].

System	**Total Lines**	**Active Lines**	**Vertical Resolution**	**Optimal Viewing Distance [m]**	**Aspect Ratio**	**Horizontal Resolution**	**Total Picture Elements**
HDTV USA	1050	960	675	2.5	16/9	600	720,000
HDTV Europe	1250	1000	700	2.4	16/9	700	870,000
NTSC	525	484	242	7.0	4/3	330	106,000
PAL	625	575	290	6.0	4/3	425	165,000
SECAM	625	575	290	6.0	4/3	465	180,000

Table 1.5 Temporal characteristics of television systems [BF91].

System	Total Channel Width [MHz]	Video Baseband Y [MHz]	Video Baseband R-Y [MHz]	Video Baseband B-Y [MHz]	Scanning Rate Camera [Hz]	Scanning Rate HDTV Display [Hz]	Scanning Rate Convent. Display [Hz]
HDTV USA	9.0	10.0	5.0	5.0	59.94	59.94	59.94
HDTV Europe	12.0	14.0	7.0	7.0	50	100	50
NTSC	6.0	4.2	1.0	0.6	59.94	NA	59.94
PAL	8.0	5.5	1.8	1.8	50	NA	50
SECAM	8.0	6.0	2.0	2.0	50	NA	50

1.4— Overview of the Book

This book is divided into ten chapters:

1. **Video compression**. This current chapter introduces the problem of compressing motion video, illustrates the motivation for the 3-D solution chosen in the book, and briefly describes the proposed solution. Image and video formats are introduced as well.

2. **MPEG**. This chapter describes the MPEG compression standard. Important contributions in the field and related work are emphasized.

3. **H.261/H.263**. This chapter describes the compression standard for video telecommunications.

4. **XYZ compression**. The XYZ video compression algorithm is described in detail in this chapter. Both encoder and decoder are presented, as well as an example of compressing 8x8x8 video block.

5. **3-D DCT**. The theory of the Discrete Cosine Transform is developed and extended to three dimensions. A fast 3-D algorithm is developed.

6. **Quantization**. Discussion is presented on the issues of determining optimal quantizers using various error criteria. A model of Human Visual System is used to develop factors that weigh the DCT coefficients according to their relative visibility.

7. **Entropy coding**. A method for encoding the quantized coefficients is developed based on the stochastic behavior of the pixel data.

8. **VLSI architectures for XYZ codec**. Issues concerning real-time implementation of the XYZ compression algorithm are analyzed including the complexity of the algorithm and mapping the algorithm into various VLSI architectures.

9. **Results**. Obtained results of an implementation of the XYZ compression algorithm are presented.

10. **Conclusion**. Summary of contributions are outlined, emphasizing the real-time features of the compression algorithm, visual quality, and compression ratio. Directions for future research are given as well.

2— The MPEG Video Compression Standard

The Motion Picture Experts' Group was assembled by the International Standards Organization (ISO) to establish standards for the compression, encoding, and decompression of motion video. MPEG-1 [IS92b] is a standard supporting compression of image resolutions of approximately 352x288 at 30 fps into a data stream of 1.5 Mbps. This data rate is suitable for pressing onto CD-ROM. The MPEG-2 standard [IS93b] supports compression of broadcast television (704x576 at 30 fps) and HDTV (1920x1152 at 60 fps) of up to 60 Mpixels/sec (appx. 700 Mb) at compression ratios of roughly three times those expected of moving JPEG [IS92a] (playback rates of up to 80 Mbps).

The MPEG standard specifies the functional organization of a decoder. The data stream is cached in a buffer to reduce the effect of jitter in delivery and decode, and is demultiplexed into a video stream, an audio stream, and additional user-defined streams. The video stream is decoded into a "video sequence" composed of the sequence header and groups of pictures.

2.1— MPEG Encoder and Decoder

The specification of the MPEG encoder defines many compression options. While all of these options must be supported by the decoder, the selection of which options to support in compression is left to the discretion of the implementer. An MPEG encoder may choose compression options balancing the need for high compression ratios against the complexity of motion compensation or adaptive quantization calculations. Decisions will be affected by such factors as:

• A need for real-time compression. MPEG algorithms are complex, and there may not be sufficient time to implement exotic options on a particular platform.

• A need for high compression ratios. For highest possible compression ratios at highest possible quality, every available option must be exercised.

• A need for insensitivity to transmission error. MPEG-2 supports recovery from transmission errors. Some error recovery mechanisms are implemented by the encoder.

• Fast algorithms. Development of fast algorithms may make compression options available that would otherwise be impractical.

• Availability of specialized hardware. Dedicated hardware may increase the performance of the encoder to the point that additional compression options can be considered.

In the MPEG standard, frames in a sequence are coded using three different algorithms, as illustrated in Figure 2.1.

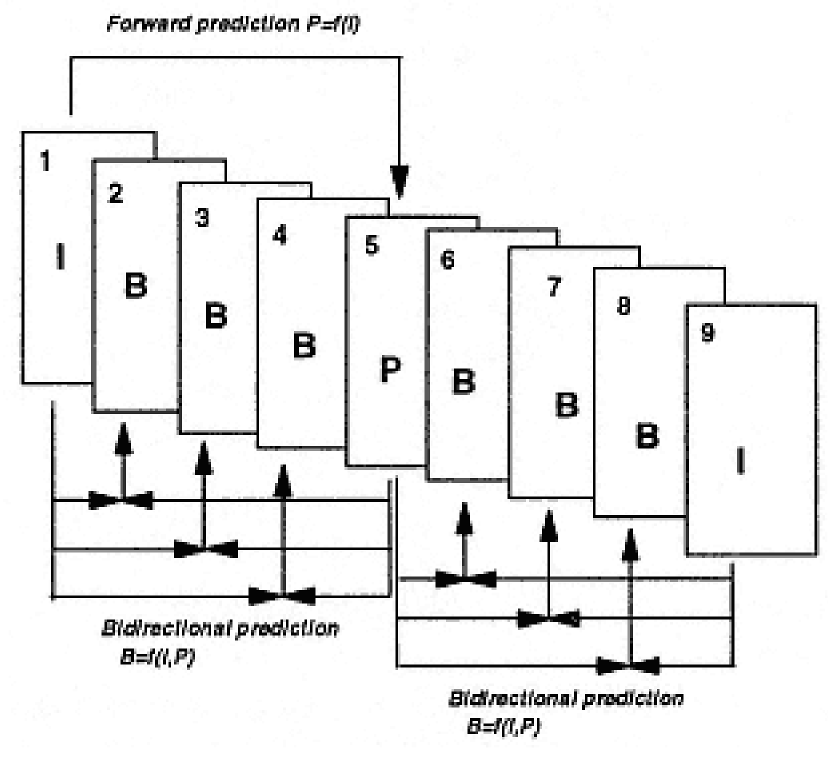


Figure 2.1
Types of frames in the MPEG standard.

I frames (intra frames) are self-contained and coded using a DCT-based technique similar to JPEG. I frames are used as random access points in MPEG streams, and they give the lowest compression ratios within MPEG.

P frames (predicted frames) are coded using forward predictive coding, where the actual frame is coded with reference to a pervious frame (I or P). This process is similar to H.261/H.263 predictive coding, except the previous frame is not always the closest previous frames, as in H.261/H.263 coding. The compression ratio of P frames is significantly higher than of I frames.

B frames (bidirectional or interpolated frames) are coded using two reference frames, a past and a future frame (which can be I or P frames). Bidirectional, or interpolated coding provides the highest amount of compression [Fur95b].

I, P, and B frames are described in more detail in Section 8.2. Note that in Figure 2.1, the first three B frames (2,3, and 4) are bidirectionally coded using the past frame I (frame 1), and the future frame P (frame 5). Therefore, the decoding order will differ from the encoding order. The P frame 5 must be decoded before B frames 2,3, and 4, and I frame 9 before B frames 6,7, and 8. If the MPEG sequence is transmitted over the network, the actual transmission order should be {1,5,2,,3,4,8,6,7,8}.

The MPEG application determines a sequence of I, P, and B frames. If there is a need for fast random access, the best resolution would be achieved by coding the whole sequence as I frames (MPEG becomes identical to Motion JPEG). However, the highest compression ratio can be achieved by incorporating a large number of B frames.

The block diagram of the MPEG encoder is given in Figure 2.2, while the MPEG decoder is shown in Figure 2.3.

I frames are created similarly to JPEG encoded pictures, while P and B frames are encoded in terms of previous and future frames. The motion vector is estimated, and the difference between the predicted and actual blocks (error terms) are calculated. The error terms are then DCT encoded and the entropy encoder is used to produce the compact code.

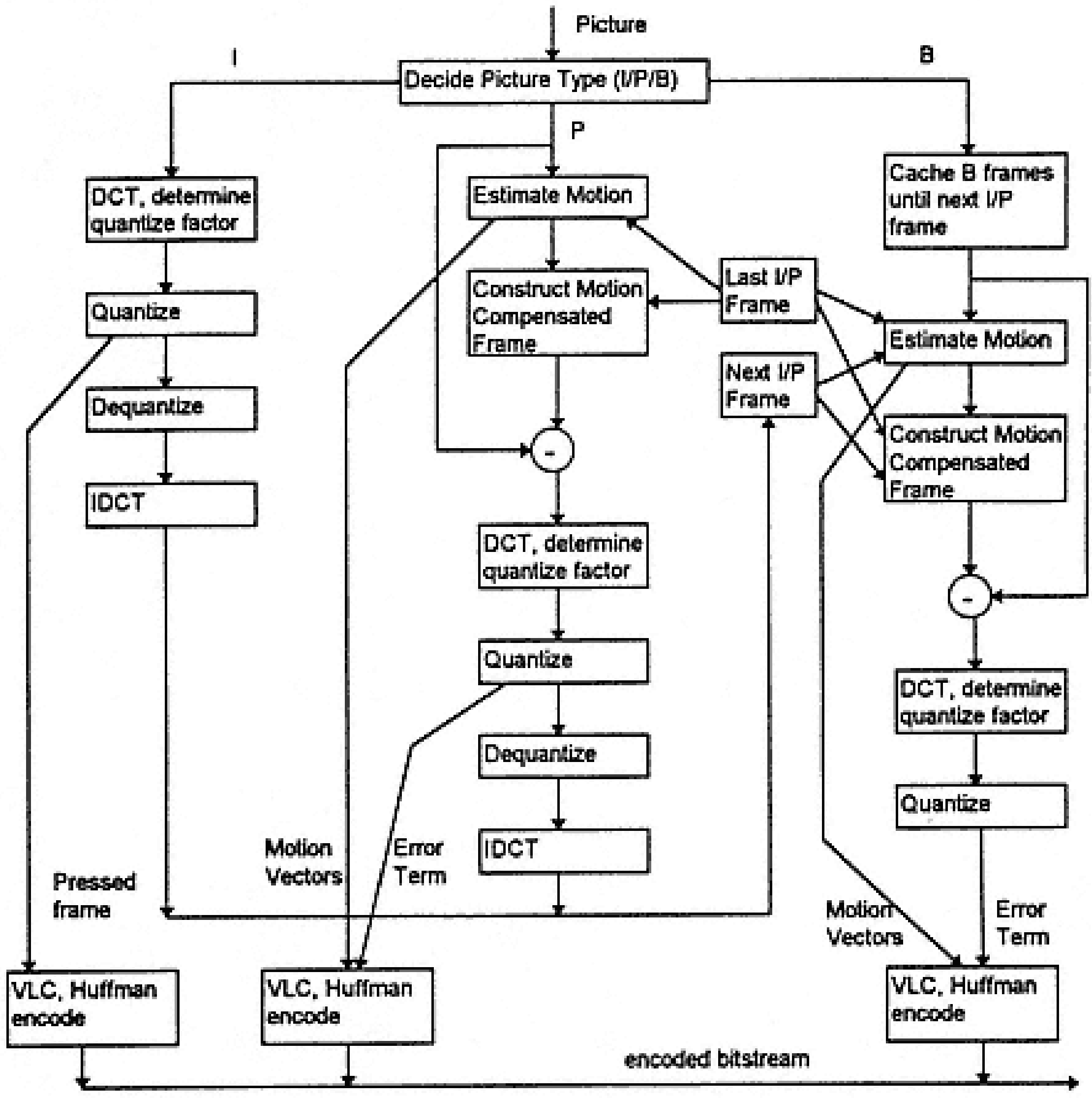


Figure 2.2
The block diagram of the MPEG encoder.

2.2— MPEG Data Stream

The MPEG specification defines a "video sequence" composed of a video sequence header and many Group-Of-Pictures (GOP), as illustrated in Figure 2.4. The video sequence header defines the video format, picture dimensions, aspect ratio, frame rate, and delivered data rate. Supported video formats include CCIR601, HDTV(16:9), and VGA. Supported chroma formats include "4:2:0" (YUV) and

"4:4:4" (RGB). A suggested buffer size for the video sequence is also specified, a number intended to buffer jitter caused by differences in decode time.

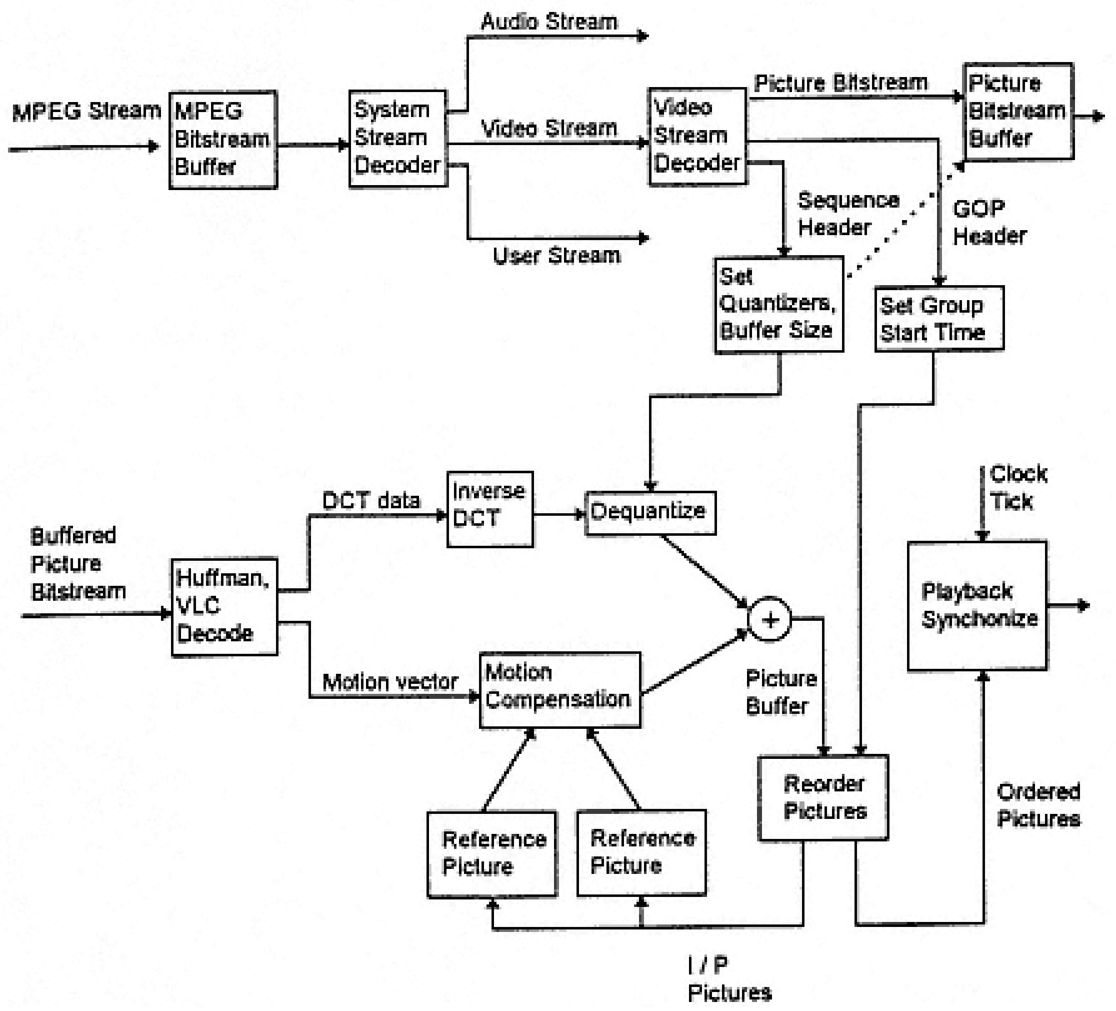


Figure 2.3
The block diagram of the MPEG decoder.

A GOP contains pictures that may be encoded into one of three supported compression formats. The GOP header contains a starting time for the group, and can therefore be used as a point of random access. Each frame within the GOP is numbered, and its number coupled with the GOP start time and the playback frame rate determines its playback time. Each picture is subdivided into "slices" and then into "macroblocks". A macroblock is composed of four 8x8 blocks of luminance data, and typically two 8x8 blocks of chrominance data, one Cr and one Cb.

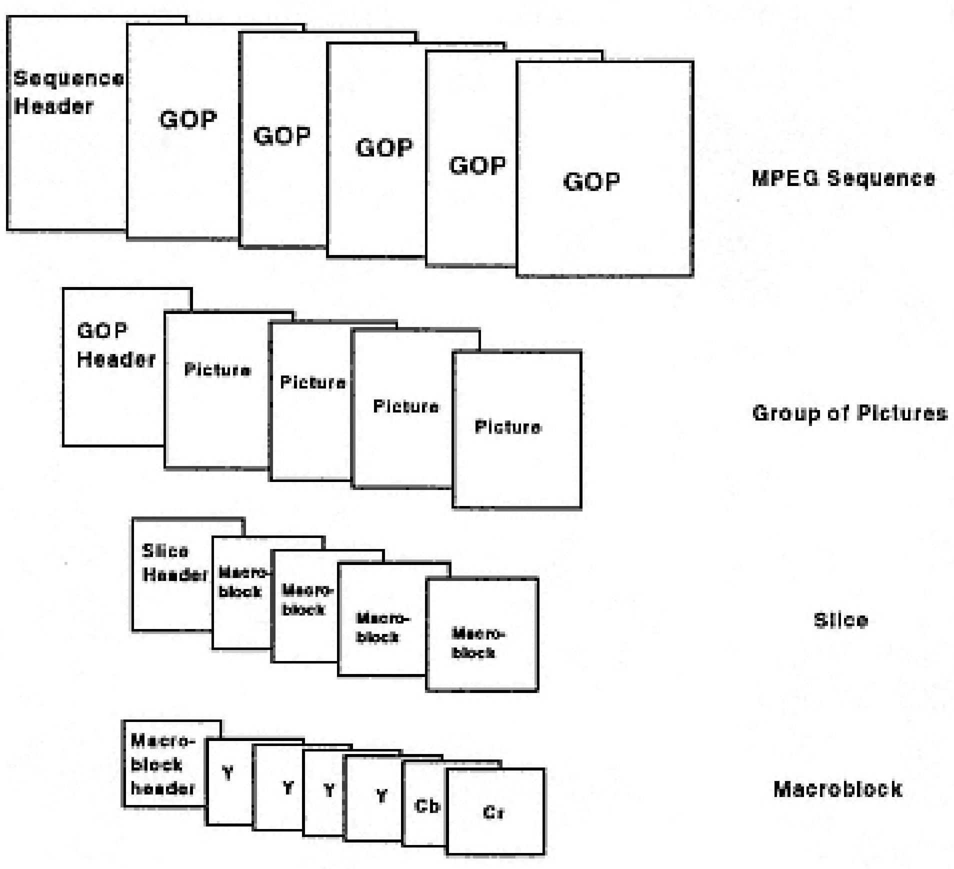


Figure 2.4
MPEG data stream.

I Picture Format

The I (Intraframe) picture format substantially corresponds to the JPEG format. These pictures are encoded by transformation into DCT space, quantization of the resultant coefficients, and entropy coding of the result. Transformation into DCT space is performed by an 8x8 DCT. Quantization is performed by reference to a user-loadable quantization table modified by a scale factor. This mechanism supports adaptive quantization at the cost of additional complexity - although 30% improvement in compression is claimed [PM93].

After quantization, the resulting coefficients are reordered in zig-zag order, run-length coded, variable-length coded, and entropy coded. The resulting data stream should show roughly JPEG levels of compression.

P Picture Format

The P (Predicted) picture format introduces the concept of motion compensation. Each macroblock is coded with a vector that predicts its value from an earlier I or P frame. The decoding process copies the contents of the macroblock-sized data at the address referenced by the vector into the macroblock of the P frame currently being decoded. Five bits of resolution are reserved for the magnitude of the vector in each of the x and y directions, meaning that 1024 possible data blocks may be referenced by the predicted macroblock. However, eight possible magnitude ranges may be assigned to those five bits, meaning as many as 8192 macroblocks might have to be evaluated to exhaustively determine the best vector. Each evaluation might require testing as many as 384 pixels, and a further complexity is seen in performing fractional interpolation of pixels (vector motions as small as 1/2 pixel are supported). Finally, the difference between the prediction and the macroblock to be compressed may be encoded in like fashion to I frame encoding above.

B Picture Format

The B (Bidirectional prediction) picture format is calculated with two vectors. A backwards vector references a macroblock-sized region in the previous I or P frame, the forward vector references a macroblock-sized region in the next I or P frame. For this reason, I and P frames are placed in the coded stream before any B frames that reference them.

The macroblock-sized regions referenced by the motion compensation vectors are averaged to produce the motion estimate for the macroblock being decoded. As with P frames, the error between the prediction and the frame being encoded is compressed and placed in the bitstream. The error factor is decompressed and added to the prediction to form the B frame macroblock.

Many demanding technical issues are raised by the MPEG specification. These include fast algorithms for the DCT, fast algorithms for motion vector estimation, algorithms for adaptive quantization, and decompression in environments that allow some errors.

3—
The H.261/H.263 Compression Standard for Video Telecommunications

ITU has developed a video conferencing standard H.324 at very low bitrate for the General Switched Telephone Network (GSTN) and mobile radio [IT95a, IT95b, IT93]]. The H.324 is a recommendation for real-time voice, data, and video over V.34 modems on the GSTN telephone network. It consists of five documents: (1) H.324 systems, (2) H.223 multiplex, (3) H.245 control, (4) H.263 video codec, and (5) G.273 speech codec. The H.261 coding standard provides coded video at bit rates 64 Kbits/s and above, whereas the H.263 video coding standard, proposed for H.324, provides coded video around 16 Kbits/s.

Figure 3.1 shows a block diagram of a generic multimedia system, compliant to the H.324 standard. The system consists of terminal equipment, GSTN modem, GSTN network, multipoint control unit (MCU), and other system operation entities.

Video equipment includes cameras, monitors, and video processing units to improve compression. Audio equipment includes microphone, speakers, telephone instrument, and attached audio devices. Data application equipment includes computers, non-standardized data application protocols, telematic visual aids such as electronic whiteboards, etc.

GSTN network interface supports appropriate signaling, ringing functions and voltage levels in accordance with national standards.

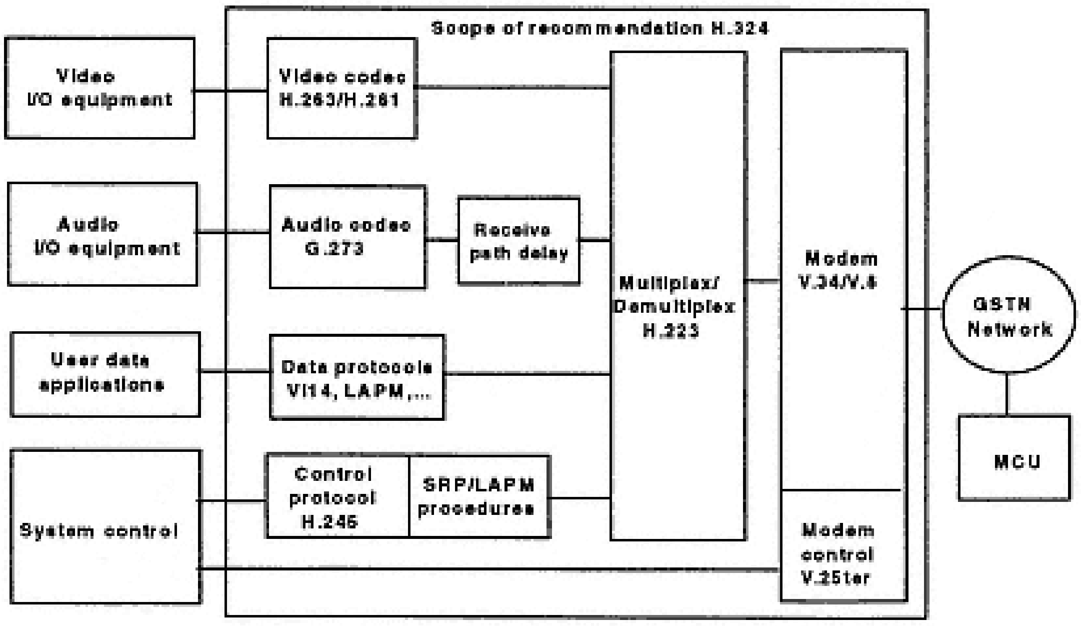


Figure 3.1
Block diagram of a generic H.324-compliant multimedia system.

3.1—
Picture Formats for H.261/H.263 Video Codecs

All H.324 terminals support both the H.263 and H.261 video codecs. For the H.261 algorithm two formats are defined: CIF and QCIF, while for the H.263 algorithm three additional formats are specified: SQCIF, 4CIF, and 16CIF.

The Common Intermediate Format (CIF) is a noninterlaced format, based on 352x288 pixels per frame at 30 frames per second. These values represent half the active lines of 625/25 television signal and the picture rate of a 525/30 NTSC signal. Therefore, 625/25 systems need only to perform a picture rate conversion, while NTSC systems need to perform only a line-number conversion.

Color pictures are coded using one luminance and two color-difference components (YCbCr format), specified by the CCIR 601 standard. The Cb and Cr components are subsampled by a factor of two on both horizontal and vertical directions, and have 176x144 pixels per frame. The picture aspect ratio for all five CIF-based formats is 4:3. Table 3.1 summarizes the picture formats for H.261 and H.263 codecs.

Table 3.1. Picture formats for H.261 and H.263 video codecs.

Picture format	Luminance pixels	Maximum frame rate [f/s]	Video source rate	Average coded bit rate	H.261 codec	H.263 codec
SQCIF	128 x 96	30	1.3 Mb/s	26 Kb/s	Optional	Required
QCIF	176 x 144	30	9 Mb/s	64 Kb/s (px64 Kbps)	Required	Required
CIF	352 x 288	30	36 Mb/s	384 Kb/s (px64 Kbps)	Optional	Optional
4CIF	704 x 576	30	438 Mb/s	3-6 Mb/s	Not defined	Optional
16CIF	1408 x 1152	50	2.9 Gb/s	20-60 Mb/s	Not defined	Optional

3.2— H.261/H.263 Video Encoder

The H.261/H.263 video encoder combines intraframe and interframe coding to provide fast processing for on-the-fly video [Oku95, FSZ95, BK95, Fur95b]. The algorithm creates two types of frames:

(1) DCT-based intraframes, compressed using DCT, quantization, and entropy (variable-length) coding (similarly to JPEG) [Fur95a], and

(2) predictive interframes, compressed using Differential Pulse Code Modulation (DPCM) and motion estimation.

The block diagram of the video encoder is shown in Figure 3.2. The H.261/H.263 coding algorithm begins by coding an intraframe block and then sends it to the video multiplex coder. The same frame is then decompressed using the inverse quantizer and inverse DCT, and then stored in the frame memory for interframe coding.

During the interframe coding, the prediction based on the DPCM algorithm is used to compare every macro block of the actual frame with the available macro blocks of the previous frame, as illustrated in Figure 3.3.

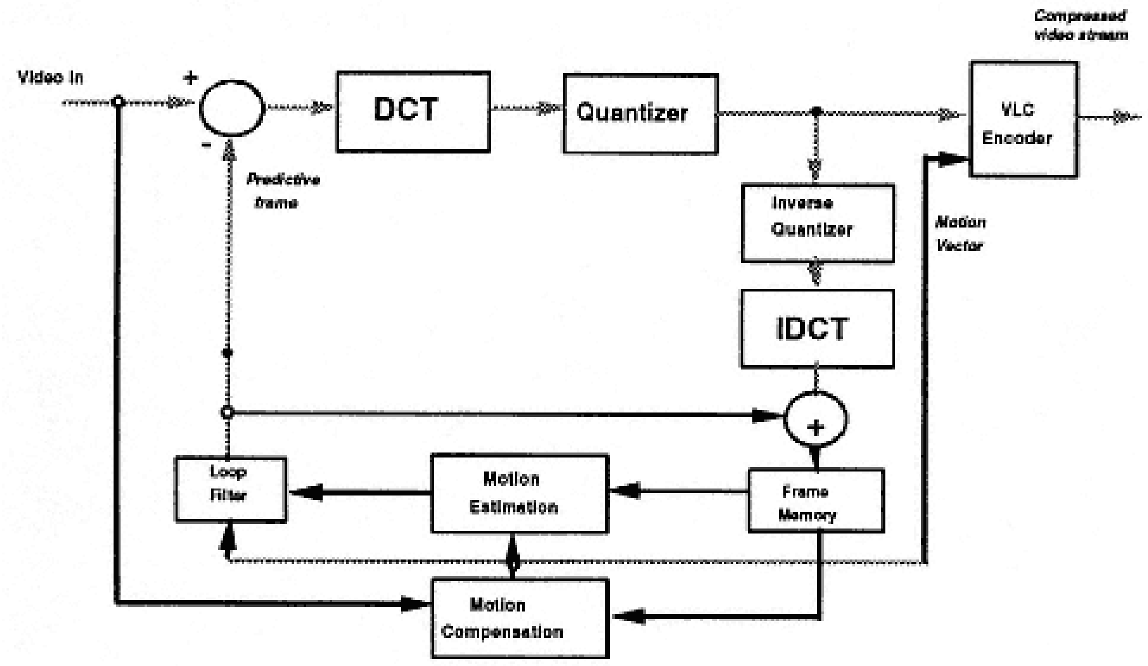


Figure 3.2
Block diagram of the H.261/H.263 video encoder.

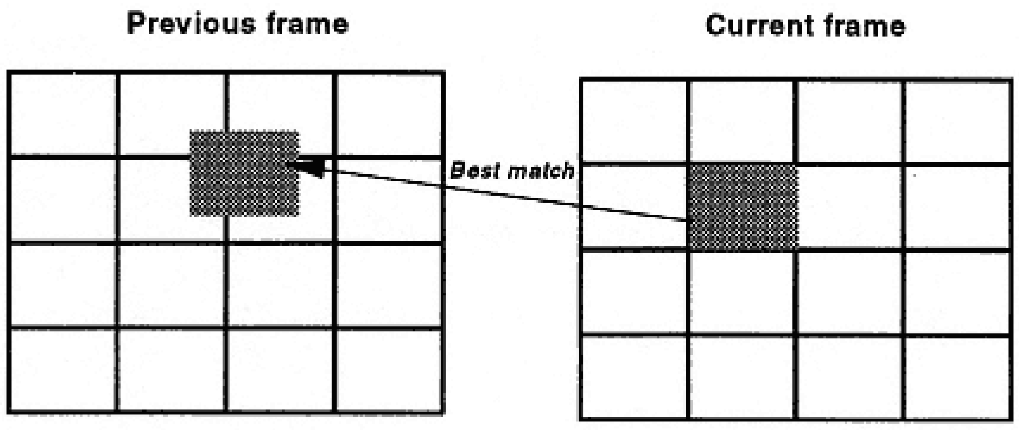


Figure 3.3
The principle of interframe coding in H.261/H.263 standard. Each macro block (16 x 16 pixels) in the current frame is compared with macro blocks from the previous frame to find the best match.

To reduce the encoding delay, only the closest previous frame is used for prediction. Then, the difference, created as error terms, is DCT-coded and quantized, and sent to the video multiplex coder with or without the motion vector. At the final step, variable-length coding (VLC), such as Huffman encoder, is used to produce more compact code. An optional loop filter can be used to minimize the prediction error by smoothing the pixels in the previous frame. At least one in every 132 frames should be intraframe coded, as shown in Figure 3.4.

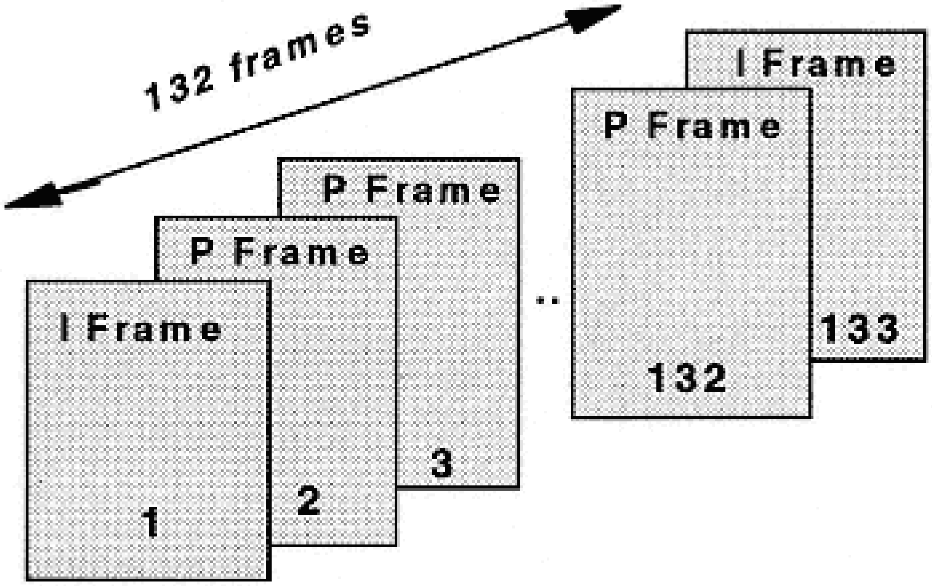


Figure 3.4
Types of frames in H.261/H.263 standard. At least every 132-nd frame should be the I frame.

The compressed data stream is arranged in a hierarchical structure consisting of four layers: Pictures, Group of Pictures (GOPs), Macro Blocks (MB), and Blocks, as illustrated in Figure 3.5.

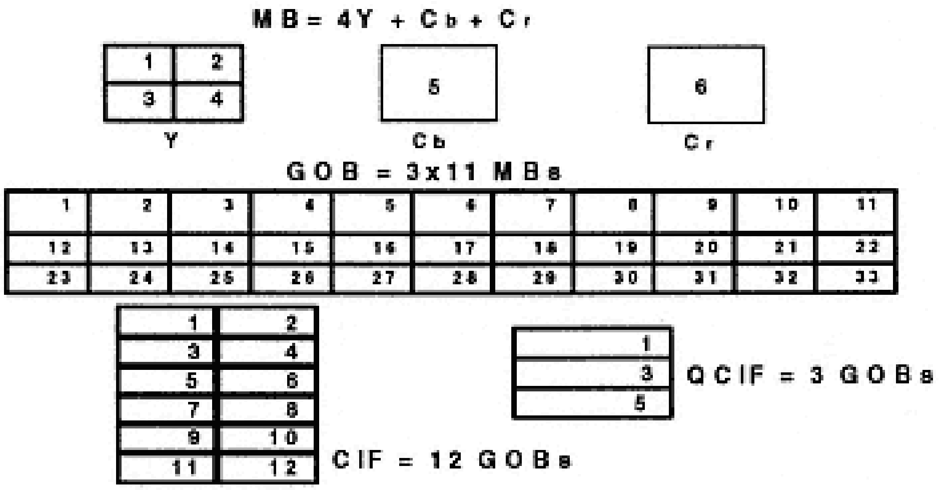


Figure 3.5
Hierarchical block structure of the H.261/H.263 video data stream.

3.3— The H.261/H.263 Video Decoder

The H.261/H.263 video decoder is shown in Figure 3.6. It consists of the receiver buffer, VLC decoder, inverse quantizer, inverse DCT, and the motion compensation, which includes frame memory and an optional loop filter [BSZ95, BK95, Fur95b].

In addition to the encoding and decoding of video, the audio data must also be compressed and decompressed. Special buffering and multiplexing/demultiplexing circuitry is required to handle the complexities of combining the video and audio.

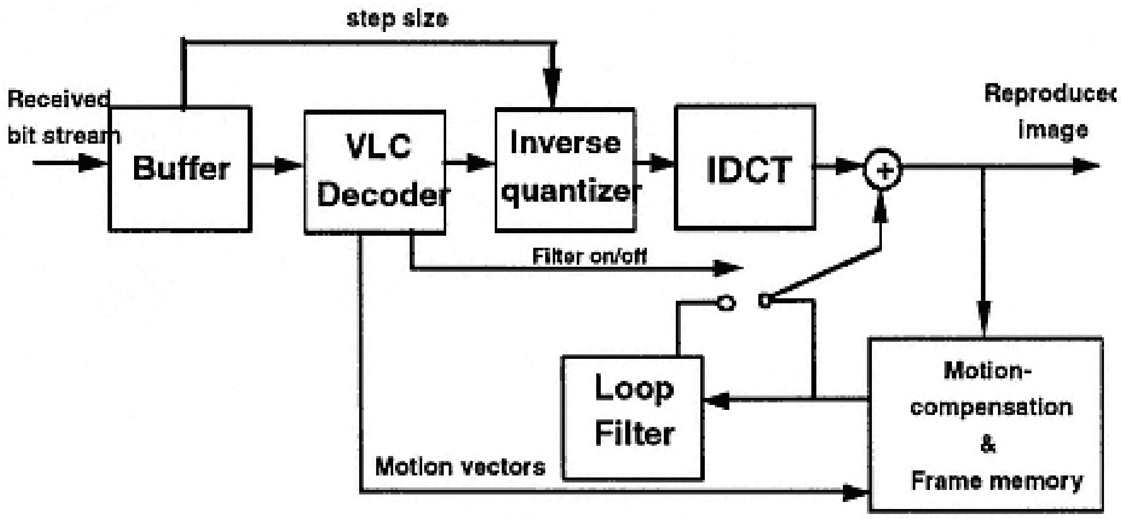


Figure 3.6
Block diagram of the H.261/H.263 video decoder.

4— The XYZ Video Compression Algorithm

The XYZ motion video compression algorithm relies on a different principle for compression of temporal information than do the MPEG and H.261/H.263 standards. While the MPEG and H.261/H.263 strategies look for motion vectors to represent a frame being compressed, the XYZ strategy more closely resembles the technique adopted by both MPEG and JPEG for intra-frame compression.

A continuous tone image can be represented as a two-dimensional array of pixel values in the spatial domain. The Forward Discrete Cosine Transform (FDCT) converts the two-dimensional image from spatial to frequency domain. In spatial representation the energy distribution of pixels is uniform, while in the frequency domain the energy is concentrated into few low-frequency coefficients.

Pixels in full-motion video are also correlated in the temporal domain, and the FDCT will concentrate the energy of pixels in the temporal domain just as it does in the spatial domain. The XYZ video compression is based on this property.

4.1— XYZ Compression Algorithm

The XYZ video compression algorithm is based on the three-dimensional DCT (3D DCT). This algorithm takes a full-motion digital video stream and divides it into groups of 8 frames. Each group of 8 frames is considered as a three-dimensional image, where X and Y are spatial components, and Z is the temporal component. Each frame in the image is divided into 8x8 blocks (like JPEG), forming 8x8x8 cubes, as illustrated in Figure 4.1. Each 8x8x8 cube is then independently encoded using the three blocks of the XYZ video encoder: 3D DCT, Quantizer, and Entropy encoder [WF95]. The block diagram of the XYZ compressor is shown in Figure 4.2.

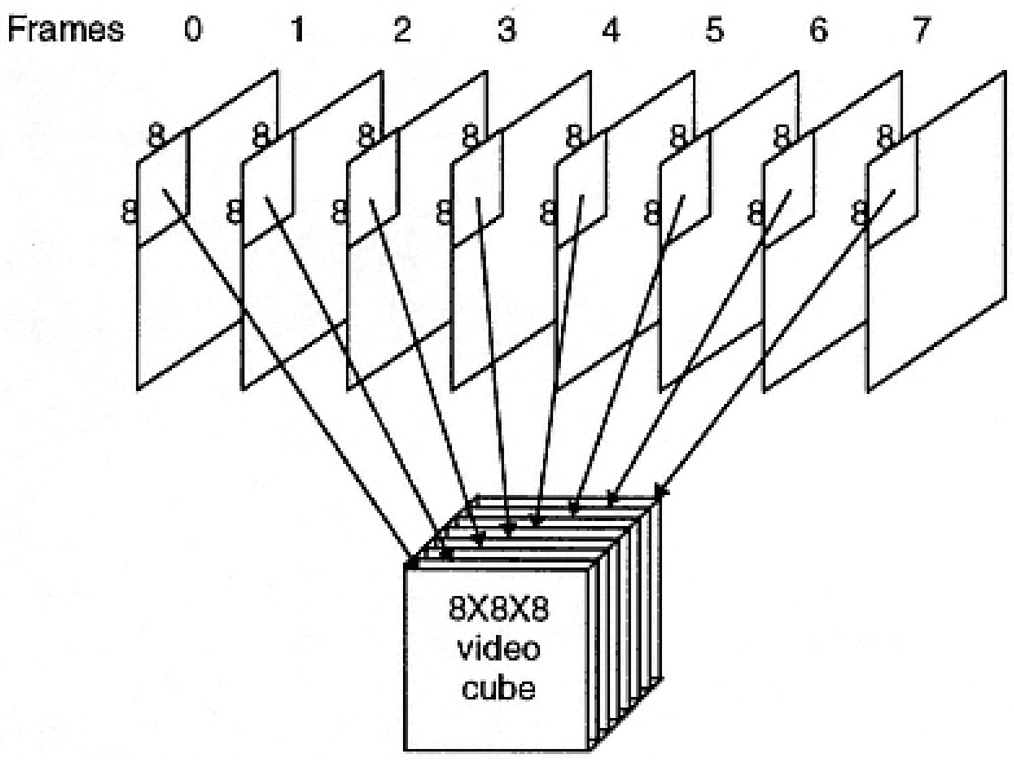


Figure 4.1
Forming 8x8x8 video cube for XYZ compression.

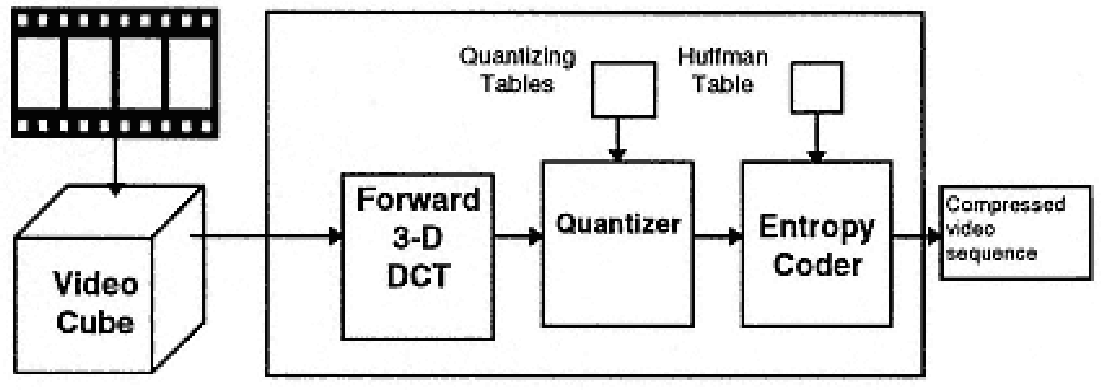


Figure 4.2
Block diagram of the XYZ compressor.

The original unsigned pixel sample values, typically in the range [0,255] are first shifted to signed integers, say in the range [-128, 127]. Then each 8x8x8 cube of 512 pixels is transformed into the frequency domain using the Forward 3D DCT:

$$F(u,v,w) = C(u)C(v)C(w) * \sum_{x=0}^{7}\sum_{y=0}^{7}\sum_{z=0}^{7} f(x,y,z) * \frac{\cos((2x+1)u\pi)}{16}\frac{\cos((2y+1)v\pi)}{16}\frac{\cos((2z+1)w\pi)}{16} \tag{4.1}$$

where:

x,y,z are index pixels in pixel space,

f(x,y,z) is the value of a pixel in pixel space,

u,v,w are index pixels in DCT space,

F(u,v,w) is a transformed pixel value in DCT space, and

$$C(i) = \frac{1}{\sqrt{2}} \quad \text{for i=0} \qquad C(i) = 1 \quad \text{for i>0} \tag{4.2}$$

The transformed 512-point discrete signal is a function in three dimensions, and contains both spatial and temporal information. Most of the energy is contained in few low-frequency coefficients, while the majority of the high-frequency coefficients have zero or near-zero values.

In the next step, all 512 DCT coefficients are quantized using a 512-element quantization table. Quantization introduces minimum error while increasing the number of zero-value coefficients. Quantization may also be used to discard visual information to which the human eye is not sensitive. Quantizer tables may be predefined, or adaptive quantizers may be developed and transmitted with the compressed data.

Quantization is performed according to the following equation:

$$F_q(u,v,w) = \left\lfloor \frac{F(u,v,w)}{Q(u,v,w)} \right\rfloor \tag{4.3}$$

where:

F(u,v,w) are the elements before the quantization,

F_q(u,v,w) are the quantized elements, and

Q(u,v,w) are the elements from the quantization table.

Each quantizer Q(u,v,w) is in the range [1,1024]. The result of the quantization operation is a collection of smaller-valued coefficients, a large number of which are 0. These coefficients are then converted into a compact binary sequence using an entropy coder (in this case, a Huffman coder).

The entropy coding operation starts with reordering the coefficients in descending order of expected value. This sequence has the benefit of collecting sequentially the largest number of zero-valued coefficients. The run-lengths of zero coefficients is computed, and the alphabet of symbols to be encoded becomes the run-length of zeros appended to the length of the non-zero coefficient. This binary sequence represents the compressed 8x8x8 block.

Figure 4.3 illustrates an example of encoding a video cube (eight frames of 8x8 pixels) using the XYZ compression algorithm. Figure 4.3 shows the original video cube, Figure 4.4a shows the DCT coefficients after the 3D DCT, and Figure 4.4b presents the quantized coefficients. Note that the largest quantized coefficient is Fq(0,0,0), which carries the crucial information on the video cube, while the majority of quantized coefficients are zero.

4.2— XYZ Decompression Algorithm

In XYZ decoding, the steps from the encoding process are inverted and implemented in reverse order, as shown in Figure 4.5.

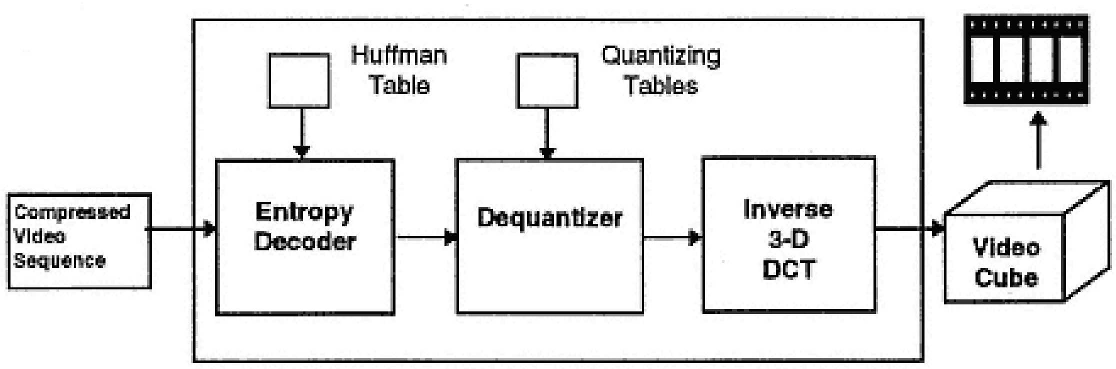
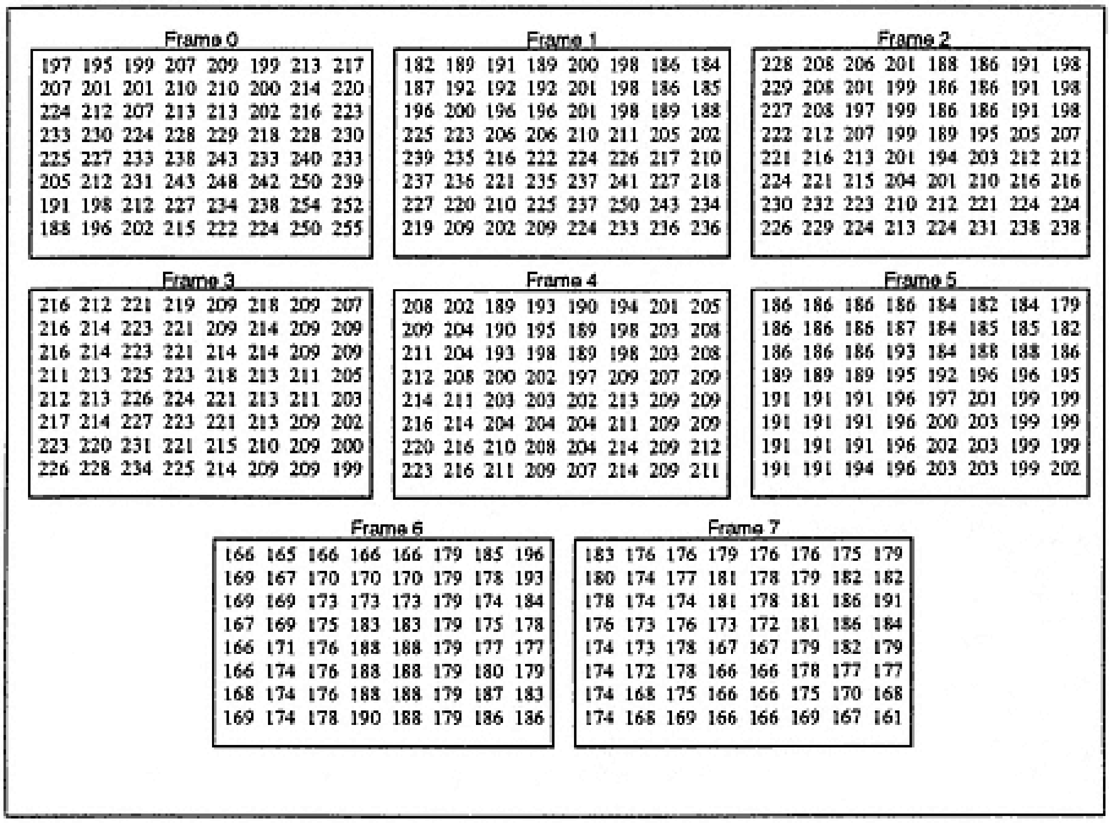


Figure 4.5
Block diagram of the XYZ decompression algorithm.



Frame 0

197	195	199	207	209	199	213	217
207	201	201	210	210	200	214	220
224	212	207	213	213	202	216	223
233	230	224	228	229	218	228	230
225	227	233	238	243	233	240	233
205	212	231	243	248	242	250	239
191	198	212	227	234	238	254	252
188	196	202	215	222	224	250	255

Frame 1

182	189	191	189	200	198	186	184
187	192	192	192	201	198	186	185
196	200	196	196	201	198	189	188
225	223	206	206	210	211	205	202
239	235	216	222	224	226	217	210
237	236	221	235	237	241	227	218
227	220	210	225	237	250	243	234
219	209	202	209	224	233	236	236

Frame 2

228	208	206	201	188	186	191	198
229	208	201	199	186	186	191	198
227	208	197	199	186	186	191	198
222	212	207	199	189	195	205	207
221	216	213	201	194	203	212	212
224	221	215	204	201	210	216	216
230	232	223	210	212	221	224	224
226	229	224	213	224	231	238	238

Frame 3

216	212	221	219	209	218	209	207
216	214	223	221	209	214	209	209
216	214	223	221	214	214	209	209
211	213	225	223	218	213	211	205
212	213	226	224	221	213	211	203
217	214	227	223	221	213	209	202
223	220	231	221	215	210	209	200
226	228	234	225	214	209	209	199

Frame 4

208	202	189	193	190	194	201	205
209	204	190	195	189	198	203	208
211	204	193	198	189	198	203	208
212	208	200	202	197	209	207	209
214	211	203	203	202	213	209	209
216	214	204	204	204	211	209	209
220	216	210	208	204	214	209	212
223	216	211	209	207	214	209	211

Frame 5

186	186	186	186	184	182	184	179
186	186	186	187	184	185	185	182
186	186	186	193	184	188	188	186
189	189	189	195	192	196	196	195
191	191	191	196	197	201	199	199
191	191	191	196	200	203	199	199
191	191	191	196	202	203	199	199
191	191	194	196	203	203	199	202

Frame 6

166	165	166	166	166	179	185	196
169	167	170	170	170	179	178	193
169	169	173	173	173	179	174	184
167	169	175	183	183	179	175	178
166	171	176	188	188	179	177	177
166	174	176	188	188	179	180	179
168	174	176	188	188	179	187	183
169	174	178	190	188	179	186	136

Frame 7

183	176	176	179	176	176	175	179
180	174	177	181	178	179	182	182
178	174	174	181	178	181	186	191
176	173	176	173	172	181	186	184
174	173	178	167	167	179	182	179
174	172	178	166	166	178	177	177
174	168	175	166	166	175	170	168
174	168	169	166	166	169	167	161

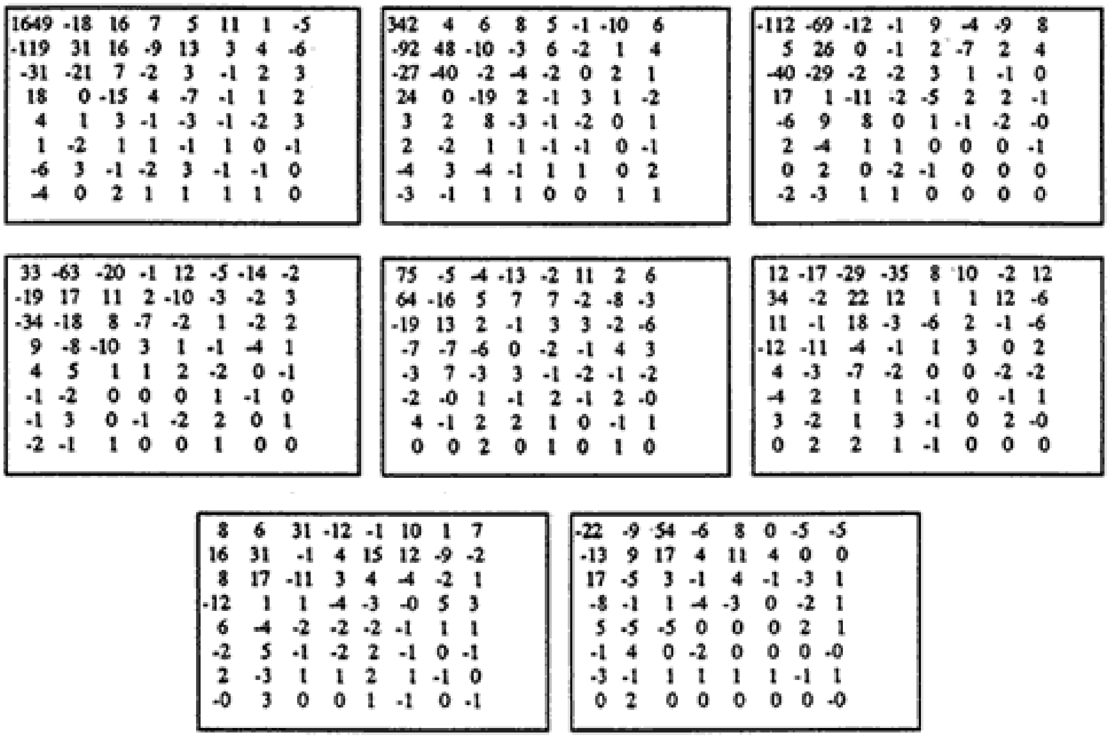
Figure 4.3
An example of encoding an 8x8x8 video cube - original pixels in video cube.

First the compressed data stream is Huffman-decoded. This data stream is now composed of the coding alphabet symbols of run-length and VLC lengths alternated with the VLC representation of the non-zero coefficient. The decoded data is run-length expanded and converted into a stream of quantized coefficients. These quantized coefficients are resequenced into XYZ video cubes of quantized coefficients.

The quantized coefficients are dequantized according to the following equation:

$$F'(u,v,w) = F_q(u,v,w) * Q(u,v,w) \tag{4.4}$$

where F'(u,v,w) is a dequantized coefficient.



1649	-18	16	7	5	11	1	-5
-119	31	16	-9	13	3	4	-6
-31	-21	7	-2	3	-1	2	3
18	0	-15	4	-7	-1	1	2
4	1	3	-1	-3	-1	-2	3
1	-2	1	1	-1	1	0	-1
-6	3	-1	-2	3	-1	-1	0
-4	0	2	1	1	1	1	0

342	4	6	8	5	-1	-10	6
-92	48	-10	-3	6	-2	1	4
-27	-40	-2	-4	-2	0	2	1
24	0	-19	2	-1	3	1	-2
3	2	8	-3	-1	-2	0	1
2	-2	1	1	-1	-1	0	-1
-4	3	-4	-1	1	1	0	2
-3	-1	1	1	0	0	1	1

-112	-69	-12	-1	9	-4	-9	8
5	26	0	-1	2	-7	2	4
-40	-29	-2	-2	3	1	-1	0
17	1	-11	-2	-5	2	2	-1
-6	9	8	0	1	-1	-2	-0
2	-4	1	1	0	0	0	-1
0	2	0	-2	-1	0	0	0
-2	-3	1	1	0	0	0	0

33	-63	-20	-1	12	-5	-14	-2
-19	17	11	2	-10	-3	-2	3
-34	-18	8	-7	-2	1	-2	2
9	-8	-10	3	1	-1	-4	1
4	5	1	1	2	-2	0	-1
-1	-2	0	0	0	1	-1	0
-1	3	0	-1	-2	2	0	1
-2	-1	1	0	0	1	0	0

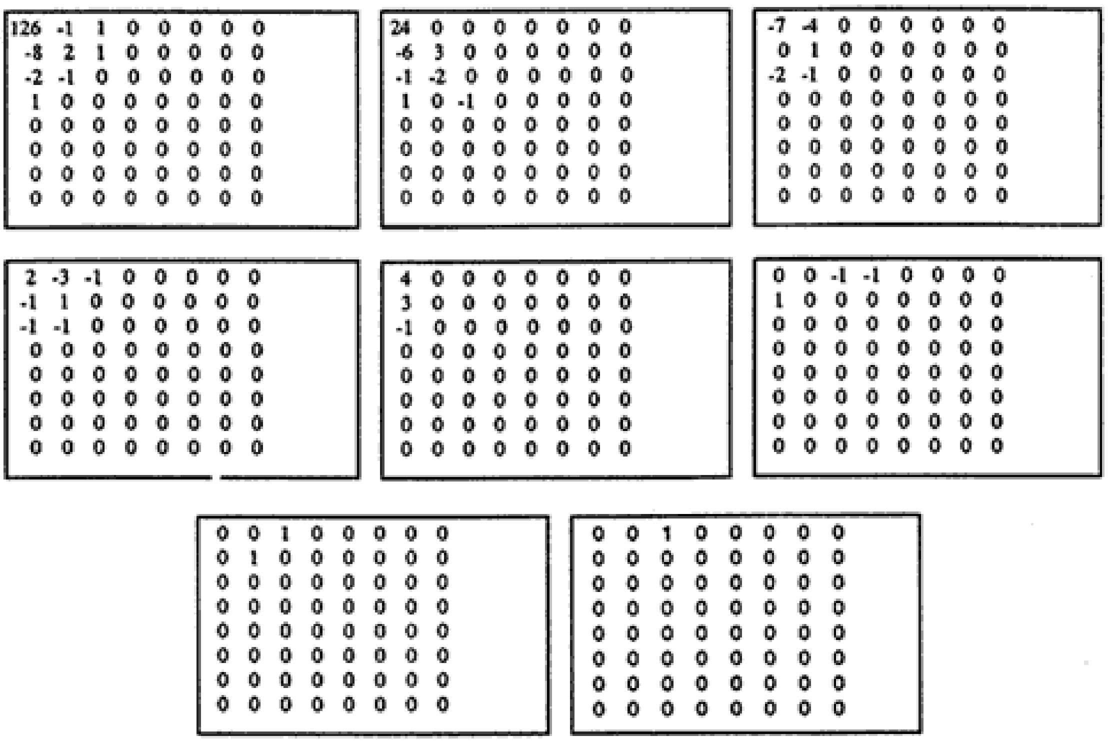
75	-5	-4	-13	-2	11	2	6
64	-16	5	7	7	-2	-8	-3
-19	13	2	-1	3	3	-2	-6
-7	-7	-6	0	-2	-1	4	3
-3	7	-3	3	-1	-2	-1	-2
-2	-0	1	-1	2	-1	2	-0
4	-1	2	2	1	0	-1	1
0	0	2	0	1	0	1	0

12	-17	-29	-35	8	10	-2	12
34	-2	22	12	1	1	12	-6
11	-1	18	-3	-6	2	-1	-6
-12	-11	-4	-1	1	3	0	2
4	-3	-7	-2	0	0	-2	-2
-4	2	1	1	-1	0	-1	1
3	-2	1	3	-1	0	2	-0
0	2	2	1	-1	0	0	0

8	6	31	-12	-1	10	1	7
16	31	-1	4	15	12	-9	-2
8	17	-11	3	4	-4	-2	1
-12	1	1	-4	-3	-0	5	3
6	-4	-2	-2	-2	-1	1	1
-2	5	-1	-2	2	-1	0	-1
2	-3	1	1	2	1	-1	0
-0	3	0	0	1	-1	0	-1

-22	-9	54	-6	8	0	-5	-5
-13	9	17	4	11	4	0	0
17	-5	3	-1	4	-1	-3	1
-8	-1	1	-4	-3	0	-2	1
5	-5	-5	0	0	0	2	1
-1	4	0	-2	0	0	0	-0
-3	-1	1	1	1	1	-1	1
0	2	0	0	0	0	0	-0

(a) DCT coefficients



126	-1	1	0	0	0	0	0
-8	2	1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

24	0	0	0	0	0	0	0
-6	3	0	0	0	0	0	0
-1	-2	0	0	0	0	0	0
1	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

-7	-4	0	0	0	0	0	0
0	1	0	0	0	0	0	0
-2	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

2	-3	-1	0	0	0	0	0
-1	1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

4	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	-1	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) Quantized DCT coefficients

Figure 4.4
An example of encoding an 8x8x8 video cube.
(a) DCT coefficients, after 3D DCT, and (b) quantized DCT coefficients.

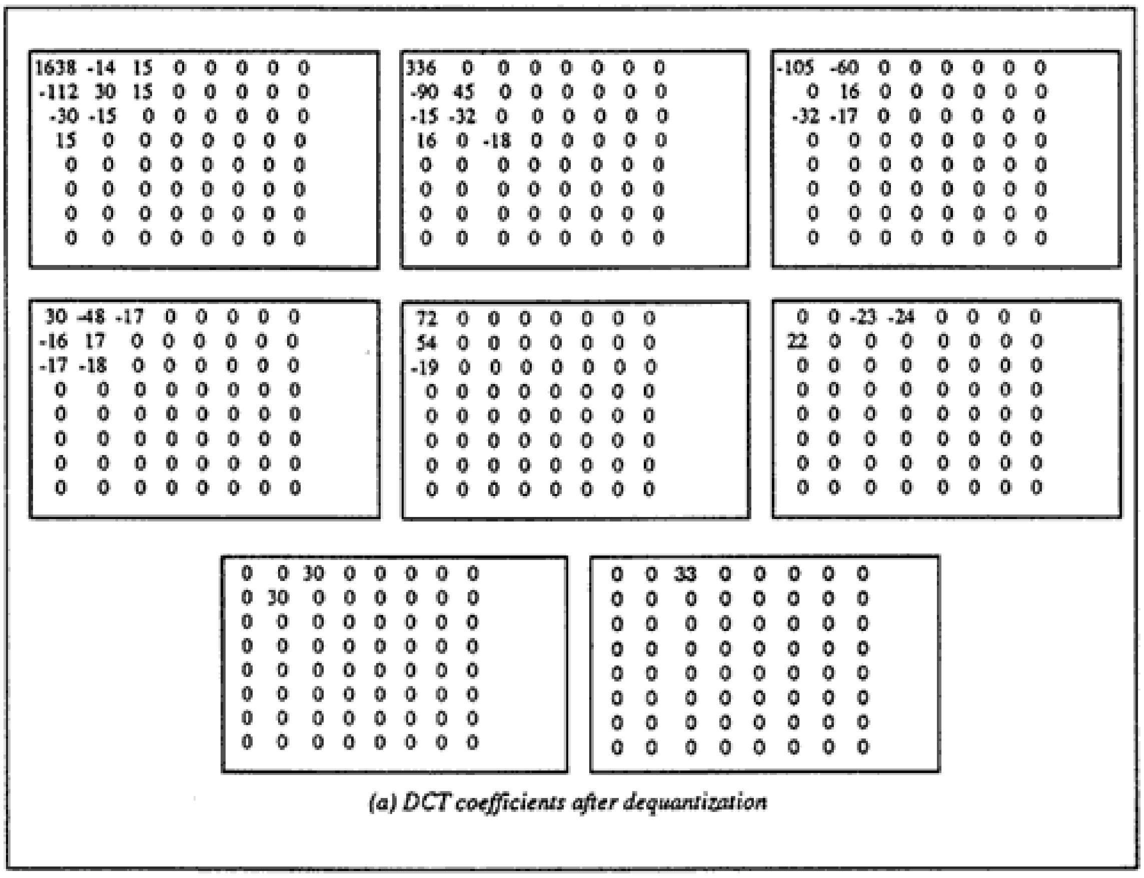
The three-dimensional inverse DCT (3-D IDCT) is implemented on the dequantized coefficients in order to convert video from the frequency domain into the spatial/temporal domain. The 3-D IDCT equation is defined as:

$$f'(x,y,z) = \sum_{u=0}^{7}\sum_{v=0}^{7}\sum_{w=0}^{7} C(u)C(v)C(w) * F'(u,v,w) * \frac{\cos((2x+1)u\pi)}{16}\frac{\cos((2y+1)v\pi)}{16}\frac{\cos((2z+1)w\pi)}{16} \qquad (4.5)$$

where f'(x,y,z) is the value of a pixel in pixel space.

After the pixels have been transformed in spatial/temporal representation, they are shifted back to the original range [0,255]. Finally, the video cubes are reorganized into frames of video data ready for playback.

Figure 4.6 illustrates an example of the XYZ decompression, applied on the same 8x8x8 video cube from Figures 4.3 and 4.4.



1638	-14	15	0	0	0	0	0
-112	30	15	0	0	0	0	0
-30	-15	0	0	0	0	0	0
15	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

336	0	0	0	0	0	0	0
-90	45	0	0	0	0	0	0
-15	-32	0	0	0	0	0	0
16	0	-18	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

-105	-60	0	0	0	0	0	0
0	16	0	0	0	0	0	0
-32	-17	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

30	-48	-17	0	0	0	0	0
-16	17	0	0	0	0	0	0
-17	-18	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

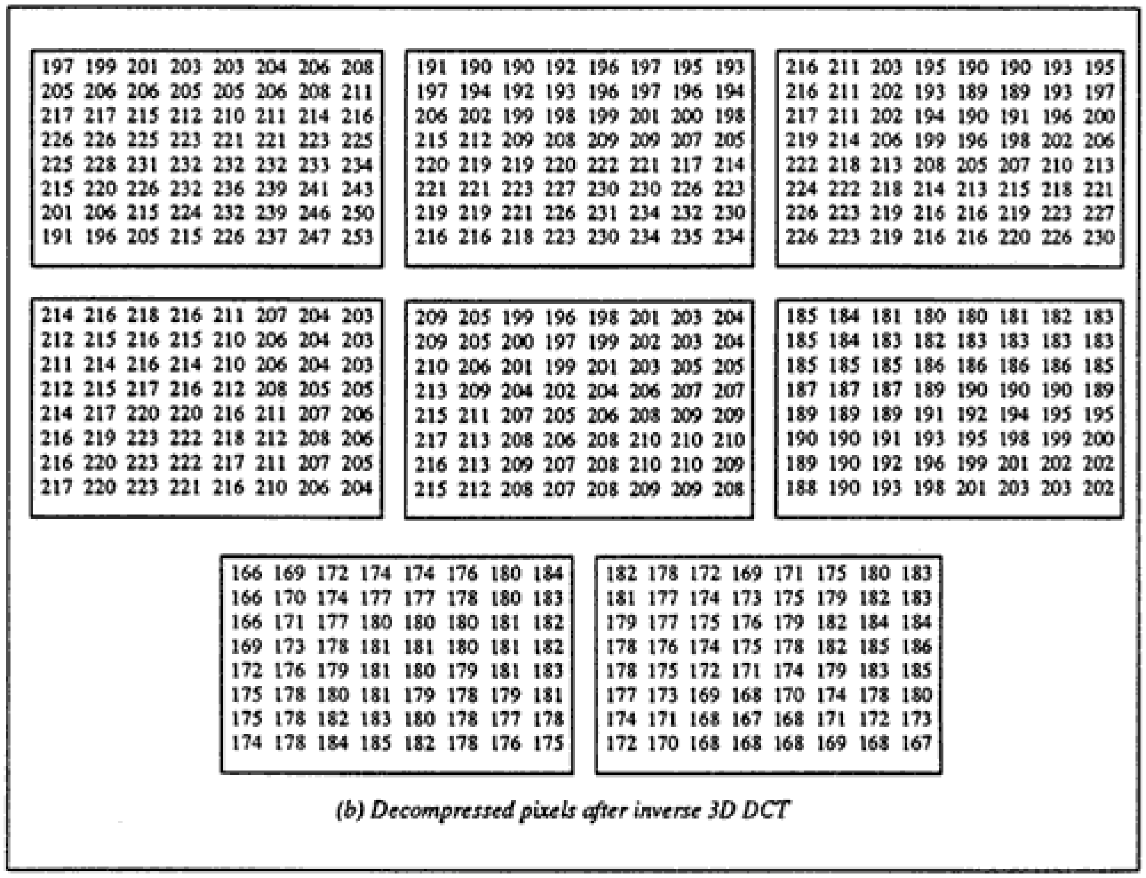
72	0	0	0	0	0	0	0
54	0	0	0	0	0	0	0
-19	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	-23	-24	0	0	0	0
22	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	30	0	0	0	0	0
0	30	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	33	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a) DCT coefficients after dequantization



197	199	201	203	203	204	206	208
205	206	206	205	205	206	208	211
217	217	215	212	210	211	214	216
226	226	225	223	221	221	223	225
225	228	231	232	232	232	233	234
215	220	226	232	236	239	241	243
201	206	215	224	232	239	246	250
191	196	205	215	226	237	247	253

191	190	190	192	196	197	195	193
197	194	192	193	196	197	196	194
206	202	199	198	199	201	200	198
215	212	209	208	209	209	207	205
220	219	219	220	222	221	217	214
221	221	223	227	230	230	226	223
219	219	221	226	231	234	232	230
216	216	218	223	230	234	235	234

216	211	203	195	190	190	193	195
216	211	202	193	189	189	193	197
217	211	202	194	190	191	196	200
219	214	206	199	196	198	202	206
222	218	213	208	205	207	210	213
224	222	218	214	213	215	218	221
226	223	219	216	216	219	223	227
226	223	219	216	216	220	226	230

214	216	218	216	211	207	204	203
212	215	216	215	210	206	204	203
211	214	216	214	210	206	204	203
212	215	217	216	212	208	205	205
214	217	220	220	216	211	207	206
216	219	223	222	218	212	208	206
216	220	223	222	217	211	207	205
217	220	223	221	216	210	206	204

209	205	199	196	198	201	203	204
209	205	200	197	199	202	203	204
210	206	201	199	201	203	205	205
213	209	204	202	204	206	207	207
215	211	207	205	206	208	209	209
217	213	208	206	208	210	210	210
216	213	209	207	208	210	210	209
215	212	208	207	208	209	209	208

185	184	181	180	180	181	182	183
185	184	183	182	183	183	183	183
185	185	185	186	186	186	186	185
187	187	187	189	190	190	190	189
189	189	189	191	192	194	195	195
190	190	191	193	195	198	199	200
189	190	192	196	199	201	202	202
188	190	193	198	201	203	203	202

166	169	172	174	174	176	180	184
166	170	174	177	177	178	180	183
166	171	177	180	180	180	181	182
169	173	178	181	181	180	181	182
172	176	179	181	180	179	181	183
175	178	180	181	179	178	179	181
175	178	182	183	180	178	177	178
174	178	184	185	182	178	176	175

182	178	172	169	171	175	180	183
181	177	174	173	175	179	182	183
179	177	175	176	179	182	184	184
178	176	174	175	178	182	185	186
178	175	172	171	174	179	183	185
177	173	169	168	170	174	178	180
174	171	168	167	168	171	172	173
172	170	168	168	168	169	168	167

(b) Decompressed pixels after inverse 3D DCT

Figure 4.6
An example of decoding the 8x8x8 video cube from Figure 4.3.
(a) DCT coefficients after dequantization,
(b) Decompressed pixels after inverse 3D DCT.

5—
Discrete Cosine Transform

5.1—
Behavior of the DCT

Moving continuous tone images are represented as a sequence of "frames". A frame is a two-dimensional array of pixel values in one "plane" for black and white images, or more planes for color images. We model the signal being sampled (a sequence of pixel values forming a row, column, or time-varying sequence) as a random variable with a mean of zero. The probability distribution, shown in Figure 5.1, of pixel x_1 given the value of pixel x_0 has been shown empirically to be an exponential (laplacian) distribution [RY90]:

$$P(X = x_1 - x_0) = \frac{e^{-\lambda|x_1 - x_0|}}{2\lambda} \tag{5.1}$$

Intuitively, this means that if pixel x_0 is red, there is a great likelihood that pixel x_1 is red. This notion is expressed with the probabilistic notion of covariance. N samples of the signal are considered, forming the sample vector **x**. Exponential distributions form a stationary process, where the random function may be described by its auto-covariance matrix **A**, where:

$$\mathbf{A} = \mu\left(\mathbf{x} * \mathbf{x}^T\right) \tag{5.2}$$

For an exponential distribution [FA90]:

$$A_{ij} = \rho^{|i-j|} \tag{5.3}$$

where 0<=r<=1 is the correlation between samples (r is a function of 1).

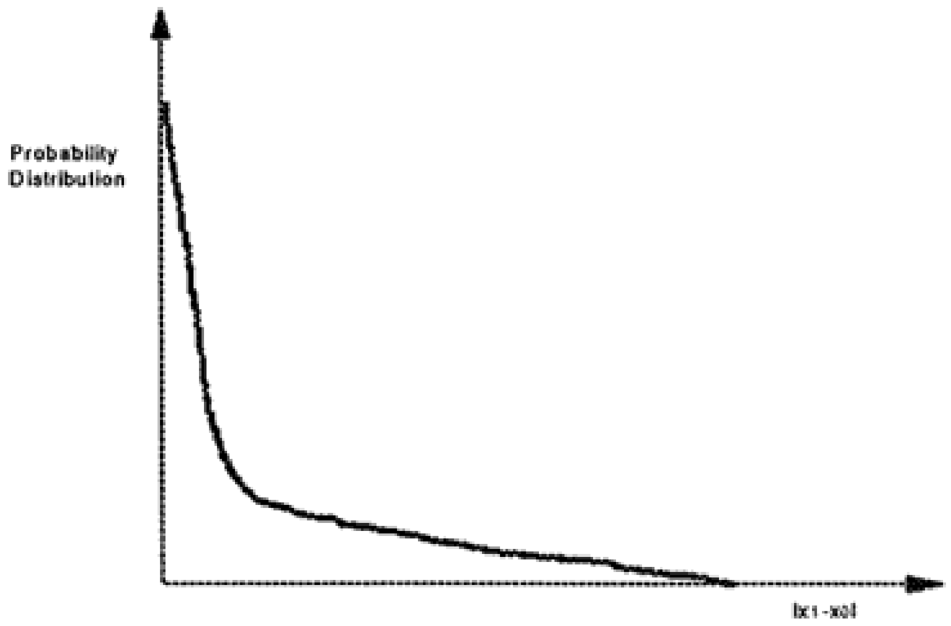


Figure 5.1
Distribution of pixel values in continuous tone images.

The Karhunen-Loeve Transform transforms the basis set in the sample space into a new basis set such that the greatest energy is contained in the fewest number of transform coordinates, and the total representation entropy of the sequence of coefficients in transform space is minimized. In other words, the greatest energy is contained in the earliest coefficients in the basis of the transform space.

Formally, we say that the K-L transform should minimize the Mean Square Error in any truncated representation of the samples in the transform space. Given a random vector **x** = (p_0, p_1, . . . , p_{N-1}), we want to find a set of basis vectors β = (β_0, β_1, . . .β_{N-1}), so we can rewrite x in this basis as x_k = (k_0, k_1, . . .k_{N-1}). We choose β in such way that a truncated representation **t** of **x**, given as t_k = (k_0, k_1, . . .k_{M-1}, 0,0, . . .0) has minimum error:

$$\mathbf{t} = \sum_{i=0,M-1} x_i \beta i \qquad (5.4)$$

It can be shown that this transformation [RY90]:

1. completely decorrelates the signal in transform space,

2. minimizes the mean square error in its truncated representation (guarantees minimum error in compression),

3. concentrates the most energy in the fewest number of coefficients,

4. minimizes the total representation entropy of the sequence.

For highly correlated images (r approaches 1), the basis vectors are exactly the Forward Discrete Cosine Transform (FDCT) [AN74]:

$$S_u = \sum_{i=0,N-1} C_u * s_i * \cos\left(\frac{(2*i+1)*u*\pi}{2*N}\right) \tag{5.5}$$

where:

$$C_u = \begin{cases} \sqrt{\frac{1}{N}}, u = 0 \\ \sqrt{\frac{2}{N}}, u > 0 \end{cases} \tag{5.6}$$

Figures 5.2a and b show energy distribution in pixel space and in DCT space, respectively. In the DCT space the energy is concentrated in few coefficients.

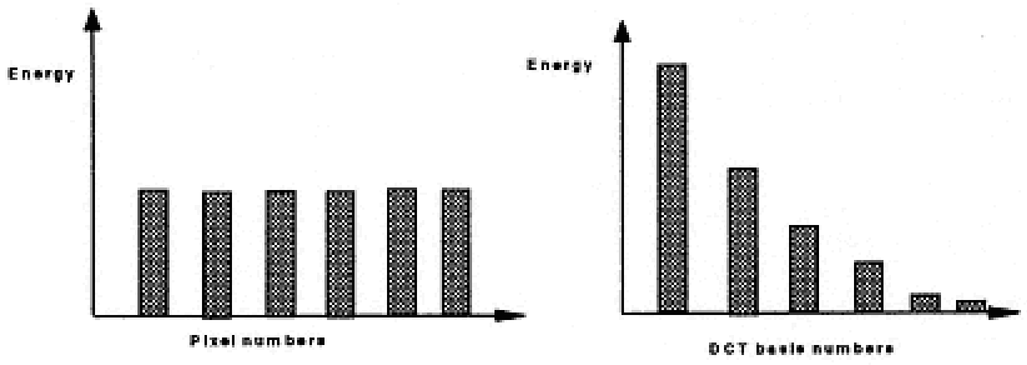


Figure 5.2
Energy distribution: (a) in pixel space energy is equally distributed over all pixels, (b) in DCT space energy is concentrated in few coefficients.

5.2— Fast One-dimensional DCT Techniques

Fast algorithms have been proposed for the one-dimensional Discrete Cosine Transform. They are based on the following techniques [CSF77, RY90, PM90]:

a) Exploiting symmetries of the cosine function

b) Relating the DCT to the Discrete Fourier Transform (DFT)

c) Relating the DCT to other discrete transformations

d) Computation using matrix representation

e) Reducing the DCT to polynomial evaluation.

The first two techniques, based on exploiting symmetries of the cosine transform and relating the DCT to DFT, are the fastest known techniques, and they are described next.

5.2.1— Exploiting Symmetries of the Cosine Function

An opportunity to improve the FDCT engine is seen in the redundancy of the cosine terms in the calculations. The redundancy in the cosine terms is enhanced by expressing all angles in the range 0-p/2 radians, as illustrated in Figure 5.3. The following identities are used to convert angles in the range p/2 - 2p:

$$\cos\left(\pi/2 + \theta\right) = -\cos\left(\pi/2 - \theta\right), 0 < \theta < \pi/2 \qquad (5.7)$$

$$\cos(\pi + \theta) = -\cos(\theta), -\pi < \theta < \pi \qquad (5.8)$$

These symmetries are used to reduce the number of calculations needed to calculate the 1-D DCT. For the 8 point DCT, the number of multiplications is reduced from 64 to 22 [PM93].

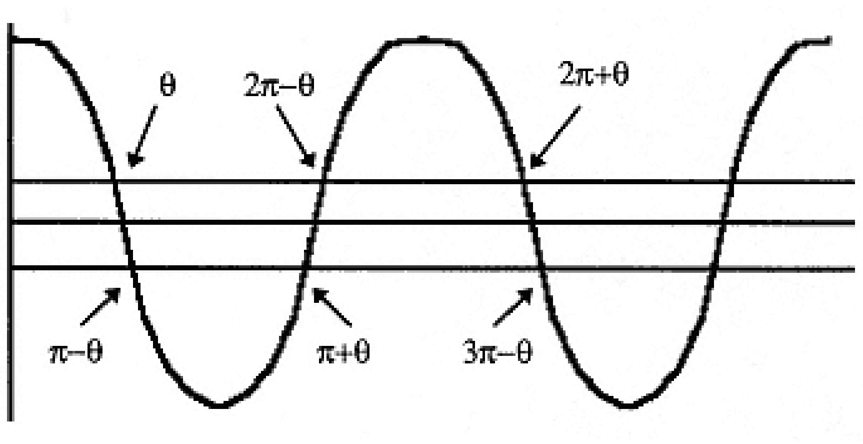


Figure 5.3
Symmetries in the cosine transform can reduce
the number of calculations for 1-D DCT.

Another symmetry can be used to further reduce the number of multiplications. This symmetry is based on the introduction of the rotation:

$$\begin{aligned} X &= x*\cos(\theta)+y*\sin(\theta), \\ Y &= -x\sin(\theta)+y*\cos(\theta) \end{aligned} \tag{5.9}$$

We rewrite the transformation into a form that will require only three multiplications:

$$\begin{aligned} X &= (x+y)*\cos(\theta)+y*(\sin(\theta)-\cos(\theta)), \\ Y &= (x+y)*\cos(\theta)-x*(\sin(\theta)+\cos(\theta)) \end{aligned} \tag{5.10}$$

By applying the rotation operation, the number of multiplications for the 8-point DCT is reduced to 20.

Yet another trigonometric identity, sum/difference of two angles, yields good results:

$$\cos\left(\frac{i*\pi}{16}\right)-\cos\left(\frac{j*\pi}{16}\right)=-2*\sin\left(\frac{(i+j)*\pi}{32}\right)*\sin\left(\frac{(i-j)*\pi}{32}\right),$$
$$\cos\left(\frac{i*\pi}{16}\right)+\cos\left(\frac{j*\pi}{16}\right)=2*\cos\left(\frac{(i+j)*\pi}{32}\right)*\cos\left(\frac{(i-j)*\pi}{32}\right) \quad (5.11)$$

The 8 point 1-D DCT overhead now becomes 13 multiplications, which is as good as any other known method [PM93].

5.2.2— Fast DCT Based on the Discrete Fourier Transform

The 8-point DCT can be extended to form a 16-point FFT. The 8 points of the pixel space to be transformed are extended to form 16 points by duplicating each of the original points i to form point 15-i.

The real part of the Fourier-transformed points is only a constant factor different from the DCT-transformed points. A careful analysis of the Fourier transform, including exploiting symmetries and trigonometric identities similar to those discussed in 5.2.1, shows that the real part of the Fourier transform may be computed with just five multiplications [CT65, Win78]. The DCT may therefore be computed with 13 multiplications [Fei90].

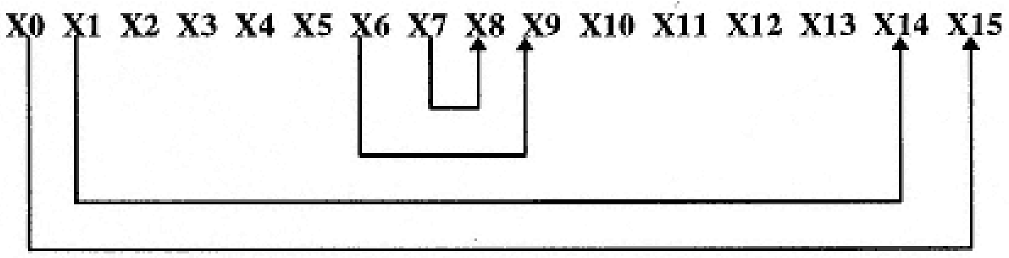


Figure 5.4
Extending 8 points of pixel data to perform a 16 point FFT.

We first extend 8 points of pixel data s(x) to 16 points (see Figure 5.4) in order to compute a 16 point FFT by defining:

$$s[x]=x[15-x] \quad (5.12)$$

The Fast Fourier Transform (FFT) is then computed as:

$$F[u] = \sum_{x=0,15} s[x]e^{-2\pi iux/16} \tag{5.13}$$

which may be rewritten because of (5.12) as:

$$F[u] = \sum_{x=0,7} s[x]e^{-2\pi iux/16} + \sum_{y=8,15} s[y]e^{-2\pi iuy/16} \tag{5.14}$$

We define z=15-y, so F[u] becomes:

$$F[u] = \sum_{x=0,7} s[x]e^{-2\pi iux/16} + \sum_{z=0,7} s[z]e^{-2\pi iu(15-z)/16} \tag{5.15}$$

We can now relabel z as x, and collect like terms. We also observe that:

$$e^{-2\pi iu(15-z)/16} = e^{-2\pi iu(16-1-z)/16} = e^{-2\pi iu(-1-z)/16} = e^{2\pi iu(1+z)/16} \tag{5.16}$$

giving F[u]:

$$F[u] = \sum_{x=0,7} s[x]\left(e^{-2\pi iux/16} + e^{-2\pi iu(x+1)/16}\right) \tag{5.17}$$

Multiplying both sides by $e^{-piu/16}$ gives:

$$e^{-\pi iux/16} * F[u] = \sum_{x=0,7} s[x]\left(e^{-2\pi iu(x+.5)/16} + e^{-2\pi iu(x+.5)/16}\right) \tag{5.18}$$

Since e^{iq} = cos(q) + i*sin(q), (5.18) becomes:

$$e^{-\pi iux/16} * F[u] = \sum_{x=0,7} s[x]\begin{pmatrix} \cos(-2\pi u(x+.5)/16) + \cos(2\pi u(x+.5)/16) + \\ i\sin(-2\pi u(x+.5)/16) + i\sin(2\pi u(x+.5)/16) \end{pmatrix} \tag{5.19}$$

Since cos(-q) = cos(q) and sin(-q) = -sin(q), (5.19) becomes:

$$e^{-\pi iux/16} * F[u] = \sum_{x=0,7} s[x]\cos\left(2\pi u(x+.5)/16\right) \tag{5.20}$$

$$e^{-\pi iux/16} * F[u] = \sum_{x=0,7} s[x]\cos\left((2x+1)\pi u/16\right) \tag{5.21}$$

Equations (5.20) and (5.21) indicate that the DCT coefficients can be derived from the complex FFT coefficients by multiplication by a complex factor. However, closer analysis shows that we can obtain the DCT coefficients using multiplication by real factors only.

If we rewrite F(u) = A(u) + i*B(u), and use e^{iq} = cos(q) + i*sin(q), we can get:

$$\begin{aligned}&\left(\cos\left(-\pi u/16\right) + i*\sin\left(-\pi u/16\right)\right)*(A[u] + i*B[u])\\ &= \sum_{x=0,7} s[x]\cos\left((2x+1)\pi u/16\right)\end{aligned} \tag{5.22}$$

From cos(-q) = cos(q) and sin(-q) = -sin(q), and multiplying, and i*i = -1, eq. (5.22) becomes:

$$\begin{aligned}&A[u]*\cos\left(\pi u/16\right) + i*B[u]*\cos\left(\pi u/16\right)\\ &- i*A[u]*\sin\left(\pi u/16\right) + B[u]*\sin\left(\pi u/16\right)\\ &= \sum_{x=0,7} s[x]\cos\left((2x+1)\pi u/16\right)\end{aligned} \tag{5.23}$$

Collecting real and complex parts gives:

$$\begin{aligned}&A[u]*\cos\left(\pi u/16\right) + B[u]*\sin\left(\pi u/16\right)\\ &= \sum_{x=0,7} s[x]\cos\left((2x+1)\pi u/16\right)\end{aligned} \tag{5.24}$$

$$i*B[u]*\cos\left(\pi u/16\right) - i*A[u]*\sin\left(\pi u/16\right) = 0 \tag{5.25}$$

Solving the complex equation (5.25) for B(u) gives:

$$B[u] = {A[u] * \sin(\pi u/16)} \big/ {\cos(\pi u/16)} \qquad (5.26)$$

Substituting for B(u) into the real equation (5.24) gives:

$$A[u] * \cos(\pi u/16) + {A[u] * \sin(\pi u/16) * \sin(\pi u/16)} \big/ {\cos(\pi u/16)} = \sum_{x=0,7} s[x]\cos((2x+1)\pi u/16) \qquad (5.27)$$

$$A[u] * {\cos^2(\pi u/16)} \big/ {\cos(\pi u/16)} + {A[u] * \sin^2(\pi u/16)} \big/ {\cos(\pi u/16)} = \sum_{x=0,7} s[x]\cos((2x+1)\pi u/16) \qquad (5.28)$$

Since A(u) is the real part of F(u), and $\sin^2(\theta)+\cos^2(\theta)=1$, we can get:

$$\mathrm{Re}(F[u]) \big/ {\cos(\pi u/16)} = \sum_{x=0,7} s[x]\cos((2x+1)\pi u/16) \qquad (5.29)$$

Fast Computation of the Discrete Fourier Transform

Since $e^i = \cos(q) + i*\sin(q)$, we can express the Fourier transform (5.17) as:

$$F[u] = \sum_{x=0,15} s[x] * \cos(-2\pi ux/16) + i * s[x] * \sin(-2iux/16), u = 0,7 \qquad (5.30)$$

In (5.30), s(x) is known to be real. The product i*s(x)*sin(-2piux/16) can only make complex contributions to F(u). Because we are only interested in Re(F(u)), the contribution of complex terms is insignificant.

Since only the cosine terms are significant, we may reduce the expression $e^{-2piux/16}$ according the cosine periodic identities described above. Certain expressions are repeated in the resulting equations, and are computed once only.

Calculation of the significant FFT coefficients can be done with only 5 multiplications, as shown in Figure 5.5. Conversion of these 8 coefficients into the first 8 DCT coefficients requires 8 additional multiplications.

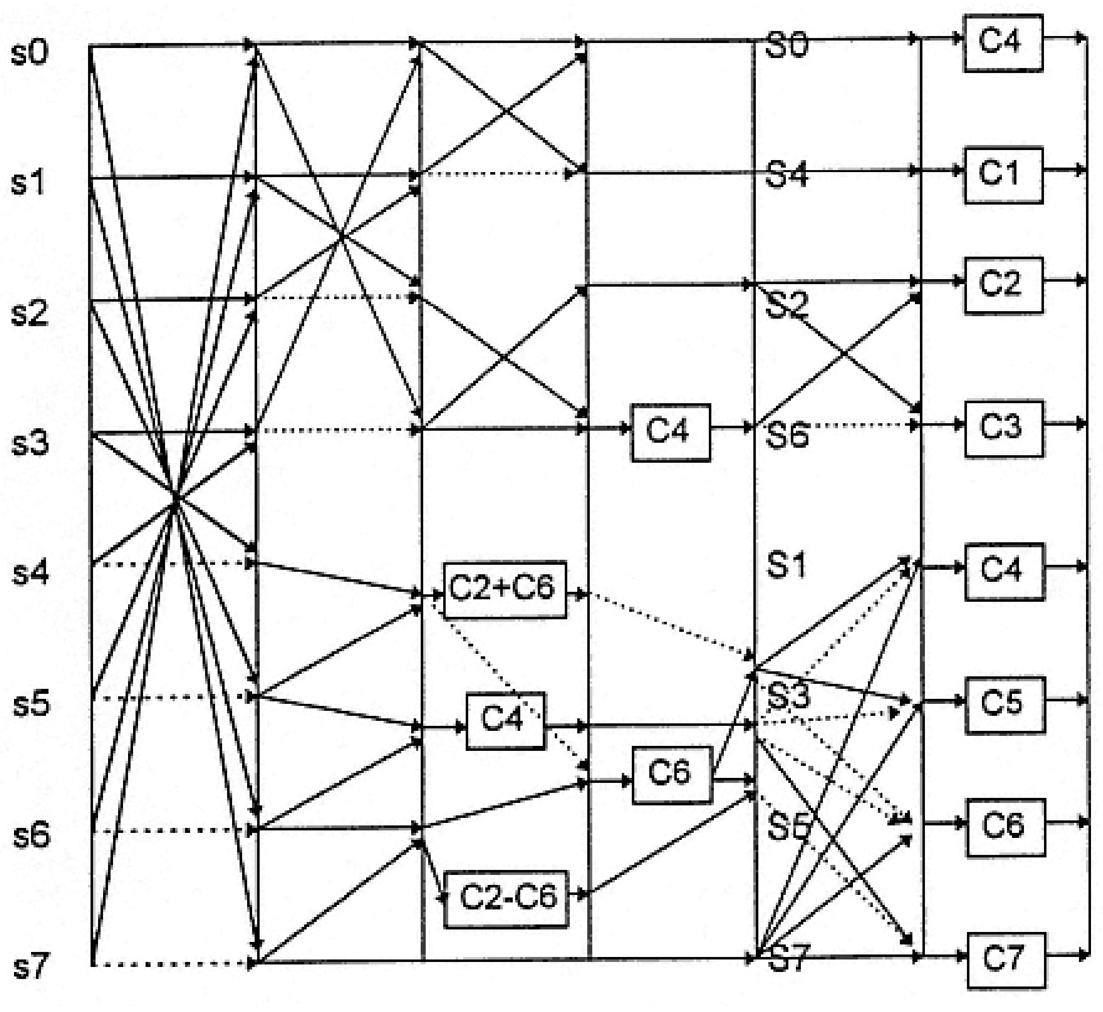


Figure 5.5
Flowgraph showing computation of the fast DCT using FFT. Note that only 5 multiplications are needed before the final stage.

5.3— Two-dimensional DCT Algorithms

Two-dimensional DCT algorithms, proposed in the literature [PM93, CSF77] include:

a) Reduction to 1-D DCT

b) Relating the 2-D DCT to the 2-D DFT

c) Relating the 2-D DCT to other 2-D discrete transformations

d) Block Matrix Decomposition.

In this Section, we describe two most promising techniques, reduction of the 2D DCT to 1D DCTs, and block matrix decomposition of the 2D DCT.

5.3.1— Reduction of the 2D DCT to 1D DCT

Since the 2D DCT is separable, it may be computed by performing eight 1D DCTs on all rows, followed by eight 1D DCTs on the eight columns produced by the first pass of eight DCTs. The number of multiplications required by this approach is then 16 times the number required by a single fast DCT.

In the case of reducing the DCT to FFT, discussed in Section 5.2.2, 8 multiplications are required to scale the results, and only 5 multiplications in the FFT calculation. If the scaling is postponed to the second pass, only 5*8 calculations are required in the first pass, and 5*8 + 8*8 in the second pass. Furthermore, it may be possible to fold the scaling multiplications into a quantization step. In this case, only 80 multiplications are required to perform the 2D DCT[PM93]. Figure 5.6 illustrates this technique.

5.3.2— Calculation of 2D DCT by Sparse Matrix and Cosine Symmetry

In this method, the 1D DCT is expressed as a matrix product, and is factored into 4 sparse matrices, only one of which contains values other than 0, 1, or -1 [CSF77].

When extended to the 2D case by forming the tensor product of the 1D DCTs, the multiplication matrix can be optimized using cosine identities [PM93].

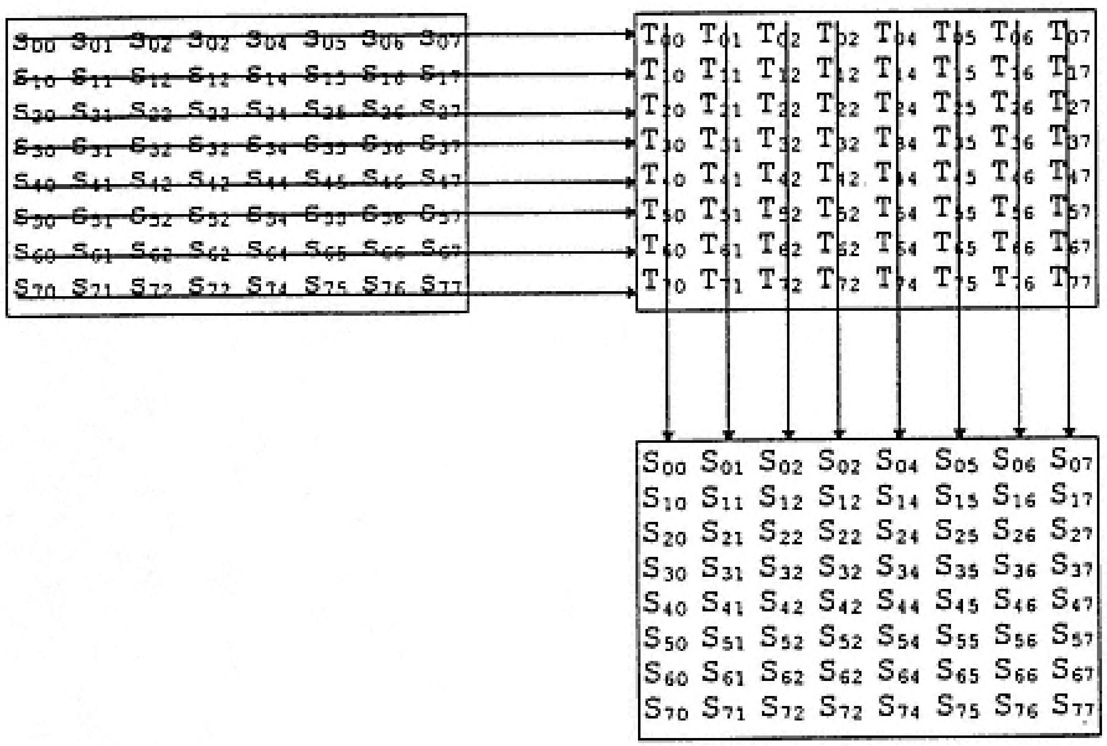


Figure 5.6
The 2D DCT transform is calculated by first performing row-wise
1D DCT. and then column-wise 1D DCT.

This factoring is suggested by the dataflow diagram for the 1D DCT. The accompanying diagram, shown in Figure 5.7, performs the 1D DCT without the final output scaling step.

The FFT transformation may be written as:

$$F = P \otimes R_1 \otimes M \otimes R_2 \qquad (5.31)$$

The flowgraph shows that all multiplications are concentrated in matrix M. In the tensor product FxF the tensor product MxM can be isolated and optimized separately due to the properties of the tensor product operator. Application of the properties of the cosine function result in a reduction of the total number of multiplications to 54 (with six additional multiplications by 1/2, which are considered a shift right by one bit).

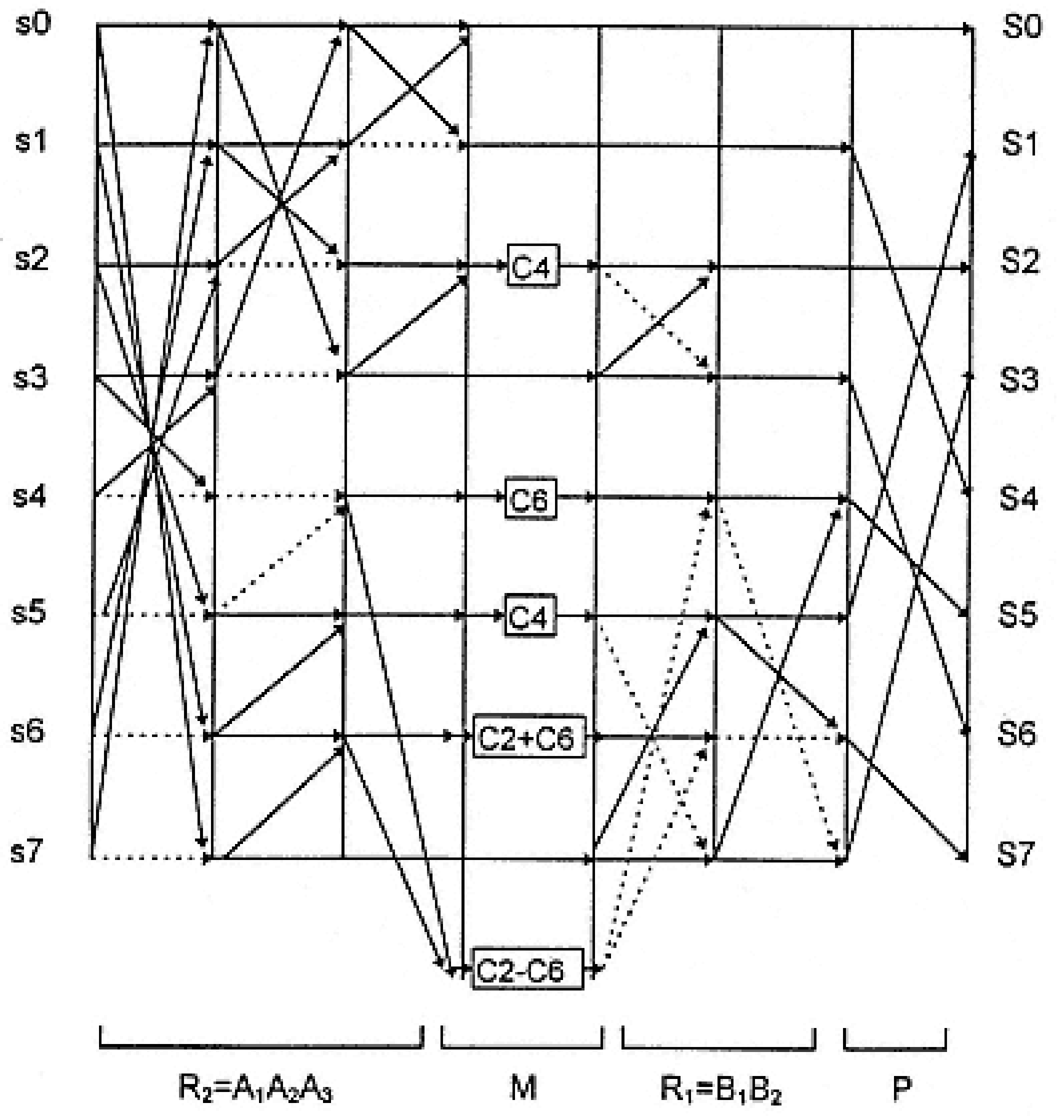


Figure 5.7
Flowgraph for scaled 1D DCT based on FFT.

5.3.3—
Comparison of 2D DCT Techniques

Table 5.1 compares the complexity of 8x8 2D DCT techniques, presented in this Section. Fast FFT-based 2D DCT algorithm is the fastest known algorithm requiring 462 multiplications and 60 shift operations.

Table 5.1 Comparison of complexity of various 2D DCT techniques.

ALGORITHM	Multiplications	Additions	Characteristics
Fast symmetry-based 1D DCT	13X16=208	29X16=464	Apply the DCT column-wise on 8 rows, then row-wise on 8 columns.
Fast FFT-based 1D DCT, not quantized	5X16+64=144	29X16=464	Also apply column-wise, then row-wise. Precompute scaling multiplications and apply once.
Fast FFT-based 1D DCT	5X16=80	29X16=464	Fold output scaling into quantization. Fastest known 1D algorithm, and simple to implement (8 equations).
Fast matrix decomposed, FFT-based 2D DCT, not quantized	54+64=118+6 shifts	462	True 2D algorithm.
Fast matrix decomposed, FFT-based 2D DCT	54 + 6 shifts	462	Fastest known algorithm. However, 2D algorithm requires coding of 64 equations (one per transformed coordinate).

5.4— Inverse DCT Algorithms

Two inverse DCT (IDCT) algorithms are described in this section: (a) forward mapped IDCT, and (b) fast IDCT derived from the DCT.

5.4.1— Forward Mapped IDCT

Ideally, the DCT creates a large number of zero coefficients. The DCT is selected as the preferred transform, because it concentrates energy into the least number of coefficients. In addition, quantization is used to increase the number of zero coefficients by discarding information to which the human eye is least sensitive [GR82], and by introducing the least mean square error [Llo82].

If we consider the IDCT to be an expression of the pixel data in a 64-dimensional vector space (the 8x8 2D DCT space), we can easily calculate the 64 basis vectors. Each basis vector has 64 components, and it is known that no more than 10 of the components are unique, as illustrated in Figure 5.8. Thus no more than 10 multiplications need to be performed for any non-zero component [HM94].

If the quantized values are fixed, all multiplications can be precalculated. In any event, if fewer than about 60 multiplications are required (about 6-10 coefficients, depending on the number of unique values in the required basis vectors), the forward mapped IDCT will outperform other fast IDCT methods.

$$\begin{bmatrix} t_{11} & t_{13} & t_{15} & t_{17} & -t_{17} & -t_{15} & -t_{13} & -t_{11} \\ t_{13} & t_{33} & t_{35} & t_{37} & -t_{37} & -t_{35} & -t_{33} & -t_{13} \\ t_{15} & t_{35} & t_{55} & t_{57} & -t_{57} & -t_{55} & -t_{35} & -t_{15} \\ t_{17} & t_{37} & t_{57} & t_{77} & -t_{77} & -t_{57} & -t_{37} & -t_{17} \\ -t_{17} & -t_{37} & -t_{57} & -t_{77} & t_{77} & t_{57} & t_{37} & t_{17} \\ -t_{15} & -t_{35} & -t_{55} & -t_{57} & t_{57} & t_{55} & -t_{35} & t_{15} \\ -t_{13} & -t_{33} & -t_{35} & -t_{37} & t_{37} & t_{35} & t_{33} & t_{13} \\ -t_{11} & -t_{13} & -t_{15} & -t_{17} & t_{17} & t_{15} & t_{13} & t_{11} \end{bmatrix}$$

Figure 5.8
The basis vector for component (u, v)=(1, 1). This vector has 10 unique values. $t_{ij} = C_i C_j \cos$ (ip/16)cos(jp/16).

5.4.2— Fast IDCT Derived from the DCT

The DCT is an orthogonal and unitary transformation. The inverse of any orthonormal transformation is the transpose of the transformation. The transpose of a flowgraph may be computed by following the graph from right to left [PM93]. Since the number of branches is equal to the number of merges, the computational complexity of the IDCT is exactly the same as that of the DCT. The IDCT for the FFT version of the DCT was calculated using this approach.

5.5— Three-dimensional DCT Algorithms

5.5.1— Applying the DCT to Motion Video

Compression strategies are chosen based upon the statistical behavior of the data to be compressed. Continuous-tone pictures have traditionally been modeled as a stochastic sequence of random variables. An autoregressive model is developed by using a causal minimum variance representation for the stochastic sequence. We express the value of pixel x_n as the sum of a causal prediction x_{pn} and the error term $e_{n:}$

$$x_n = x_{pn} + \varepsilon_n \quad (5.32)$$

The prediction term is chosen to minimize the error term e_n. The minimum occurs when x_{pn} is the conditional mean:

$$x_{pn} = E\left(x_n \middle| x_{n-i}\right), i = 1,2,\cdots n \quad (5.33)$$

If the x_n process is Gaussian, the predictor is linear. We assume a first-order autoregessive model, where r is the one-step correlation coefficient:

$$x_{pn} = \rho * x_{n-1} + \varepsilon_n \quad (5.34)$$

The variance s^2 of e_n is:

$$\sigma^2 = E\left(\varepsilon_n^2\right) \quad (5.35)$$

and s^2 is related to the variance of the pixel distribution s_{x2}:

$$\sigma^2 = \sigma_x^2 * \left(1 - \rho^2\right) \quad (5.36)$$

Statistics have been collected for the correlation ρ in the pixel, scan line, and motion directions for various video effects (motion, pan, break). The results are shown in Table 5.2. These images show extremely high correlations in all three directions, showing that the energy of the pixel representation can be concentrated in each direction with the DCT. This justifies the use of the DCT in all three dimensions (pixel, scan line, motion).

Table 5.2 Measured one-step correlation coefficient ρ for various video clips.

Scene Characterization	**Pixel Direction**	**Scan-Line Direction**	**Frame Direction**
TYPICAL	.9997	.9991	.9978
PAN	.9988	.9970	.9980
ZOOM	.9910	.9893	.9881
BREAK	.9994	.9976	.9970

5.5.2— Development of Fast 3D DCT Algorithms

The heart of both XYZ compression and decompression processes is the three-dimensional DCT. Optimizing this transformation is the key part of developing a practical real-time algorithm [VDG89].

Applying a separable algorithm gives a baseline for evaluating the performance advantages of various 3-D optimizations. Direct separable calculation of the 3-D DCT requires computation of 64 1-D DCTs in each dimension, as illustrated in Figure 5.9. The 1-D DCT requires 64 multiplications and 63 additions. Thus the 3-D transformation will require 3x64x64 multiplications, and 3x64x63 additions : 12,288 multiplications and 12,096 additions.

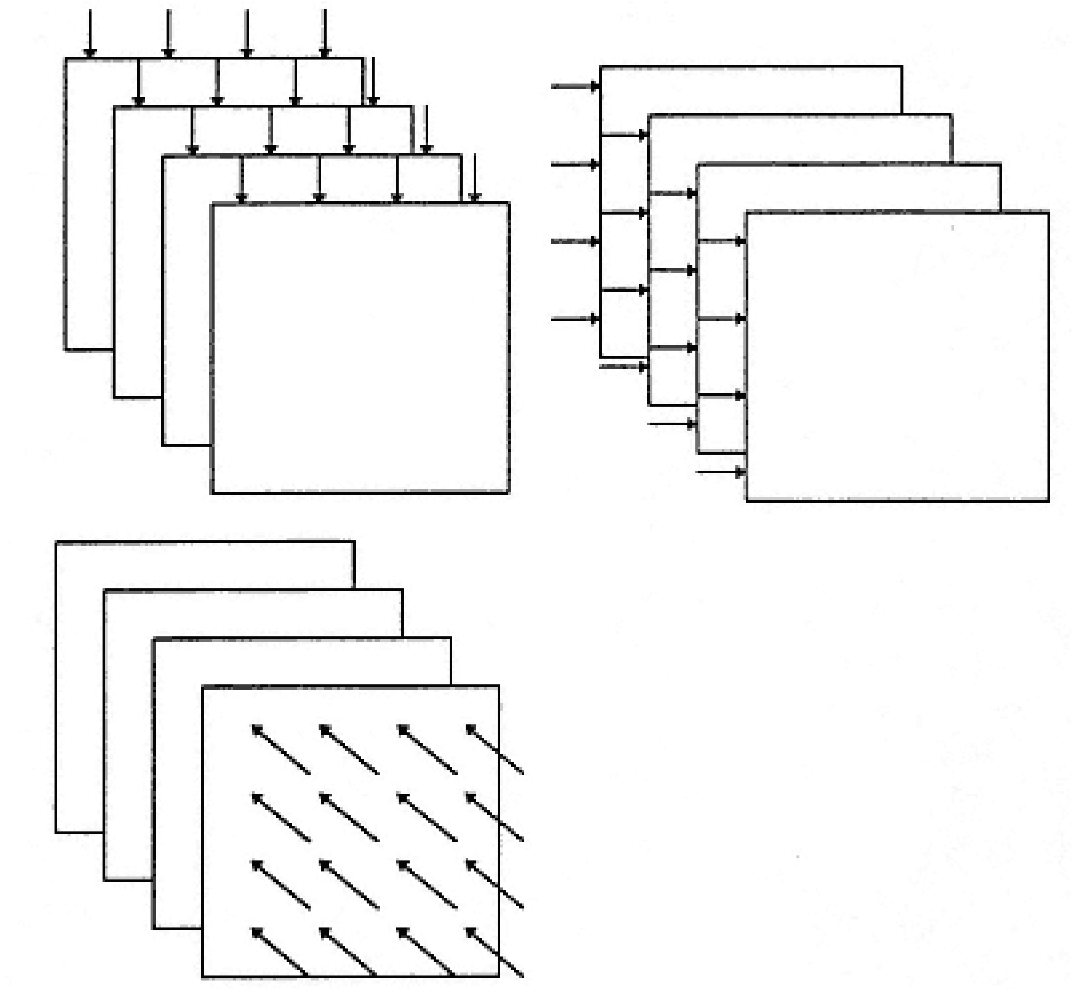


Figure 5.9
Applying the separable 1-D DCT to compute the 3-D DCT (4x4x4 case).

The fastest known Fourier-based 1-D DCT requires 13 multiplications. Applying this fast 1-D DCT separately to the 3-D case will require 3x64x13 multiplications, and 3x64x29 additions: 2,496 multiplications and 5,568 additions. However, the Fourier-based 1-D DCT uses 5 multiplications to compute the (abbreviated) Fourier transform, and 8 multiplications to scale the FFT into the DCT. These scaling multiplications can be folded into the quantizing phase. This results in a total of 3x64x5 multiplications: 960 multiplications and 5,568 additions.

A true two-dimensional algorithm has been developed using the sparse matrix technique, described in Section 5.3.2, that requires 54 multiplications, 6 shifts, and 462 additions. Using this algorithm in 3 dimensions, as shown in Figure 5.10, will require 8x54 + 64x5 multiplications, 8x6 shifts, and 8x462 + 64x29 multiplications, which gives total of 752 multiplications, 48 shifts, and 5,552 additions. However, the estimated size of the code required to implement this algorithm is about eight times the size of the 1D DCT.

The estimated performance savings of a true 3D DCT is about 15% over separable 1D DCT. But the code size is an estimated 50-60 times the size of the 1D DCT. The complexity of various 3-D DCT algorithms is shown in Figure 5.11.

Based on the comparison of the 3D DCT algorithms (Fig. 5.11) and analyzing their implementations, we decided to implement the 3-D DCT as the convolution of the 1-D DCT algorithms.

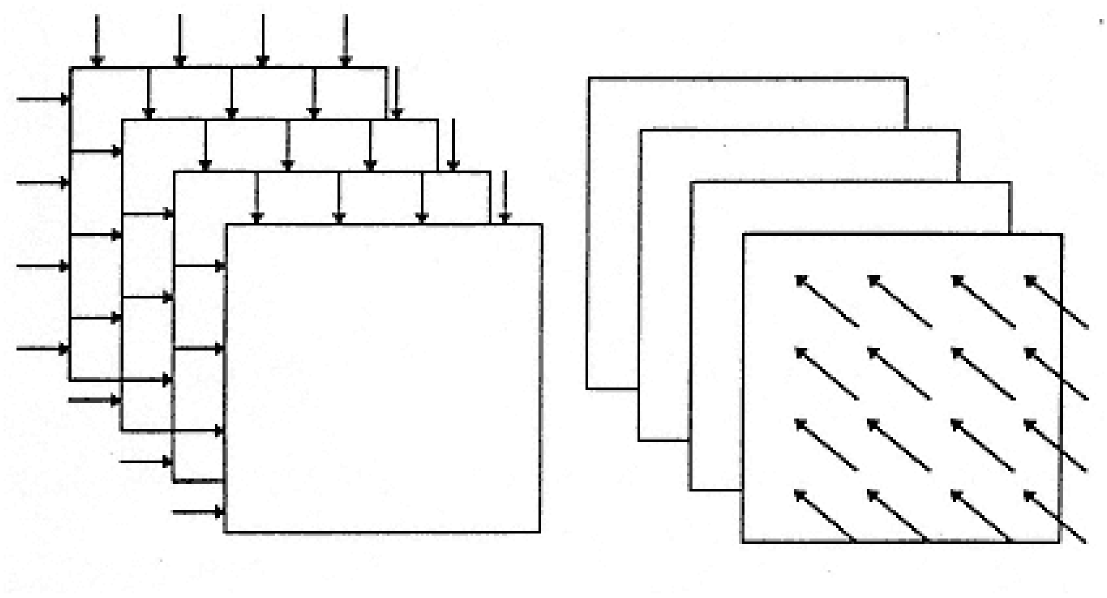


Figure 5.10
Applying the separable 1-D DCT after the
2-D DCT to compute the 3-D DCT (4x4x4 case).

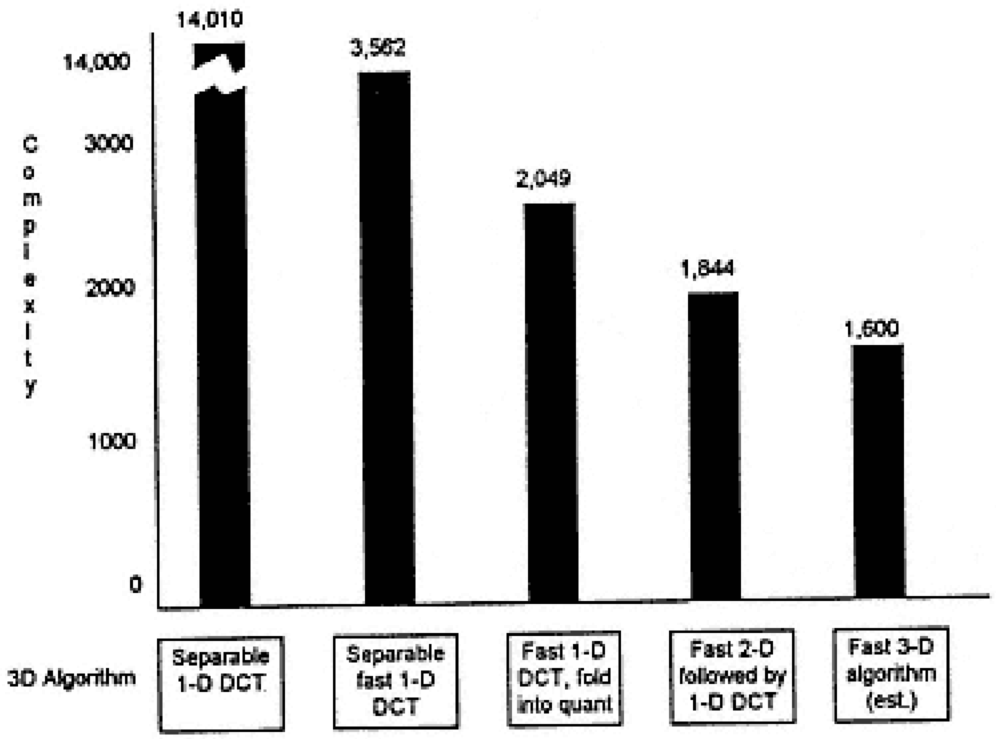


Figure 5.11
The complexity of various 3-D DCT algorithms for 8x8x8 video cube.
10 additions are assumed to be worth 1 multiplication.

6— Quantization

We return to the model of continuous-tone full-motion pictures as a stochastic sequence of random variables, as described in Section 5.5. Pixel values are used as predictors, and thus:

$$x_n = x_{n-1} + \varepsilon_n \quad (6.1)$$

where:

x_n is the pixel value to be predicted,

x_{n-1} is the pixel value to be used as predictor, and

ε_n is the prediction error distribution.

A first-order autoregression model is used to minimize the error distribution e_n, where r is the one-step correlation coefficient:

$$x_{pn} = \rho * x_{n-1} + \varepsilon_n \quad (6.2)$$

The variance s^2 of e_n is calculated from the assumption that its expectation is 0:

$$\sigma^2 = E\left(\varepsilon_n^2\right) \quad (6.3)$$

and s^2 is related to the variance of the pixel distribution $s_{x}2$:

$$\sigma^2 = \sigma_x^2 * \left(1 - \rho^2\right) \quad (6.4)$$

Lloyd and Max have developed the optimal quantizers by minimizing the mean square error of the introduced quantizing noise [Llo82, Max60]. In this case, the optimal quantizers are determined using the variance of the stochastic variables in the transform space to predict the error introduced in pixel space.

6.1— Defining an Invariant Measure of Error

The problem of adaptive quantization requires prediction of the error caused in pixel space by the introduction of error in DCT space. This problem is addressed by recalling that the DCT (and all unitary transformations) are distance preserving (the norm of the sum/difference of two vectors is invariant, Parseval's relation):

$$\sum_{x=0,n-1}\left(s_1[x]-s_2[x]\right)^2 = \sum_{u=0,n-1}\left(S_1[u]-S_2[u]\right)^2 \tag{6.5}$$

where:

s_1, s_2 are expressions of pixel values in pixel space,

S_1, S_2 are expressions of pixel values in DCT space, and

the DCT is a unitary transform (i.e., $DCT^{-1} = DCT^{T}$).

The Mean Square Error (MSE) is defined as:

$$MSE = \sum_{x=0,n-1}\left(s[x]-s_q[x]\right)^2 \tag{6.6}$$

where:

s is the pixel value in pixel space, and

s_q is the pixel value in pixel space after quantization.

Thus Mean Square Error is invariant under the DCT transformation. We define the invariant measure of error Normalized Root Mean Square Error (NRMSE) as:

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{x=0,n-1}\left(s[x]-s_q[x]\right)^2}}{\mu} \tag{6.7}$$

where μ is the mean pixel value (in pixel space).

The foundation can now been laid for the definition of a criterion for measuring quantization error that is invariant under the DCT. Quantization error is the term used to describe the discrepancy introduced when a component of the DCT is recovered from its quantized value. Quantization is defined as the rounded quotient of the DCT component and a quantizing factor:

$$Q(x) = \left\lceil \frac{x}{q} + .5 \right\rceil \tag{6.8}$$

where:

x is the DCT component to be quantized,

q is the quantizing factor, and

Q(x) is the quantized value of x.

Quantizing error is defined as:

$$\begin{aligned} x' &= q * Q(x) \\ E_q &= x - x' \end{aligned} \tag{6.9}$$

where x' is the dequantized value of x.

E_q has a distribution whose mean is 0 if the distribution of x is even (we assume it is Gaussian), and q is odd (if q is even and large, we may assume the mean of E_q is "close" to 0).

Let's choose a typical DCT component x to be quantized by the factor q, and define a "normalized modulus" function that returns a value in the range [-q/2, q/2] as follows:

$$N(x) = \begin{cases} x \bmod q, x \bmod q <= {}^{q}\!/_{2} \\ x \bmod q - q, otherwise \end{cases} \tag{6.10}$$

Then quantzing error becomes:

$$\begin{aligned} x' &= x - N(x), \\ E_q &= N(x) \end{aligned} \qquad (6.11)$$

The average quantizing error is then calculated as:

$$\mu = \frac{\sum_x E_q(x) * P(x)}{N} \qquad (6.12)$$

where:

P(x) is the probability of the value x,

E_q is the error introduced by quantizing the value x by the factor q, and

N is the number of distinct values x.

Since P(x) is even:

$$P(x) = P(-x) \qquad (6.13)$$

Further, it follows from the properties of the modulus function that:

$$N(-x) = -N(x) \qquad (6.14)$$

Thus (6.13) becomes:

$$\begin{aligned} \mu &= \frac{\sum_{x>0} N(x) * P(x) + \sum_{x<0} N(x) * P(x) + N(0) * P(0)}{N}, \\ &= \frac{\sum_{x>0} N(x) * P(x) + \sum_{x>0} N(-x) * P(-x)}{N}, \\ &= \frac{\sum_{x>0} N(x) * P(x) - \sum_{x>0} N(x) * P(x)}{N}, \\ &= 0 \end{aligned} \qquad (6.15)$$

The variance of a probability distribution is calculated as:

$$\sigma^2 = \sum_x E(x-\mu)^2 \tag{6.16}$$

where:

E represents the expectation, or average, function,

x represents the values taken by the probability distribution, and

μ represents the previously calculated expectation, or mean, of the probability distribution.

Since the quantizer error distribution for E_q has mean 0, the variance σ^{2uijk} of Eq can then be calculated as:

$$\sigma^2_{uijk} = \sum_{x=0,N_p,8} \sum_{y=0,N_x,8} E^2_{qijk}[x][y] \tag{6.17}$$

where:

E_{qijk} is the quantizer error distribution in DCT space for component i,,j,k,

σ^{2uijk} is the variance in DCT for component i,j,k, and

x,y range over all 8x8x8 blocks in the group of 8 frames being compressed.

The obtained result estimates quantizer truncation error introduced by representation of a coefficient by an n-bit number for the unit Gaussian distribution as [Sha49]:

$$E_n = 2^{-n} \tag{6.18}$$

where E_n is the normalized truncated quantizer error.

Quantizer error for an unnormalized Gaussian distribution is related to that of a normalized distribution by the standard deviation of the unnormalized distribution:

$$E_{qijk} = \sigma_{uijk} E_n \tag{6.19}$$

Then quantizers introduce error as follows:

$$E_{qijk} = \sigma_{uijk} * \frac{q_{ijk}}{\sigma_{uijk}} \tag{6.20}$$

where:

q_{ijk} is the quantizer factor for component i,j,k, and

$1/\sigma_{ijk}$ is the minimum error caused by representing the error terms with the expected range of the error distribution.

Then the Normalized Root Mean Square Error may be written:

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{x=0,n-1}\left(\sigma_{ijk} * \frac{q_{ijk}}{\sigma_{ijk}}\right)^2}}{\mu} \tag{6.21}$$

6.2—
Calculation of the Transform Variances

In one-dimensional space, the transform variances are calculated as:

$$\sigma^2_{uk} = \sigma^2_{ux} * \left[\mathbf{CAC}^{\mathrm{T}}\right](k,k) \tag{6.22}$$

where:

σ^{2}_{uk} and σ^{2}_{xk} are variances of the *kth* basis in DCT and pixel space, respectively,

C is the DCT transform,

A is the autocorrelation matrix, and thus a function of ρ, and

$\mathbf{CAC}^{\mathrm{T}}(k,k)$ is the k^{th} diagonal of the product $\mathbf{CAC}^{\mathrm{T}}$.

Figure 6.1 shows the variance of DCT transform variables for different correlation coefficients.

The problem of adaptive quantization requires prediction of the error caused in pixel space by the introduction of error in DCT space. This problem is addressed by recalling that the DCT (and all orthonormal transformations) are energy preserving (the sum of the squares of the basis elements is invariant):

$$\sum_{x=0,n-1} s[x]^2 = \sum_{u=0,n-1} S[u]^2 \tag{6.23}$$

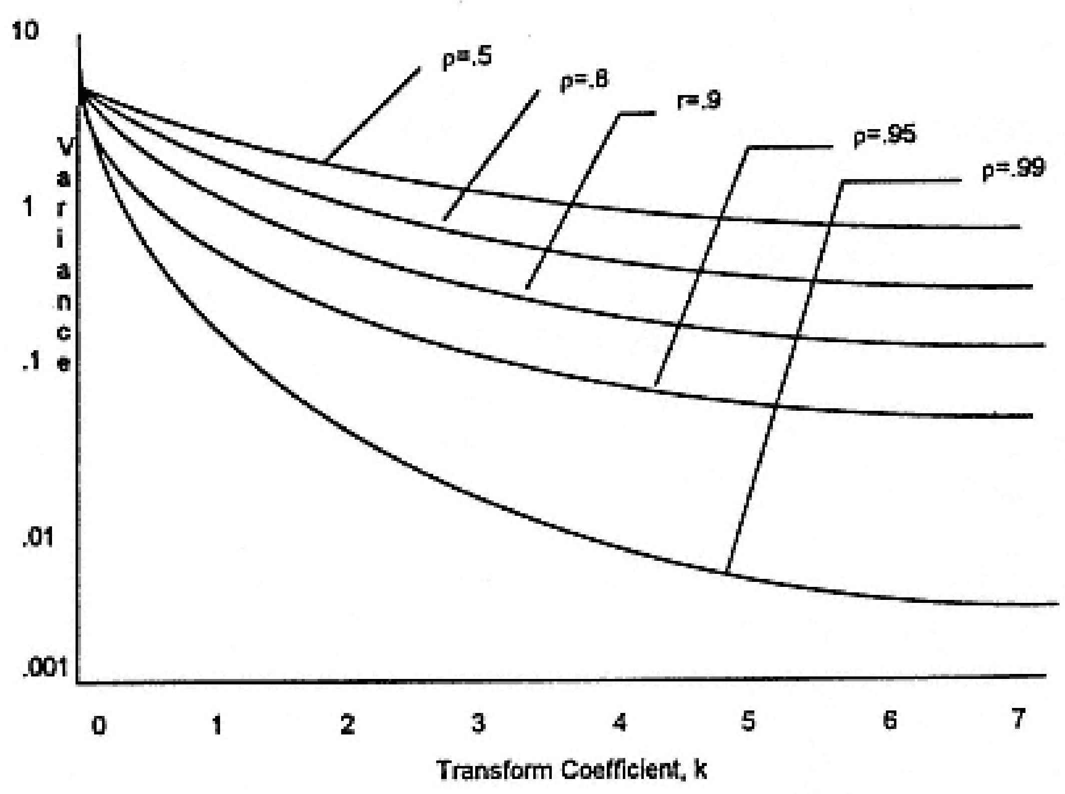


Figure 6.1
Variance of DCT transform variables foe different correlation coefficients.

The variance is preserved under the energy preserving property of the DCT. Since the mean of all AC coefficients of the DCT is already 0, only the DC value has a difficult variance to calculate.

The approach taken to solve this problem in XYZ compression is to average all pixels in the sequence of frames to be compressed. This average is subtracted from each of the pixels in the frames, returning a new pixel distribution whose mean is 0.

Let S_{ijk} represent the pixels to be compressed. The average of all pixels, μ is calculated by:

$$\mu = \frac{\sum_{i=0,N_p} \sum_{j=0,N_s} \sum_{k=0,N_f} s[i][j][k]}{N_p * N_s * N_f} \tag{6.24}$$

where:

N_p is the number of pixels in a scan line (720 for NTSC),

N_s is the number of scan lines (480 for NTSC),

N_f is the number of frames to be compressed (typically 8), and

S[i][j][k] is a typical pixel to be compressed.

Now the transformed pixel distribution s' is calculated from the distribution s as follows:

$$s[i][j][k] = s[i][j][k] - \mu \tag{6.25}$$

The new distribution s' has a mean of 0. The transformed DC coefficient will have a mean of 0:

$$S[0][0][0] = \frac{\sqrt{2}}{4} \sum_{i=0,7} \sum_{j=0,7} \sum_{k=0,7} s'[i][j][k] \tag{6.26}$$

Now the variances of all components can be easily calculated in either pixel space or DCT space, as the mean of each variable is now zero in either space. Thus the variance in pixel space for a typical pixel s'[i][j][k] taken from the 8x8x8 cube of pixels to be compressed is calculated across the entire block of frames:

$$\sigma^2_{xijk} = \sum_{x=0,N_p,8} \sum_{y=0,N_s,8} s'^2[i][j][k] \tag{6.27}$$

where:

σ^{2xijk} is the variance in pixel space for the pixel s'[i][j][k],

x counts every eighth pixel up the N_p number of pixels in the frame, and

y counts every eighth scan line up the Ns number of scan lines in the frame.

Similarly, variances are calculated in DCT space:

$$\sigma^2_{uxijk} = \sum_{x=0,N_p,8} \sum_{y=0,N_s,8} S'^2[i][j][k] \tag{6.28}$$

where σ^{2}_{uijk} is the variance in DCT space for the DCT component S'[i][j][k].

The energy-preserving property of the DCT is used to assert:

$$\sum_{i=0,7}\sum_{j=0,7}\sum_{k=0,7}\sigma^2_{uijk} = \sum_{i=0,7}\sum_{j=0,7}\sum_{k=0,7}\sigma^2_{xijk} \tag{6.29}$$

6.3—
Generating Quantizer Factors

Quantizer distortion is proportional to the variance of the transformed random variable. However, no closed form expression for optimal bit allocation is known. However, it is reasonable to expect that the number of bits allocated will be proportional to the log of the variance of the variable in transform space. Results similar to this conjecture have been reported in [Sha49, HS63, WK68, WJ79]. In this research, we use results given by Shannon [Sha49].

An integer bit allocation algorithm may be used to allocate the expected number of bits needed to represent each DCT component at the desired error level. Once the bit allocation has been completed, the quantizing factors to arrive at the allocated number of bits have be calculated. These quantizers will return the desired error level in pixel space. We begin by initializing all quantizers to their maximum error contribution state:

$$q_{ijk} = \sigma_{ijk} \tag{6.30}$$

Then we initialize:

$$\begin{aligned} n_k &\leftarrow 0 \\ d_k &\leftarrow \sigma_k^2 \\ D &\leftarrow \sum_k \sigma_k^2 \\ D_D &\leftarrow E^2 \end{aligned} \tag{6.31}$$

where:

k ranges through all 512 DCT components,

n_k counts the number of bits allocated to each component,

d_k is the variance of the kth component not yet represented by the bit allocation,

D is the total of d_k,

E is the desired error (input by the agent performing the compression), and D_D is the square of the desired error.

The Normalized Root Mean Square Error at each iteration is calculated as:

$$NRMSE = \frac{\sqrt{\frac{D}{n}}}{\mu} \tag{6.32}$$

The algorithm is iterated while the Normalized Root Mean Square Error exceeds the desired error, i.e., while:

$$D_D < \frac{\sqrt{\frac{D}{n}}}{\mu} \tag{6.33}$$

At each step of the iteration, find the index i for the component which will most minimize the unrepresented variance by being allocated one bit:

$$d_i = \max(d_k) \tag{6.34}$$

Allocate one bit to the ith component, and reduce the variance, and calculate the new quantizer:

$$\begin{aligned} n_i &\leftarrow n_i + 1 \\ D &\leftarrow D - {}^{3*d_i}\!/_4 \\ d_i &\leftarrow {}^{d_i}\!/_4 \\ q_i &= {}^{q_i}\!/_2 \end{aligned} \tag{6.35}$$

When the algorithm converges, the optimal quantizers have been calculated.

6.4— Adding Human Visual Factors

It may be desirable to modify the optimum quantizers by weighing them with factors determined by evaluating human visual acuity. By analogy, JPEG continuous tone picture quantizers are skewed by factors intending to take advantage of the reduced human visual sensitivity to rapid small changes in the picture. It is well-known that the human eye is insensitive to motion changes of 1/60 of a second, and this fact was used to establish the frame rate of television transmission in the United States.

The process of quantizing coefficients in DCT space relies heavily on the decorrelation property of the DCT. Since the DCT decorrelates the random variables in DCT space, each DCT coefficient may be individually processed. Now a different view of DCT coefficients can be developed.

A model of human visual acuity can be developed based on DCT coefficients. Since DCT coefficients can be individually processed, we can also study their individual visibility. Two-dimensional human visual acuity has been modeled by a modulation transfer function of radial frequency in cycles/degree of visual angle subtended.
Figure 6.2 illustrates typical visual acuity curves.
The results of this work have been published in [CR90].

The relative visibility of DCT basis vectors in the spatial dimensions is documented in Table 6.1.

In order to determine the correct curve for perceived amplitude versus frequency of change over time, test samples were played to test audiences. Each of the three-dimensional DCT components was used to build test cases individually for playback at their natural frequency.

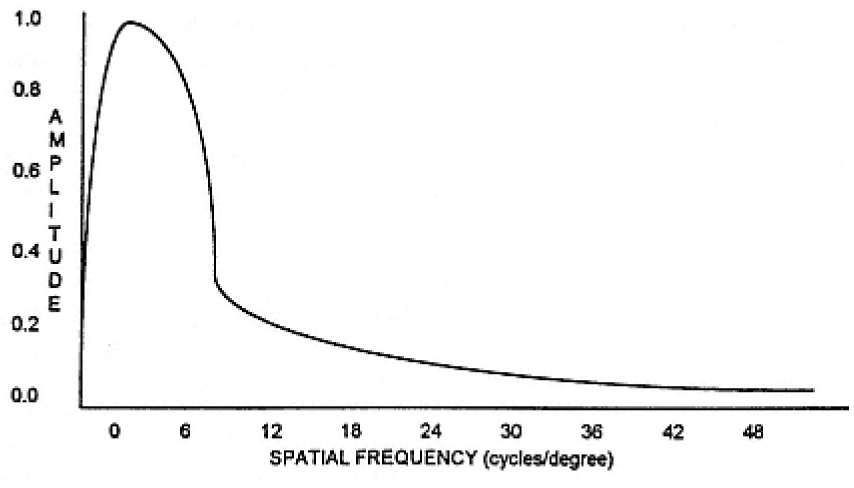


Figure 6.2
Empirically determined human visual actuity for continuous tone pictures.

Table 6.1 Relative visibility of DCT basis vectors in the spatial dimensions.

.4942	1.000	.7023	.3814	.1856	.0849	.0374	.0160
1.000	.4549	.3085	.1706	.0845	.0392	.0174	.0075
.7023	.3085	.2139	.1244	.0645	.0311	.0142	.0063
.3814	.1706	.1244	.0771	.0425	.0215	.0103	.0047
.1856	.0845	.0645	.0425	.0246	.0133	.0067	.0032
.0849	.0392	.0311	.0215	.0133	.0075	.0040	.0020
.0374	.0174	.0142	.0103	.0067	.0040	.0022	.0011
.0160	.0075	.0063	.0047	.0032	.0020	.0011	.0006

The experimental system is composed of two windows whose display is controlled by the viewer, as shown in Figure 6.3. The user selects a DCT component from the 512 available (in the 8x8x8 DCT space), and assigns it an amplitude. An inverse DCT is performed on the selected component to create an 8x8x8 array in pixel space. The array of pixels is organized into 8 frames of 8 scan lines of 8 pixels.

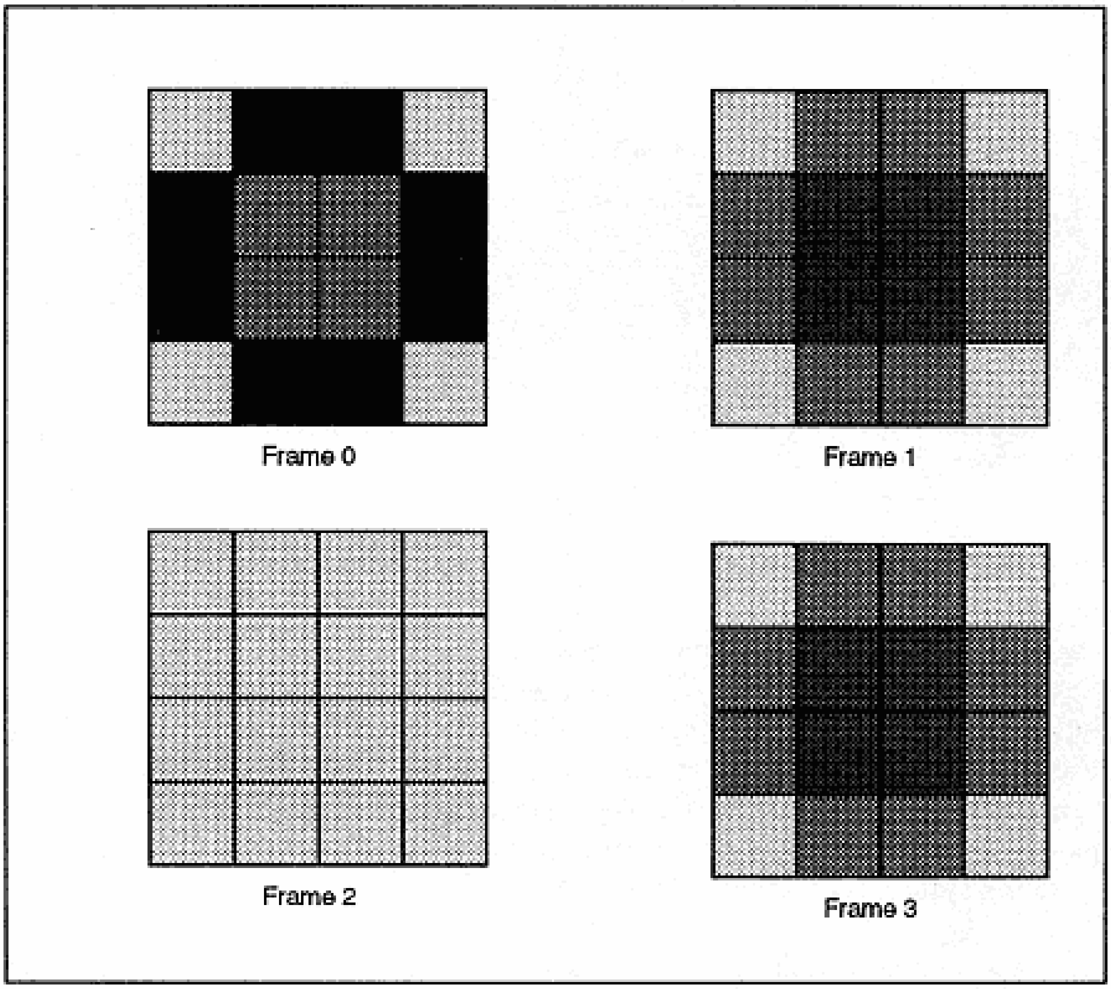


Figure 6.3
Sample test pattern for determining human visual factors.

As the demonstration system, we used the ActionMedia II motion video capture and display adapter. This adapter is composed of a fully programmable microcontroller, 2 million bytes of VRAM, a register-programmable display processor, and a register-programmable capture subsystem, as illustrated in Figure 6.4.

The key concept behind using the ActionMedia II is removing the real-time constraint of video playback from the host processor (in this case, a PC). However, interactive real-time playback is an absolute requirement for determining the relative visibility of the DCT components.

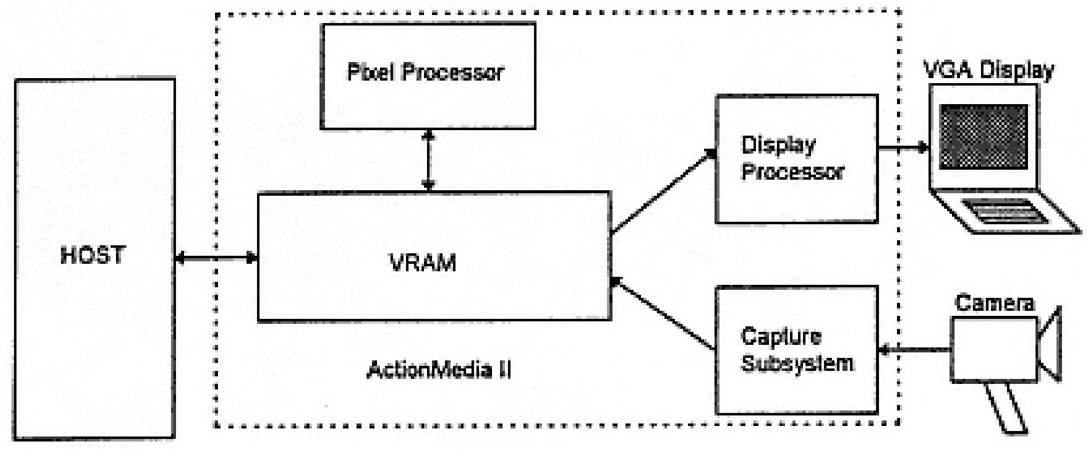


Figure 6.4
Block diagram of the demonstration system based on ActionMedia II.

The real-time playback requirement may be separated from the PC by playing back pre-computed video frames stored on the ActionMedia II. This was accomplished by building a minimal playback system on the ActionMedia II. The host processor is responsible for allocation of resources and scheduling of activity on the ActionMedia II. The pixel processor assumes the responsibility of traversing the queues and executing any work scheduled there. Block diagram of ActionMedia II operating system is presented in Figure 6.5.

The host processor loads the operating system into the pixel processor, and starts the pixel processor running. The host processor loads the copy microcode into RAM on the ActionMedia II. When the user selects a DCT component for display, the host processor calculates the IDCT of the component, then downloads the 8 frames to the ActionMedia II. The pixel processor of the ActionMedia II is capable of displaying the contents of a frame within the vertical block period between frames.

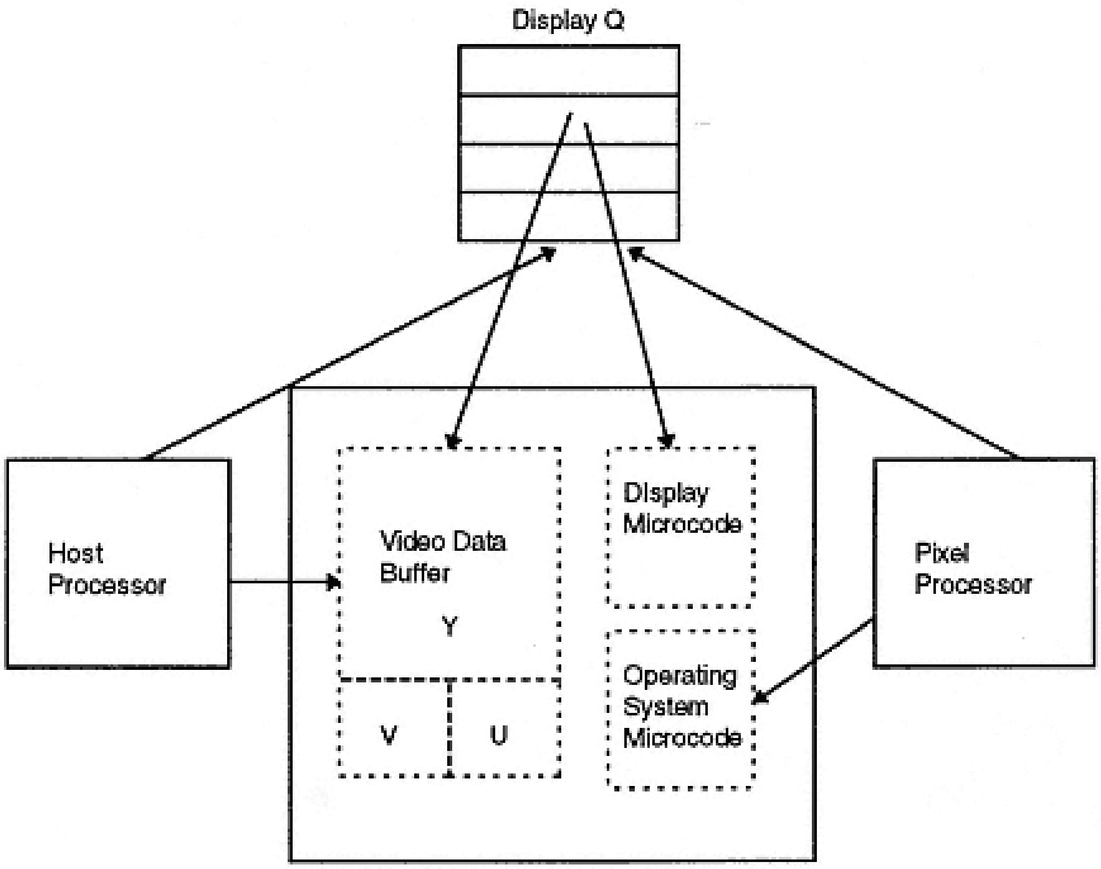


Figure 6.5
Block diagram of ActionMedia II operating system.

The host processor then waits for the vertical blank interval. When the start of the vertical blank period is detected, the host formats a context block with the arguments to the copy routine. The context block is a copy of the internal registers and context of the processor. This organization allows an item of work to be interrupted and saved into the same data structure as that used to load the initial context of the task.

The pixel processor monitors the display queue. When a work item has been detected, the microcode corresponding to that work item is loaded into the host processor. The context block containing the arguments for the microcode are also loaded, and the microcode program begins execution. When complete, execution returns to the microcode operating system.

7—
Entropy Coding

Binary encoding of data is a natural means of representing computational data on modern digital computers. When the values to be encoded are uniformly distributed, this is an space-efficient means of representing the data as well. Information theory gives us several efficient methods of encoding "alphabets" where the likelihood of symbol occurrence varies symbol by symbol. Coding techniques that minimize space in the representation of random sequences of symbols (optimize space used in the representation of symbols based upon the probability the symbol) are known as entropy coding techniques.

There are two popular methods of entropy coding in the literature, Huffman coding and arithmetic coding. Huffman coding represents symbols with words of integer length while arithmetic coding is not limited to integer-length codes. Huffman coding is computationally less expensive to implement and typically gives compression ratios close to those of arithmetic coding. XYZ compression is developed to support Huffman coding [Huf52].

7.1—
Huffman Coding

Consider the problem of encoding the six symbols defined in Table 7.1. The amount of information transferred in a symbol A that occurs with probability p is:

$$I_A = \log_2\left({}^{1}\!/_{p_A} \right) \tag{7.1}$$

where I_A is the number of bits required to express the amount of information conveyed by symbol A, and p_A is the probability that symbol A will occur.

The entropy of a code sequence is the average amount of information contained in each symbol of the sequence:

$$H = \sum_s p(s) * \log_2 \left(\frac{1}{p(s)} \right) \tag{7.2}$$

where H is the entropy of the coding representation, and s ranges through all symbols in the alphabet of symbols.

Table 7.1 Symbols and their associated Huffman code.

Symbol	**Probability**	**Information**	**Code**
A	1/2	1 bit	0
B	1/4	2 bits	10
C	1/16	4 bits	1100
D	1/16	4 bits	1101
E	1/16	4 bits	1110
F	1/16	4 bits	1111

The entropy of the sequence represents the lower bound of the space needed to communicate the information contained in the sequence. A fixed word length of three bits may be used to represent the six symbols in Table 7.1. Using the Huffman coding representation, we get an average code length of 2, which for these probabilities happens also to be the entropy, or lower limit of the average code length:

$$\begin{aligned} H &= \left(\tfrac{1}{2}\right) \times 1 + \left(\tfrac{1}{4}\right) \times 2 + \left(\tfrac{1}{16}\right) \times 4 + \left(\tfrac{1}{16}\right) \times 4 + \left(\tfrac{1}{16}\right) \times 4 + \left(\tfrac{1}{16}\right) \times 4 \\ &= 2 \end{aligned} \tag{7.3}$$

Assignment of Huffman codes is done by developing a Huffman coding tree, as illustrated in Figure 7.1. The tree is developed "left to right" (or bottom to top). Symbols are listed in decreasing order of probability. Iteratively, the two adjacent branches whose sum is least are combined into a new branch, until the tree is completed. The tree is then traversed and labeled.

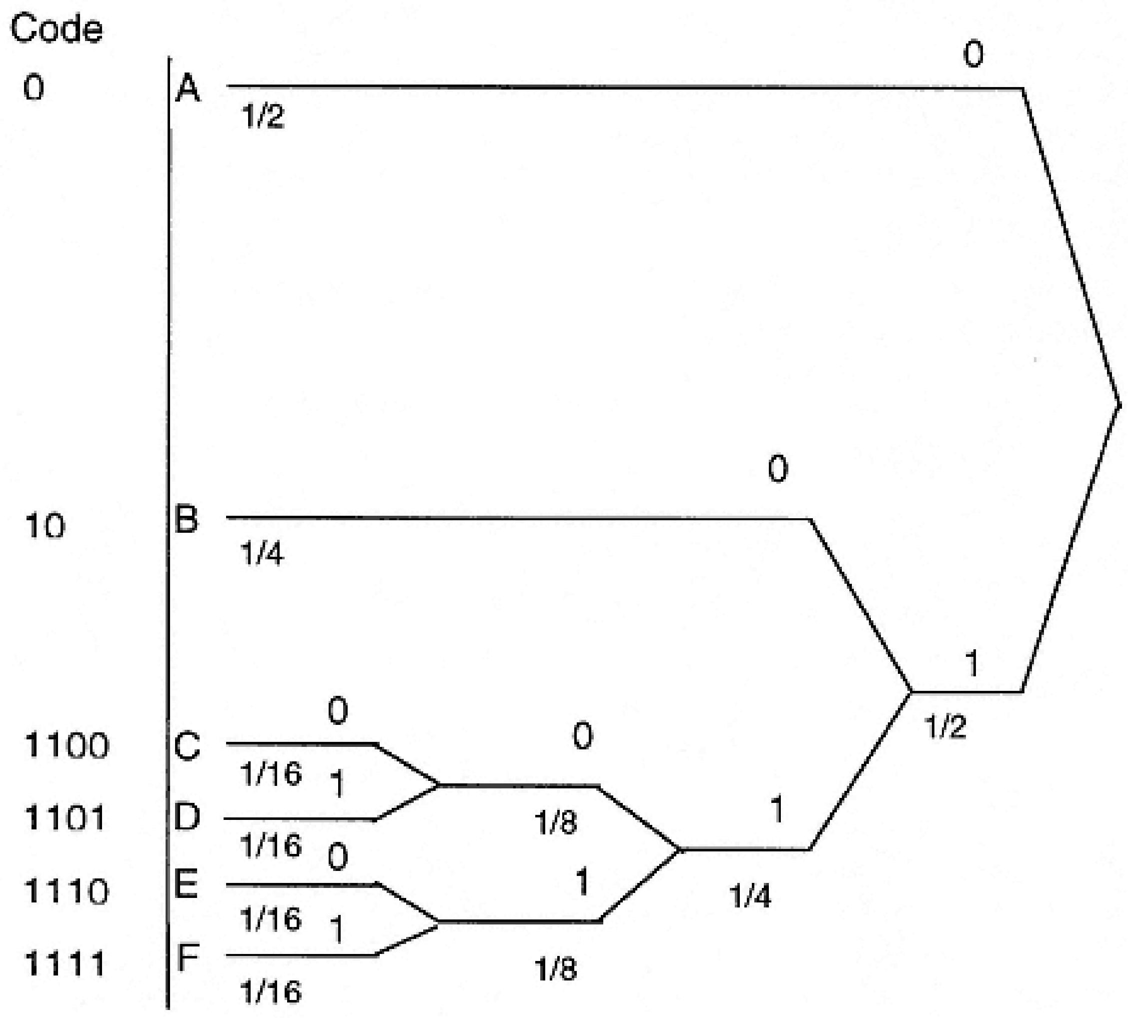


Figure 7.1
Huffmand coding tree - an example.

This algorithm computes Huffman codes for arbitrary sequences of symbols, and will work regardless of their relative probabilities (real cases don't show the simple logarithmic behavior of this example). The algorithm is simple enough (O(NlogN)) to form the kernel of a real-time algorithm for the adaptive entropy coding of images. Huffman coding is used in the JPEG and MPEG specification

7.2— Use of Entropy Coding in JPEG and MPEG

Once Huffman coding has been chosen to represent the data, the compression attained by the system is influenced by the choice of alphabet. Several different alphabets have been defined in the JPEG, MPEG-1, and MPEG-2 standards.

Variable-length coding assigns a length prefix to each word, and follows it with a value code, as shown in Table 7.2. If the values to be represented are small on average, this technique helps reduce the number of bits required to represent them.

Table 7.2 VLC length code and range of encoded values.

Number of bits	**Range of encoded values**
1	-1,1
2	-3..-2, 2..3
3	-7,..-4, 4..7
4	-15..8, 8..15
5	-31, 16, 16..31
6	-63..32, 32..63
7	-127..64, 64..127
8	-255..-128, 128..255
9	-511..-256, 256..511
10	-1023..-512, 512..1023
11	-2047..-1024, 1024..2048
12	-4095..-2047, 2047..4095
13	-8191..-4096, 4096..8191
14	-16383..-8192, 8192..16383
15	-32767..-16384, 16384..32767
16	-65535..-32768, 32768..65535

The concept of run-length coding is also introduced. In its more general form, run-length coding encodes a "run" of consecutive equal values into a length code followed by the value of the run. In the case of JPEG, only zeroes are run-length coded. The run-length code stands alone, and the zero value is implied.

JPEG introduced a modified Huffman coding scheme that introduces many of the concepts later used by the MPEG specifications. JPEG constructs symbols from sequences of DCT components. Two fundamental concepts are used: variable length coding, and run-length coding. JPEG words are formatted into a run length, a VLC length, and a VLC value. The run length and VLC length are combined to form the

JPEG alphabet, given in Table 7.3. Four bits are allocated to each length code, giving 256 symbols in the alphabet.

Table 7.3 The JPEG alphabet.

Run\VLC	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	EOB	01	02	03	04	05	06	07
1	N/A	11	12	13	14	15	16	17
2	N/A	21	22	23	24	25	26	27
3	N/A	31	32	33	34	35	36	37
4	N/A	41	42	43	44	45	46	47
5	N/A	51	52	53	54	55	56	57
6	N/A	61	62	63	64	65	66	67
7	ZRL	71	72	73	74	75	76	77

The JPEG run length and VLC words are combined and treated as symbols to be Huffman encoded. The value 0 is never coded, but is treated as part of a run. Two special symbols are introduced, EOB and ZRL. EOB is the End Of Block symbol, and marks the end of each block, implying a run of zeroes to fill the rest of the block. The ZRL code is the Zero Run Length code, and represents a run of exactly 16 zeroes. This code is used to segment runs of over 16 zeroes.

DCT coefficients are arranged in a simple "zig-zag" order (Table 7.4) intended to concentrate the zero components and increase the run lengths. The variance of each DCT component predicts the probability that component will be 0 (smaller variances predict 0). Higher frequency components tend to have larger variances, and so JPEG orders the components into increasing order of frequency before encoding.

Table 7.4 The JPEG zig-zag order of DCT coefficients.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

MPEG-1 encodes a different alphabet - the run length, VLC length, and the VLC value are combined into a single symbol of the alphabet. Even worse, MPEG-1 allows run lengths of up to 63 zeroes (the maximum possible as there are only 64 DCT coefficients), resulting in a proliferation of symbols. However, DCT components are limited in value to the range [-256..255]. The number of symbols in the alphabet now becomes the number of possible zero runs multiplied by the number of VLC values plus the number of special symbols (EOB). This makes the total number of symbols 32K+1. MPEG-1 zero run length codes are shown in Table 7.5.

Table 7.5 MPEG-1 zero run length codes.

Run Length	**Code**
0	000000
1	000001
2	000010
.	.
62	111110
63	111111

In order to limit the length of code words to 16 bits, MPEG-1 introduces a new symbol, escape (ESC). The ESC symbol triggers the decoder to look for a run-length code followed by a level code. Upon examination of sample video, the most common 222 VLC codes were selected as the alphabet for MPEG-1. All other VLC codes are coded with the escape mechanism. MPEG-1 level codes are shown in Table 7.6.

The MPEG-2 encoding scheme is virtually identical to the MPEG-1 encoding scheme, except that the escape levels are fixed at 12 bits, while those of MPEG are either 8 or 16 bits.

7.3— Adaptive Huffman Coding

This section describes a modification of the JPEG encoding scheme well suited to encoding motion video. The JPEG alphabet is extended with the ESC symbol. The alphabet to be encoded will be composed of all 256 combinations of 4 bit run length and 4 bit VLC codes, ZRL, and EOB.

Table 7.6 MPEG-1 level codes.

LEVEL	**CODE**
-255	1000 0000 0000 0001
-254	1000 0000 0000 0010
.	.
-129	1000 0000 0111 1111
-128	1000 0000 1000 0000
-127	1000 0001
-126	1000 0010
.	.
-2	1111 1110
-1	1111 1111
1	0000 0001
2	0000 0010
.	.
126	0111 1110
127	0111 1111
128	0000 0000 1000 0000
129	0000 0000 1000 1001
.	.
254	0000 0000 1111 1110
255	0000 0000 1111 1111

Statistics for the group of frames to be decoded are gathered. The count of the occurrence of each symbol in the alphabet within the group of frames is collected, and a table of the counts is collected. The count for ESC is set to 0.

The standard Huffman coding procedure, described in Section 7.1, is applied to the data. Two additional tables are constructed, one for the temporary storage of symbols waiting to be coded (representing the root of each branch of the Huffman coding tree), and one for the storage of the symbols that are in the process of being coded (the branches of the tree), as illustrated in Figure 7.2.

Entries in the Encoder Alphabet Table (Table 7.7) whose count is non-zero are copied into the Root Table, setting the #bits field to 0, and the left and right subtrees to none. The procedure to construct the Huffman code tree is then followed.

SYMBOL	COUNT	#BITS	LEFT	RIGHT

Root Table

SYMBOL	COUNT	#BITS	LEFT	RIGHT

Branch Table

Figure 7.2
Format of Root Table and Branch Table

The Root Table is sorted by the count field. The two symbols (branches) with smallest count are removed from the Root Table and added into Branch Table. A new branch entry is created to be added into the Root Table. The left and right fields of the new entry will point to the two branches just added into the Branch Table. The count field of the new entry will be set to the sum of the count fields of the two branches just added to the Branch Table, and the new branch is added to the Root Table in its sorted position. This process is repeated until there is exactly one entry in the Root Table.

The Huffman coding tree is traversed, and the number of bits to be assigned to each code is accumulated. Any symbol that requires more than 12 bits to encode is removed from the Encoder Alphabet Table (by setting its count field to 0), and the value of its count field is added to the count field of the ESC symbol.

A new Huffman Coding tree is built - a new Root Table is built from the Encoder Alphabet Table, a new Huffman Coding Table is built. When this tree is traversed, all entries with less than 12 bits have their codes written to the Encoder Alphabet Table. The Encoder Alphabet Table is now ready for use to encode the frame data.

Encoding is straightforward for entries with less than 12 bits. Entries with over 12 bits are coding using an escape coding procedure - the ESC symbol is written, then the 8 bit run length / variable length field, then the VLC value to be encoded.

Table 7.7 Encoder Alphabet Table

SYMBOL (run/vlc)	**COUNT**	**CODE**
0/1	nnn	
0/2	nnn	
.	.	
0/F	nnn	
1/1	nnn	
1/2	nnn	
.	.	
1/F	nnn	
F/1	nnn	
F/2	nnn	
.	.	
F/F		
ZRL	nnn	
EOB	nnn	
ESC	0	

8— VLSI Architectures for the XYZ Video Codec

8.1— Complexity of Video Compression Algorithms

In video and signal processing applications a measure of algorithmic complexity, which is typically used, is the total number of operations expressed in MOPS (million operations per second) or GOPS (giga operations per second). When designing a new video codec, the first step is to estimate its complexity [BK95, PDG95, F+95, GGV92]. For example, Table 8.1, adapted from [BK95], estimates MOPS (Million Operations Per Second) requirements for H.261 codec using CIF format at 30 frames/s. These estimates were computed assuming fast implementations of DCT and IDCT algorithms and fast 2-D logarithmic search for motion estimation. Total MOPS requirements for both the encoder and decoder are 1,166 MOPS. If the exhaustive search was used for motion estimation, this number would be much higher — 7,550 MOPS.

In [F+92] MOPS requirements for an H.261 codec has been estimated to be about 3,500 MOPS (the encoder required 2,500 MOPS and the decoder and postprocessing about 1,000 MOPS).

MOPS requirements for the MPEG algorithm, for different frame sizes, and different percentages of frames computed with bi-directional motion estimation (B-frames) are reported in [BK95], and presented in Table 8.2.

Table 8.1 MOPS requirements for H.261 codec. CIF format at 30 frames/s. [BK95] Fast DCT and IDCT algorithms and fast search for motion estimation are used.

COMPRESSION	**MOPS**
RGB to YCbCr conversion	27
Motion estimation (exhaustive search for p=8)	608
Inter-/Intraframe coding	40
Loop filtering	55
Pixel prediction	18
2-D DCT	60
Quantization and zig-zag scanning	44
Entropy coding	17
Frame reconstruction	99
TOTAL	**968**

DECOMPRESSION	**MOPS**
Entropy decoder	17
Inverse quantization	9
Inverse DCT	60
Loop filter	55
Prediction	30
YCbCr to RGB conversion	27
TOTAL	**198**

Table 8.2 MOPS requirements for MPEG compressor and decompressor at 30 fps. No preprocessing and postprocessing. No audio and other system-related operations[BK95].

COMPRESSION	**SIF FORMAT**	**CCIR 601**	**HDTV FORMAT**
No B-frames	738	3,020	14,498
20% B-frames	847	3,467	16,645
50% B-frames	1,011	4,138	19,865
70% B-frames	1,120	4,585	22,012

DECOMPRESSION	**SIF FORMAT**	**CCIR 601**	**HDTV FORMAT**
No B-frames	96	395	1,898
20% B-frames	101	415	1,996
50% B-frames	108	446	2,143
70% B-frames	113	466	2,241

Due to the complexity of the motion vector search algorithm, the compression algorithms for both MPEG and H.261/H.263 are much more complex than the decompression algorithm.

The obtained complexity estimates can be used in evaluating various alternatives in the implementation of the video codecs, such as general-purpose RISC microprocessors versus digital signal processors (DSPs), or programmable video processors. Figure 8.1 shows the current trends in computing power for general purpose RISC processors, programmable DSPs, and programmable video processors.

From Tables 8.1 and 8.2 and Figure 8.1, it can be concluded that it is now easily achievable to implement decompression using one or more general-purpose RISC or DSP processors. However, the encoder requirements (even assuming 1000 MOPS of processing power by using a fast motion estimation algorithm) is outside of the complexity of general-purpose processors at this time.

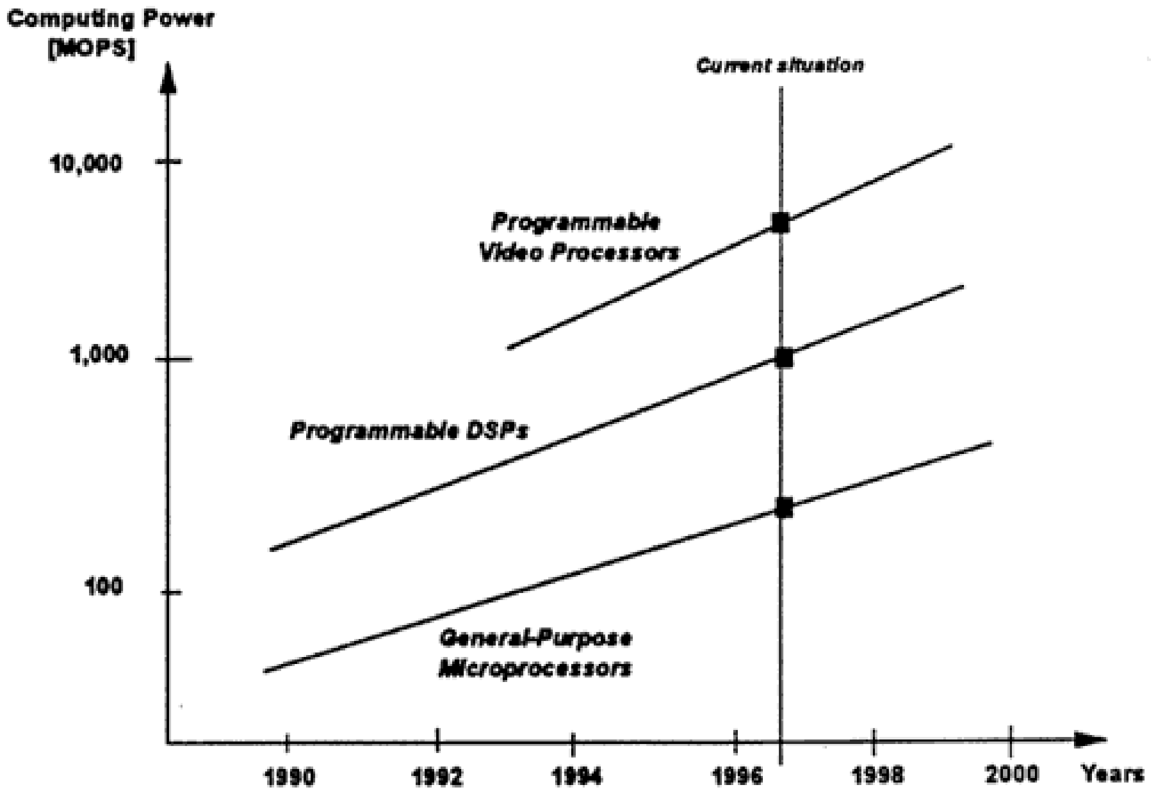


Figure 8.1
Trends in performance of programmable processors.

8.2— From Algorithms to VLSI Architectures

The first step in designing a VLSI architecture for a video codec is to map the specific algorithm into the VLSI architecture. In the mapping process, the main goal is to keep low manufacturing cost and small size of the architecture. Manufacturing cost depends on the number of integrated chips, chip packaging, and silicon area per chip. Using large area silicon, the number of chips can be kept small. On the other hand, production of very large area chips is not economic, due to the defect density.

The required silicon area for VLSI implementation of algorithms is related to required resources such as logic gates, memory, and the interconnect between the modules. The amount of logic depends on the concurrency of operations. A measure for the required concurrency ($N_{con.op}$) can be expressed as:

$$N_{con.\,op} = R_S \bullet N_{op.\,pel} \bullet T_{op} \qquad (8.1)$$

where:

R_s - is the source rate in pels per time unit,

$N_{op.pel}$ - is the number of operations per pel,

T_{op} - is average time for performing an operation.

The number of operations per pel ($N_{op.pel}$) is an average value derived by counting all operations required for performing a specific coding algorithm. The video coding algorithms are periodically defined over a basic interval. In MPEG and H.261/H.263 coding algorithms, this interval is a macro block. Almost all tasks of the coding algorithms are defined on a macro block of 16x16 luminance pels and the associated chrominance pels. For this reason, counting is done over one macro block. For example, for the H.261/H.263 codec the $N_{op.pel}$=1,170 MOPS, when applying 2D logarithmic search algorithm. With the present technology, T_{op} is order of 20 nsec. From Table 3.1 for video source rate, we can obtain the number of concurrent operations for the H.261/H/263 video algorithms ranging from 40 (for SQCIF) to over 1000 (for 16CIF).

The required interconnects between the operation part and the memory highly depends on the access rate. Considering one large external memory for storing video data and intermediate results, the number of parallel bus lines for connecting the operation bus and the memory (N_{bus}) becomes approximately:

$$N_{bus} = R_s \bullet N_{acc.\,pel} \bullet T_{acc} \qquad (8.2)$$

where:

$N_{acc.pel}$ - specifies the mean number of accesses per pel, and

T_{acc} - is the memory access time.

Because of one- and two-operand operations, $N_{acc.pel}$ is larger than $N_{op.pel}$. For simplicity, let us assume that $N_{acc.pel}$ x T_{acc} is in the same order as $T_{op.pel}$ x T_{op}. Thus, the number of bus lines is the same order of magnitude as the concurrency, which is very large. Taking into consideration that the access rate is mainly influenced by multiple accesses of image source data and intermediate results, the access rate to an external memory can be significantly reduced by assigning a local memory to the operative part. Because of the periodicity over the macro block, the local memory is in order to a macro block size.

8.3— Classification of Video Codec VLSI Architectures

Designs of video processors range from fully custom architectures, referred to as ***function specific architectures***, with minimal programmability, to fully ***programmable architectures***, based on multiprocessors. Furthermore, programmable architectures can be classified to ***flexible programmable architectures***, which provide moderate to high flexibility, and ***adapted programmable architectures***, which provide an increased efficiency and less flexibility.

The selection of the architecture depends on the speed requirements of the target application and the constraints on circuit integration, performance, power requirements, and cost. Regardless of the implementation details, discussed later in this section, the general design theme is to use either a DSP or a RISC core processor for main control and special hardware accelerators for the DCT, quantization, entropy encoding, and motion estimation.

In order to assess and evaluate various architectures, the well-known AT-product will be used. Efficiency of an architecture (E) is defined as:

$$E = \frac{1}{A_{si} \bullet T_p} \tag{8.3}$$

where:

T_p - is the effective processing time for one sample, and

A_{si} - is the required silicon area for a specific architecture under evaluation.

8.3.1— Function Specific Architectures

Function specific (or dedicated) architectures for video codecs provide limited, if any, programmability, because they use dedicated architectures for a specific encoding or decoding standard. For high volume consumer products, the silicon area optimization achieved by dedicated architectures, compared to programmable architectures, leads to lower production cost.

Function specific architectures include dedicated chips for DCT, quantization, entropy encoder, and motion estimation algorithm. In the first generation of VLSI chips, each of these functions were implemented in one chip, and a chipset was necessary to create the system for MPEG or H.261 encoding or decoding. Examples of function specific encoders and decoders include AT&T's AV4310 encoder for and AB4220A decoder for MPEG-1 and H.261, AV 6101 and AV6110 MPEG-2 decoders, STi3400 H.261 and MPEG-1 decoder, and Sti3500 MPEG-2 decoder (SGS-Thomson), C-Cube's CL480VCD MPEG-1 audio and video decoder and CL9100 MPEG-2 decoder, and LSI Logic's L64112 MPEG-1 decoder and L64002 MPEG-2 decoder.

Figure 8.2 shows the block diagram of a typical video encoder, based on motion estimation (such as MPEG and H.261/H.263), using function specific architecture.

8.3.2— Programmable Architectures

In contrast to function oriented approach with limited flexibility, programmable architectures enable the processing of different tasks under software control. The main advantage of programmable architectures is the increased flexibility. Changes of architectural requirements, such as changes of algorithms or an extension of the application domain, can be handled by software changes. Thus, with programmable architectures a cost intensive redesign of the hardware can be avoided.

On the other hand, programmable architectures require a higher expense for design and manufacturing, since additional hardware for program control is required. In addition, programmable architectures require software development for the application. Video coding applications require a real-time processing of the image data, and therefore parallelization strategies have to be applied. The two basic alternative parallelization strategies, which will applied in the project, include: (1) data distribution, and (2) task distribution.

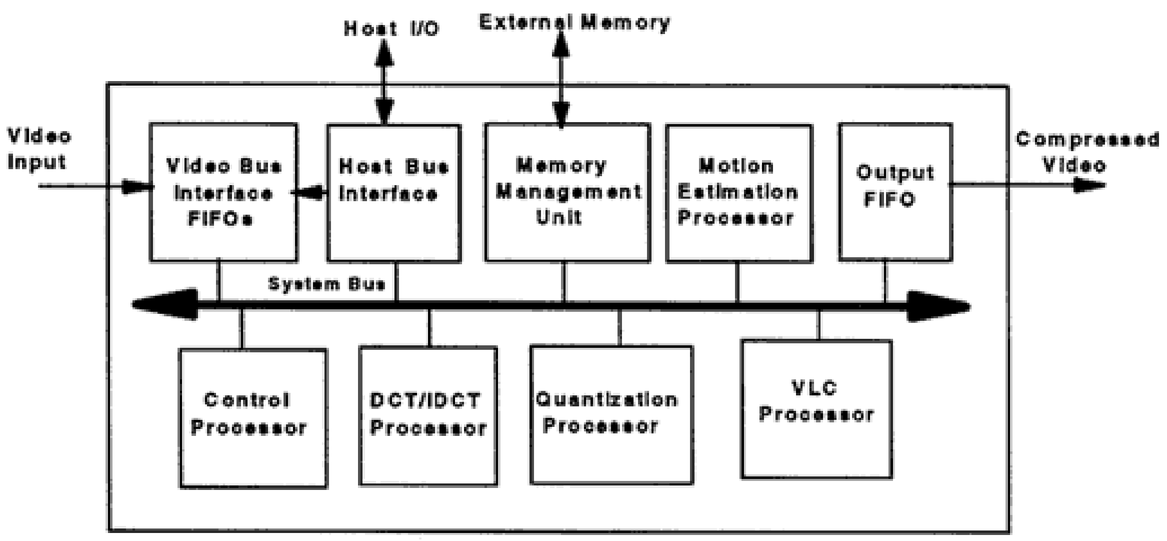


Figure 8.2
Block diagram of a typical function specific architecture for video encoder based on motion estimation. The dedicated processors are used for various operations, such as DCT, quantization, variable length coding, motion estimation, etc.

Two alternative programmable architectures include:

(1) Flexible programmable architectures, with moderate to high flexibility. These architectures are based on coprocessor concept as well as parallel datapaths and deeply pipelined designs with high clock frequency, and

(2) Adapted programmable architectures, with increased efficiency by adapting the architecture to the specific requirements of video coding applications. These architectures provide dedicated modules for several tasks of the video codec algorithm, such as DCT module, or variable length coding.

Figure 8.3, adapted from [PDG95], compares these two programmable architectures in terms of silicon area and frame rate, for several H.261 codec implementations, reported in the literature. Adapted processor design can achieve an efficiency gain in terms of the AT-criterion (eq. 8.3) by a factor of about 6-7 compared to flexible architectures. Assuming approximately 200 operations per pel for the implementation of a H.261 codec [PDG95], this comparison leads to 100 mm^2/GOPS for flexible programmable architectures, and 15 mm^2/GOPS for adapted programmable architectures.

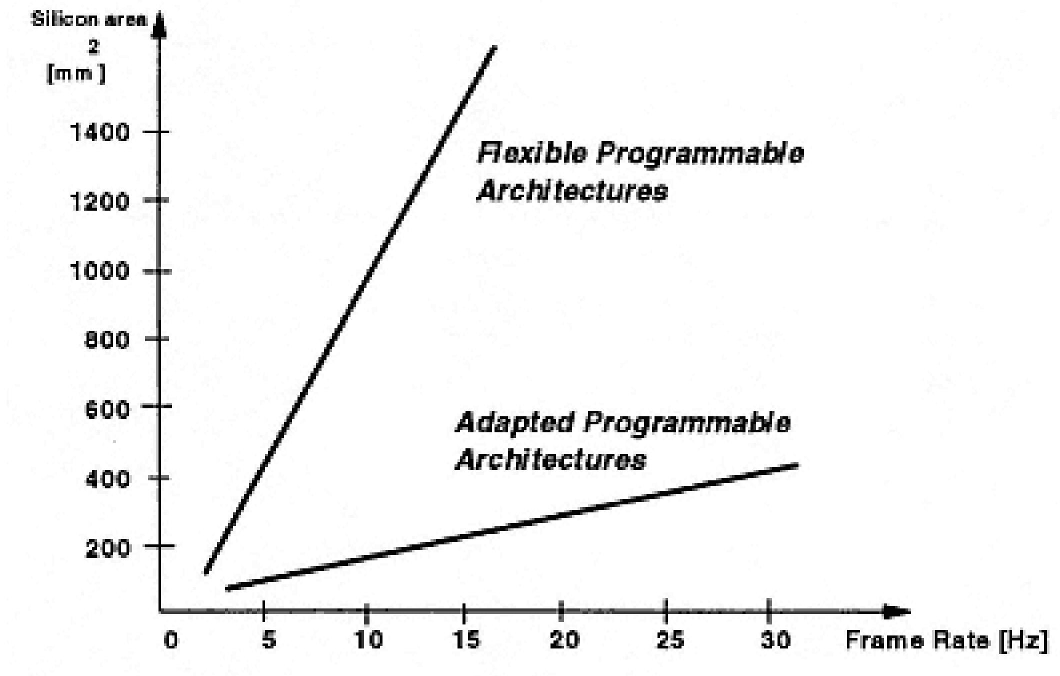


Figure 8.3
Normalized silicon area and throughput (frame rate) for programmable architectures for H.261 codec implementations, reported in the literature [PDG95].

Examples of MPEG-1, MPEG-2, and H.261 flexible programmable architectures include MVP chip (Texas Instruments) [GGA92], and VCP chip (Integrated Information Technology) [FSZ95, BK95], while VSP3 chip (NEC) is an example of H.261 adapted programmable architecture [BK95, PDG95].

8.4— Implementations of the XYZ Video Compression Algorithm

The main disadvantage of dedicated, function specific architectures is the lack of flexibility. Thus, in this book we propose the implementation of the XYZ codec based on a programmable architecture. The selected architecture should balance two controversial requests — flexibility and architectural efficiency. We explore and study two alternative programmable architectures, both based on flexible and adapted programmable approach. Our methodology in this study consists of the following activities:

<u>*1. Mapping tasks into multiprocessor system*</u>

In the first phase, we decompose the XYZ algorithm into a series of concurrent tasks, and map them into multiple processors, as illustrated in an example in Figure 8.4.

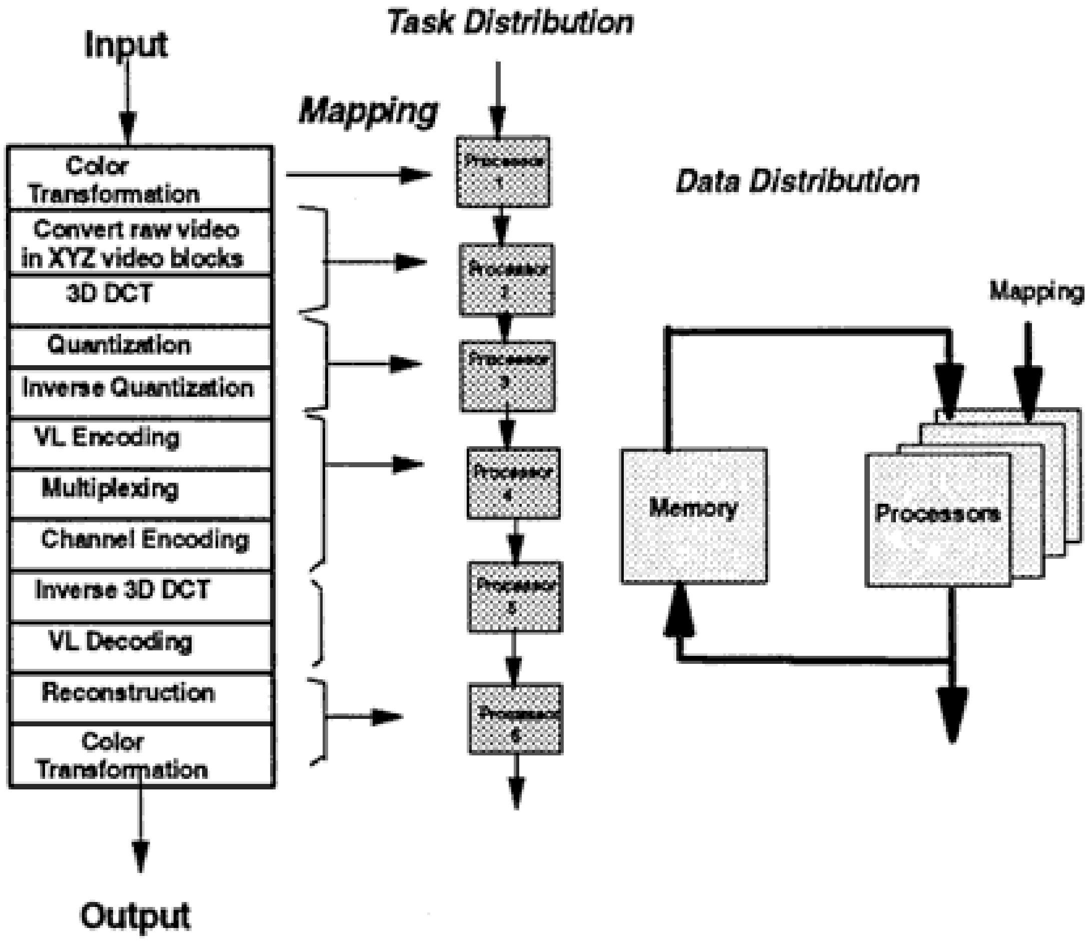


Figure 8.4
Mapping tasks of a XYZ codec into multiprocessor system
exploiting data distribution and task distribution.

Since the results of one task are required for the proceeding task, a local memory is required for storage of intermediate results. The required concurrency can be achieved by parallel operation of processors, whereby a subsection of the image is assigned to each processor. The task mapping onto processors. The task mapping onto set of homogenous programmable processors is referred to as *data distribution*.

In order to increase the silicon efficiency of programmable processors, we will also consider using additional coprocessors for specific functions. This will result in heterogeneous programmable processors referred to as *task distribution*.

We developed two XYZ algorithms, described later in this Chapter:

(1) Non-adaptive XYZ algorithm, which is fast and with lower complexity, and

(2) Adaptive XYZ algorithm, which is of higher complexity, but gives better performance compared to non-adaptive algorithm.

2. Design of two alternative architectures

Two alternative architectures will be exploited and analyzed in this study. The Architecture-1, referred to as FP architecture, will based on flexible programmable architectural concept. The FM architecture will consist of a master general-purpose RISC processor, a number of RISC processors (2 to 10), a shared memory, and an interconnection network, as illustrated in Figure 8.5.

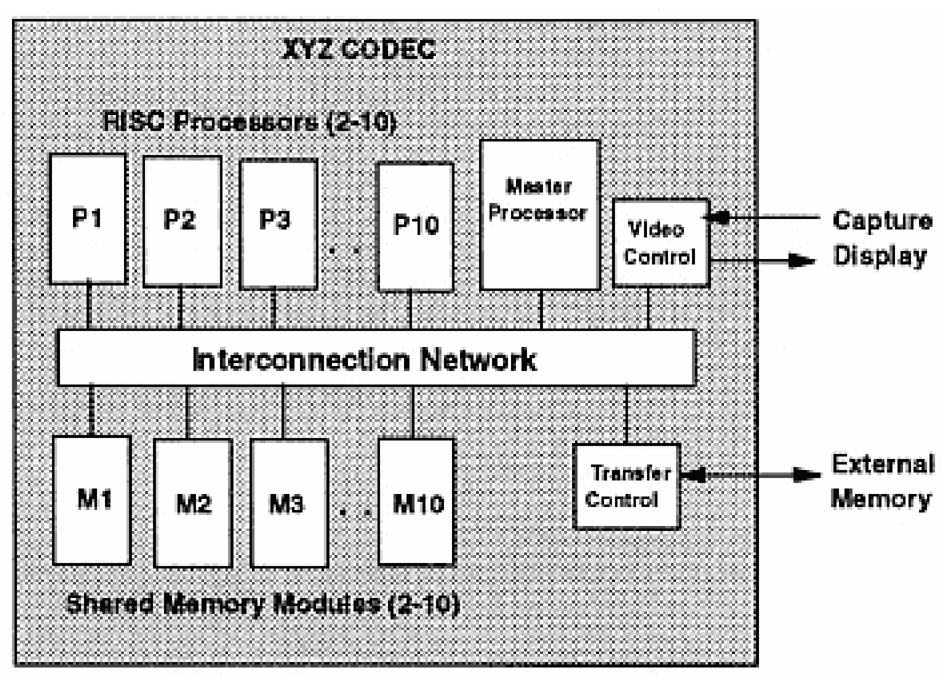


Figure 8.5
An example of a flexible programmable architecture for the XYZ codec implementation. The codec consists of a master RISC processor, 2-10 parallel RISC processors, shared memory, and an interconnection network.

In Sections 8.5 and 8.6 of this Chapter, we present detailed implementations of the XYZ codec using the mesh-based fully programmable architecture and the adapted programmable architecture using dedicated 3D DCT processors.

The core of the system are parallel processors, which handle various digital processing functions in parallel, as pixel processing and other massively parallel integer and floating point operations. The shared RAM memory (2-10 modules) provides on-chip memory for the parallel processors. Each memory module can be accessed in parallel over an interconnection network. Several interconnection networks will be evaluated in this study, including crossbar switch, multistage networks, and common bus.

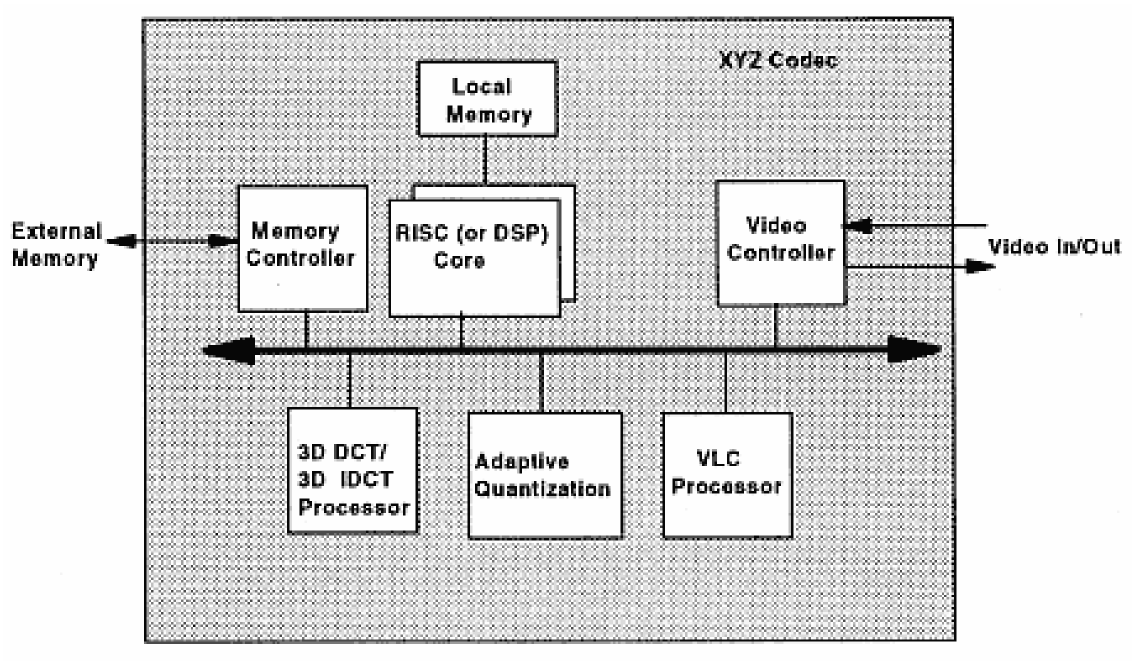


Figure 8.6
An example of adapted programmable architecture for the XYZ codec implementation.
The codec consists of a RISC (or DSP core), and several dedicated processors.

The Architecture-2, referred as AP architecture, will be based on adapted programmable architectural concept, in which we propose to use additional coprocessors for specific functions in order to increase computational power without increasing the required semiconductor areas (see Figure 8.3). In this approach, we intend to combine a flexible programmable concept with one or more adapted modules. We will study what functions of the XYZ codec should be implemented as coprocessors. Typically, computationally intensive tasks, such as DCT and variable length coding, are good candidates for an adapted implementation. In Figure 8.6, one possible implementation of an AP architecture for the XYZ codec is shown. The architecture consists of a RISC (or DSP core) and several dedicated processors for DCT, VLC, and adaptive quantization.

8.4.1— Non-adaptive XYZ Compression Algorithm

The XYZ motion video algorithm is composed of two parts, a video compressor, and a video decompressor. Each part of the algorithm may support adaptive quantization, or may use precomputed tables. The non-adaptive XYZ motion video algorithm is intended to be a very fast, low-overhead algorithm, and it is simple to implement. It compresses motion video in the following steps, as illustrated in Figure 8.7.

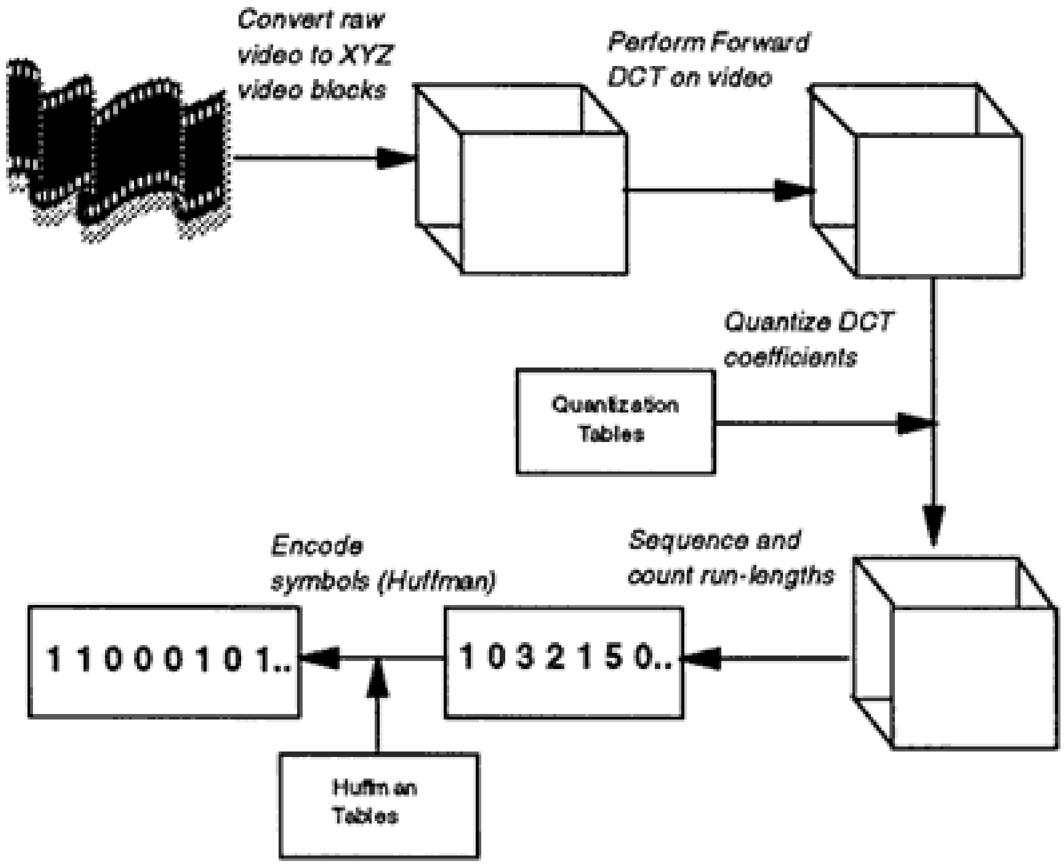

Convert raw video to XYZ video blocks
Perform Forward DCT on video
Quantize DCT coefficients
Quantization Tables
Encode symbols (Huffman)
Sequence and count run-lengths
1 1 0 0 0 1 0 1..
1 0 3 2 1 5 0..
Huffman Tables

Figure 8.7
Overview of the XYZ non-adaptive compression.

1. In the step 1, raw video is collected into 8x8x8 cubes at extremely high data rates. Video is typically captured in YUV format and sent in a digital data stream in 4:2:2 format. This data must be converted into 4:2:0 format and cached into memory. In the process of caching, the data must be re-sequenced into appropriate format for access by the video processor(s).

2. In the second step, the 3D DCT is performed on the raw cubes returning DCT coefficients. The forward DCT may be performed in several different ways depending on the processor throughput and architecture. The forward DCT may be performed by brute force, leading to a simple algorithm that requires considerable processor power, or may be performed by a fast algorithm. Two of the most promising fast algorithms are the Fourier-based algorithm, and a fused multiply-add algorithm. The Fourier-based algorithm minimizes the number of multiplications, and is well suited to general-purpose processors. The fused multiply-add algorithm is better suited to DSPs, where accumulation of products is a one-tick operation. Figure 8.8 compares shows the flowgraphs of the slow FDCT algorithm and a simplified fused add-multiply fast FDCT algorithm.

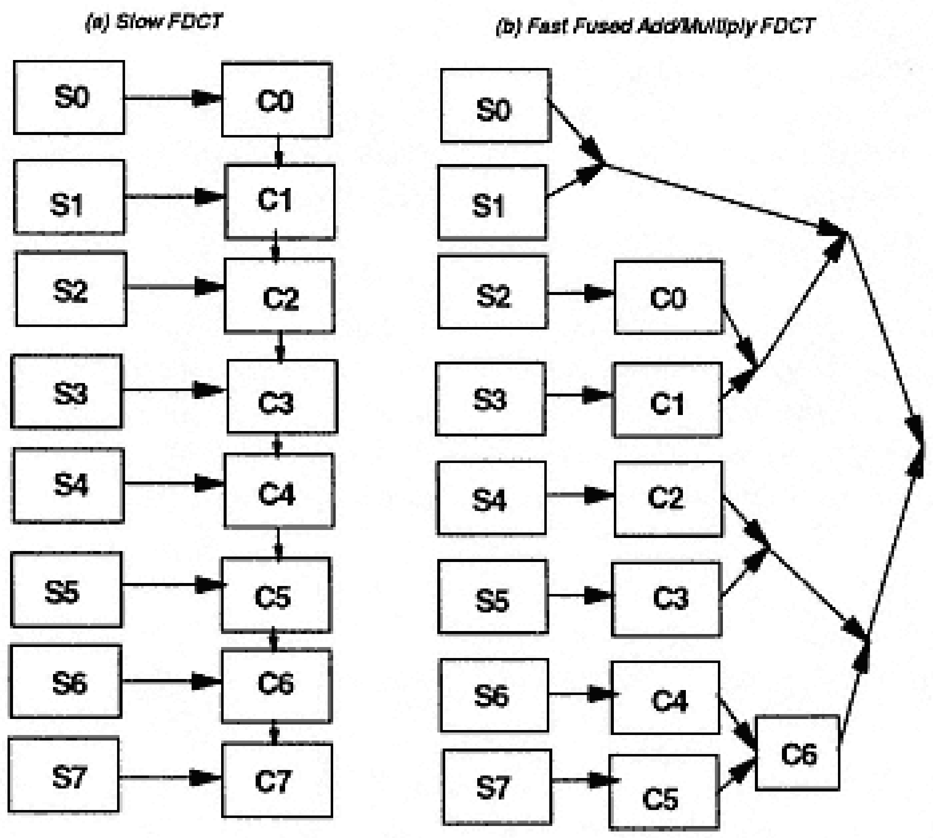


Figure 8.8
Flowgraphs of FDCT algorithms: (a) slow FDCT algorithm, (b) simplified fused add-multiply fast FDCT algorithm.

3. In the next step, the cubes are quantized according to predefined constants. Each DCT coefficient is quantized by dividing it by a constant. One such constant is defined for each DCT coefficient. If either the Fourier-based or the fused multiply-add algorithm is used, the scaling multiplies may be folded into the quantizing factors.

4. The DCT coefficients are sequenced according to a predefined sequence order table. A sequence is defined that typically orders coefficients in decreasing order of variance. This order of coefficients will be used to generate symbols for compression. In practice, this step is integrated into the symbol generation procedure. A new "zig-zag" sequence for the 512 XYZ DCT coefficients must be defined. This sequence may be developed based on the average statistics of different actual video, or on the expected theoretical behavior of the DCT.

5. A sequence of symbols is created and run lengths of zeroes are counted. If a run length exceeds 16, the ZRL symbol is coded. If a run length exhausts the block, no ZRLs are issued and the EOB symbol is coded. When a non-zero coefficient is reached, the run length and the VLC length of the coefficient are combined to form a symbol to be coded.

6. The symbols are encoded into a compressed data stream. Each symbol is replaced by its Huffman code. Huffman codes representing non-zero terminated run lengths are followed by VLC codes. Symbols that occur infrequently and have no explicit Huffman code are represented by the ESC code and the corresponding escape sequence (composed of the run length, the VLC length, and the VLC code).

8.4.2— Adaptive XYZ Compression Algorithm

Adaptive compression is more complex in the sense that tables for quantization and entropy coding are generated from statistics gathered from the entire sequence of eight frames to be compressed, while non-adaptive compression may be performed with reference only to the pixels within a single 8x8x8 cube. Overview of the adaptive XYZ compression algorithm is given in Figure 8.8.

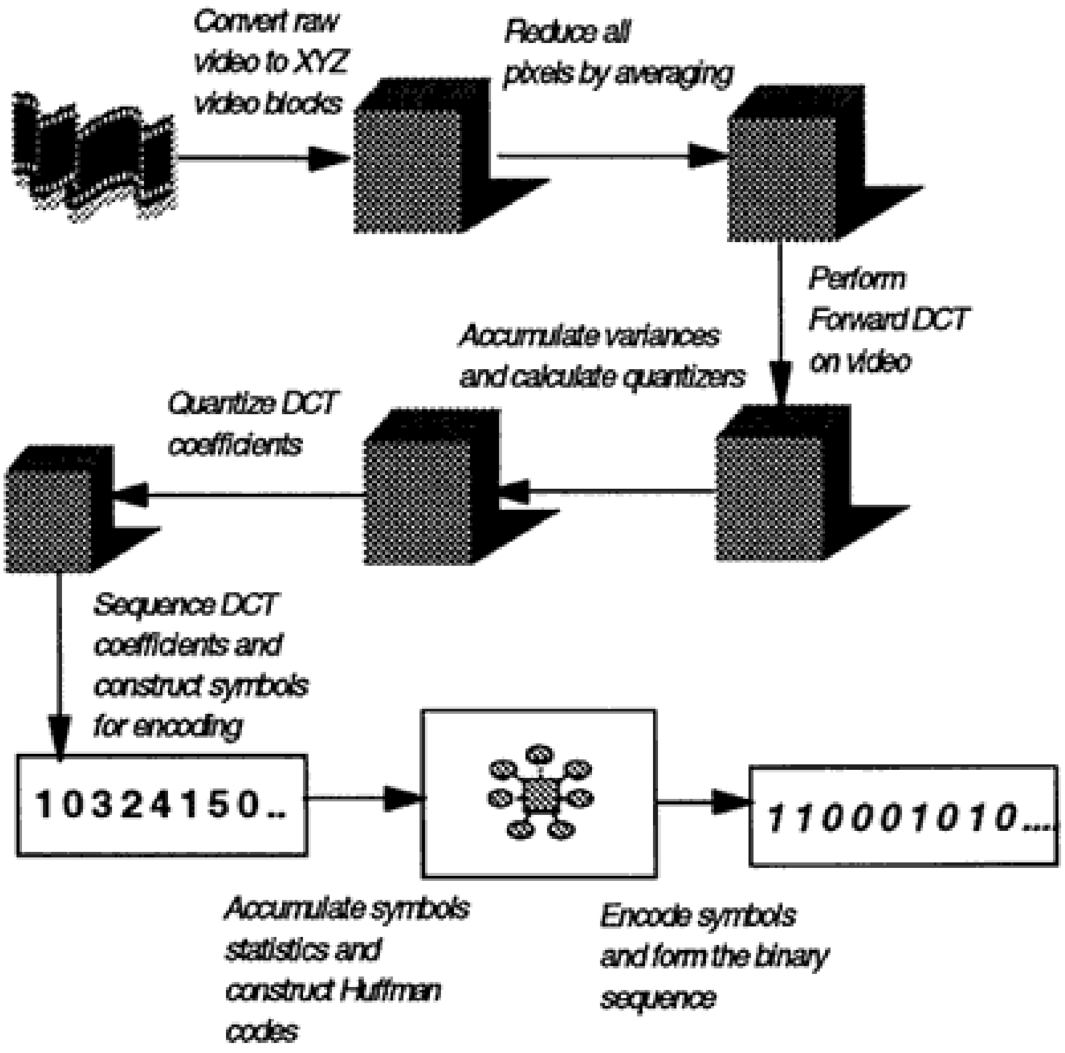

Convert raw video to XYZ video blocks
Reduce all pixels by averaging
Perform Forward DCT on video
Accumulate variances and calculate quantizers
Quantize DCT coefficients
Sequence DCT coefficients and construct symbols for encoding
10324150..
110001010....
Accumulate symbols statistics and construct Huffman codes
Encode symbols and form the binary sequence

Figure 8.8
Overview of the XYZ adaptive compression algorithm.

The adaptive compression algorithm is composed of the following steps:

1. Calculate the mean of all pixels. Calculation of pixel mean involves summing all pixels, then dividing by the number of pixels:

$$\mu = \frac{\sum_{i=0,N-1} s[i]}{N} \qquad (8.4)$$

where:

μ is the pixel mean,

i ranges over all pixels in all 8 frames,

s[i] represents the value of an individual pixel, and

N is the number of pixels in all 8 frames.

However, in a full implementation of adaptive compression, it may be convenient to view compression as taking place in stages. In this case, pixel averages may be computed within each block, and the overall average then computed:

$$\mu_b = \frac{\sum_{i=0,511} s_b[i]}{512}, \qquad \mu = \frac{\sum_B \mu_b}{N_B} \tag{8.5}$$

where:

μ_b is the pixel average for one block,

s_b is a pixel within the block,

μ is the overall average of all pixels,

B ranges over all blocks in the 8 frames, and

N_B is the number of blocks.

2. Subtract the pixel mean from all pixels. Once the mean pixel value has been calculated, all pixels are reduced by the mean:

$$s[i] \leftarrow s[i] - \mu \tag{8.6}$$

where s[i] is a typical pixel to be compressed.

This ensures that all pixels are distributed with a mean of 0, and that all DCT coefficients will be distributed with a mean of 0.

3. Perform the forward DCT on each 8x8x8 block. The forward DCT is now performed on each block of pixels, returning a block of DCT coefficients. Should a general-purpose processor be chosen to perform the DCT, the Fourier-based DCT algorithm may be a good choice to implement this transformation. If a DSP-like processor is used, the fused multiply-add algorithm may be a better choice.

4. Calculate the variance of each DCT component from all blocks. The variance is computed for each of the 512 DCT component within a block. Since the mean of each DCT component is zero, the variance may be calculated as:

$$\sigma_i^2 = \frac{\sum_B S_b^2[i]}{N_B} \qquad (8.7)$$

where:

$\sigma_i 2$ is the variance for coefficient i of a block,

and $S_b[i]$ is the value of coefficient i in block b.

5. Calculate the adaptive quantizer values. Development of adaptive quantizers is done for a specified Normalized Root Mean Square Error (NRMSE), defined as:

$$NRMSE = \sqrt{\frac{\sum_i \left(S_i - S_i'\right)^2}{512}} \Big/ \mu \qquad (8.8)$$

where:

S_i is the ith DCT coefficient,

S_i' is the ith DCT coefficient after recovery from quantization,

and µ is the mean value of the pixels.

6. Calculate the encoding sequence. Since each DCT component has an expectation of 0, smaller variances imply greater likelihood the DCT value is 0. In order to maximize the average run length, the DCT components are encoded in order of decreasing variance. This should give rise to longer run lengths than a fixed coding sequence.

7. Quantize each block. Each block of coefficients is quantized by division by quantizer factors. The quantization of each coefficient within the block is done by adding 1/2 the quantizer value for than coefficient to the coefficient value, and then dividing by the quantizer value. This effects a rounded division by the quantizer.

8. Order the quantized DCT coefficients according to the encoding sequence. A simple lookup table can be used to represent the encoding sequence (Figure 8.9). The encoding process can then be sequentially driven by looking up the index for each coefficient.

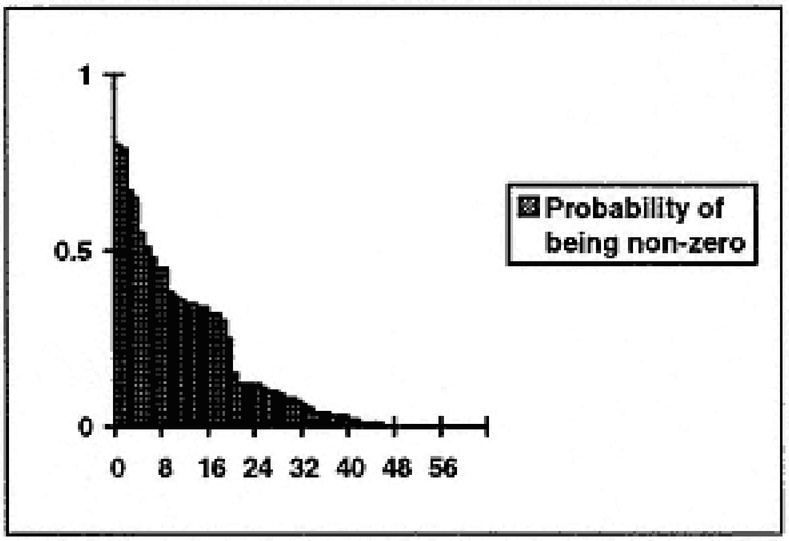


Figure 8.9
Probability of being non-zero for 64 AC coefficients
when ordered in XYZ sequence.

9. Compute the run lengths of zeros to generate sequences of symbols to encode. Run lengths of zeros are counted until a non-zero coefficient is found, or the end of block. If the run length reaches the end of block, the EOB symbol is coded. If the run length is longer than 16, ZRL symbols are coded for each run length of 16. The remaining run length and the VLC code for the non-zero coefficient are combined to for a symbol for Huffman encoding.

10. Collect number of occurrences of each symbol. As symbols are formed, their occurrence is counted. The accumulation of symbol statistics is a prerequisite for creating Huffman codes.

11. Construct the Huffman encoding tables. A tree is formed collecting lowest-probability symbol events, and is used to encode the symbols. The tree is pruned to limit the length of code words to 12 bits. The pruned symbols will be represented by the ESC symbol.

12. Encode the symbols into a compressed data stream. The symbols within a block are encoded according to the Huffman codes. The bit streams are collected across blocks to form a single compressed data stream, as shown in Figure 8.10.

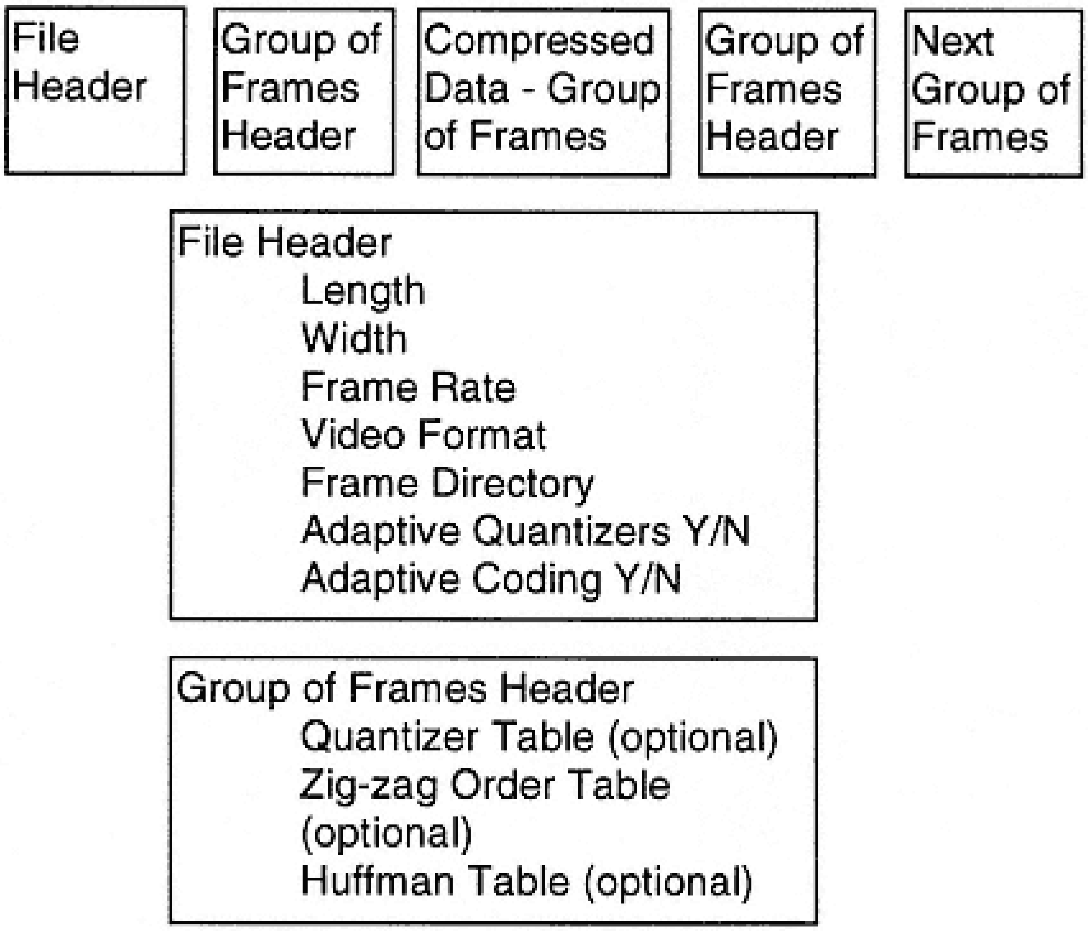


Figure 8.10
Format of the XYZ compressed data stream.

Decompression is done following essentially the non-adaptive encode procedure in reverse. Some additional overhead is dedicated to construction of Huffman lookup tables when decompressing data streams that were adaptively encoded. The decompression procedure consists of the following steps:

1. Construction of Huffman decoding tables. For non-adaptive data streams, this is done once at the start of the decode process. For adaptive data streams, the Huffman codes are read at the beginning of the data stream representing one group of eight compressed frames.

2. Assignment of quantizer factors. For non-adaptive data streams, quantizing factors are assigned at the start of the decode process. For adaptive data streams, quantizing factors are read at the beginning of the data stream representing one group of eight compressed frames.

3. Decode Huffman data stream into stream of symbols. The longest Huffman code used by XYZ compression is 12 bits in length. The next 12 bits in the data stream is used as an index into the Huffman decoding table. Thus the Huffman decoding table must recognize Huffman codes of varying lengths. This is accomplished in the construction of the Huffman decoding table, as shown in Figure 8.11. For each Huffman code, every possible 12 bit index with that code as prefix is generated. In the case of collisions, the longer prefix takes priority. Once completed, every Huffman code will correctly index the table.

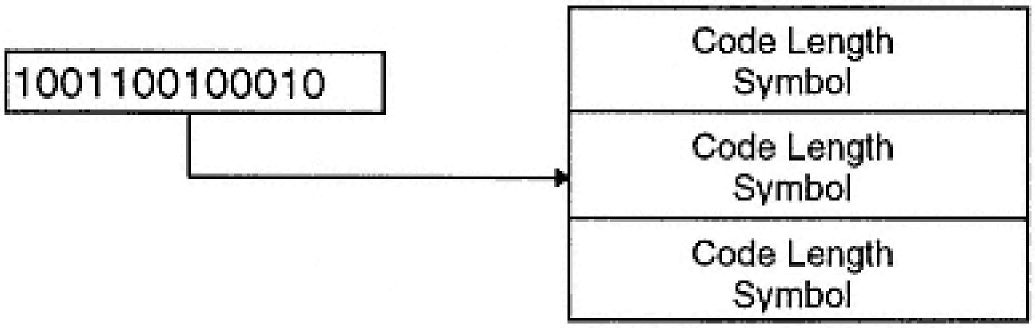


Figure 8.11
The Huffman Decoding Table

4. Expand symbol stream into DCT coefficients. As the symbols are decoded, they cause specify zero run lengths and non-zero DCT coefficients. These values are resequenced from their encoded zig-zag order and stored into the DCT block.

5. Perform IDCT on DCT coefficients. An inverse DCT is performed on the DCT coefficients, returning pixel values. The pixel values must be clamped to their legal range (0..255).

6. Reorder pixel data and display. Pixel data must be sequenced in scan-line order, and played through a D/A converter to generate a frame of video.

8.5— Adaptive XYZ Codec Using Mesh Architecture

The adaptive XYZ compression algorithm is well suited to mesh implementation. The implementation of the algorithm follows the steps given in Figure 8.8. Each so-called "global" stage performs an essentially single process of highly-coupled multi-process task. The "block" stages perform highly independent, parallel tasks.

Assumption is made that enough local or shared memory is available to the processors to store the entire contents of 8 frames (N_B blocks). N_p parallel processors are organized into a 2-D array of length n_x and depth n_y. Blocks of pixels are allocated to each parallel processor in row-stripe fashion. The maximum number of blocks per processor N_b is:

$$N_b = \left\lceil \frac{N_B}{N_p} \right\rceil \tag{8.9}$$

Parallel processors are connected to their 4 nearest neighbors. Cut-through routing is provided as well as copy during routing.

1. Compute mean pixel value. Each processor accumulates the pixel values for each block it has been allocated. Each processor performs 512 additions per block (ignoring looping overhead). Time to accumulate pixel values on each processor is:

$$T_1 = N_b * 512 * t_a \tag{8.10}$$

where:

t_a is the time required to perform a single addition, and

t_1 is loop overhead time.

The average value and the number of blocks per processor are collected across processor rows and columns. Collection across n_x rows is done in $\log_2(n_x)$-1 parallel collection steps. At each step i, data is sent from processors whose x rank 2^{i+1} is 2^i to the processor whose rank is x rank - 2^I, as illustrated in Figure 8.12.

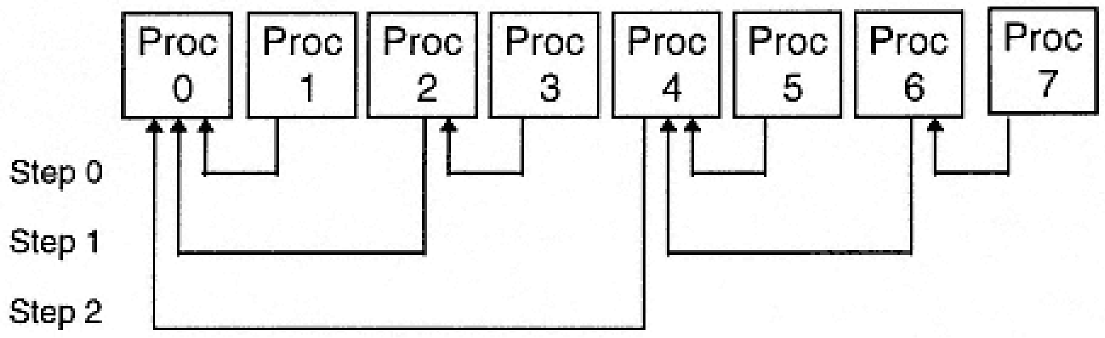


Figure 8.12
Collecting values in the x direction.

Time to perform cut-through routing transfers is modeled as (Figure 8.13):

$$t_{comm} = t_s + mt_w l \qquad (8.11)$$

where:

t_{comm} is the total communications overhead,

t_s is the startup overhead,

m is the number of words in the message,

t_w is the transfer time per word, and

1 is the number of links over which the transfer will be made.

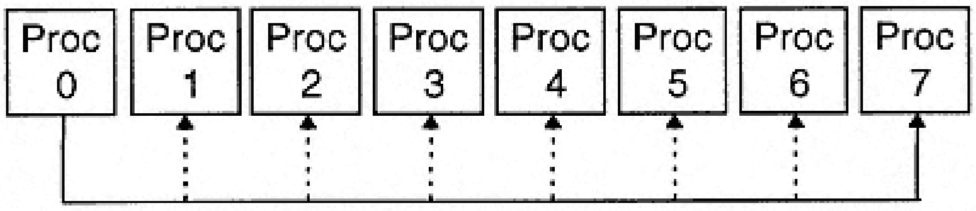


Figure 8.13
Distributing values in the x direction using the copy cut-through operation.

The time to collect m words over n_x processors can be calculated. At each step i, $l = 2^i$. The total communication time in the x direction is then:

$$T_x = \sum_{i=0,\log(n_x)-1} t_s + m * t_w * 2^i \qquad (8.12)$$

$$T_x = \log(n_x)*t_s + m*t_w*(n_x - 1) \quad (8.13)$$

Assuming $t_w n_x$ dominates $t_s \log(n_x)$, we conservatively estimate:

$$T_x = m*n_x*t_w \quad (8.14)$$

The same reasoning applies in the y direction. Total communication time to distribute m words is then:

$$T_d = m*(n_x + n_y)*t_w \quad (8.15)$$

Time to collect two words (number of blocks, total pixel value) is:

$$T_2 = m*(n_x + n_y)*t_w \quad (8.16)$$

At each step in the communication process, the accumulation of the total number of blocks and the total pixel values is made. At the last step, the average pixel value is calculated by dividing the accumulated pixel total by the accumulated total number of blocks. The time to perform these calculations is:

$$T_3 = (\log(n_x) + \log(n_y))*(2t_a) + t_m \quad (8.17)$$

where t_m is the time needed to perform a multiply/divide.

2. Reduce pixels by mean pixel value, perform FDCT, calculate variances. The mean pixel value must be distributed amongst all processors. This effort requires a cut-through copy distribution in the x direction, and then the y direction. The overhead to distribute m words in the x direction is:

$$T_x = t_s + mt_w(n_x - 1) \quad (8.18)$$

which may be conservatively approximated by:

$$T_x = m * n_x * t_w \tag{8.19}$$

Distribution to all processors is done by distribution in the x direction followed by distribution in the y direction. Thus time to distribute m words to all processors is:

$$T_d = m * (n_x + n_y) * t_w \tag{8.20}$$

The time to distribute the mean is given as:

$$T_4 = (n_x + n_y) * t_w \tag{8.21}$$

Reducing the pixel values by the mean is a loop of 512 subtractions per block, and thus the time is:

$$T_5 = N_b * 512 * t_a \tag{8.22}$$

The forward DCT is calculated independently on all processors. Three passes are taken over each block. 64 one-dimensional DCTs are performed for each pass. The time needed to perform a single 8 word FFT-based unscaled DCT is:

$$t_f = 5t_m + 29t_a \tag{8.23}$$

The time to perform the unscaled 3-D DCT to all blocks is then:

$$T_f = N_b * (960t_m + 5568t_a) \tag{8.24}$$

Adding the scaling operation (one multiplication per each of 512 coefficients) to the DCT gives an overall DCT computation time of:

$$T_6 = N_b * (1472t_m + 5568t_a) \tag{8.25}$$

Variances are calculated as the square of each DCT component. The time to calculate the individual variances for one block is then one multiplication per coefficient, and for all blocks:

$$T_7 = N_b * 512 * t_m \tag{8.26}$$

3. Accumulate variances, compute quantizing factors, compute encoding sequence. Variances are accumulated in much the same manner as the pixel mean is computed. Of course, accumulations of 512 variances takes place. The time to accumulate the 512 variances over the blocks within one processor is:

$$T_8 = N_b * 512 * t_a \tag{8.27}$$

The communication time to accumulate variances is:

$$T_9 = 512 * \left(n_x + n_y\right) * t_w \tag{8.28}$$

The total of the time needed to accumulate variances at each communication step, plus the time needed to scale the variances after accumulation, is:

$$T_{10} = 512 * \left(\log\left(n_x\right) + \log\left(n_y\right)\right) * \left(t_a\right) + 512 * t_m \tag{8.29}$$

Computation of quantizing factors requires sort of the variances followed by iteration of the bit allocation algorithm until convergence of the Root Variance Error. Sorting 512 coefficients takes on the order of 512log(512) operations. Double this number to arrive at the sort overhead estimate of:

$$T_{11} = 8192 t_a \tag{8.30}$$

In typical video sources, the accumulated variance requires on the order of 20 bits to encode at NRMSE error levels of 5%. This algorithm requires, per iteration, an average of roughly 16 comparisons to determine the largest unrepresented variance, 3 adds and 3 multiplies. Startup overhead is 4096 assignments. Bit allocation overhead is then roughly (assignment overhead is assumed to be t_a):

$$T_{12} = 4296 t_a + 60 t_m \tag{8.31}$$

Quantizers are developed by dividing expected range of each coefficient (square root of variance) into range represented by each coefficient's bit allocation. The requires 512 square roots, roughly 20 shifts, and 512 divisions. Assuming a shift takes place in time t_a and a square root requires roughly 16 multiplication times, generation of quantizers takes:

$$T_{13} = 8704 t_m + 20 t_a \tag{8.32}$$

4. Quantize DCT coefficients, construct symbol sequences for encoding. Calculated quantizers and the encoding order must be broadcast to each processor. Per the pixel average broadcast above, the time to broadcast 512 quantizers and the 512 word coding sequence array is:

$$T_{14} = 1024 * \left(n_x + n_y\right) * t_w \tag{8.33}$$

Quantization per block is done with a shift, an add, and a multiply per coefficient (of the 512 in the block). Total quantization time is:

$$T_{15} = N_b * 512 * \left(2t_a + t_m\right) \tag{8.34}$$

Calculation of symbols is done by counting zero run lengths in the coding sequence. This requires 512 decisions and for most of them, an increment operation. For about 8 of the coefficients, a non-zero value will be detected. In this case, the logarithm must be calculated to compute the VLC length. Overall, the cost of this procedure is 20 shifts. The cost of this process is estimated at one add time per sequencing lookup, one add time per decision, and either one add time per zero or 2.5 shifts per VLC code:

$$T_{16} = N_b * \left(492 * \left(3t_a\right) + 8 * \left(2.5 t_a\right)\right) \tag{8.35}$$

5. Collect symbol statistics, construct Huffman codes. The collection of symbol statistics follows a model quite similar to calculating the pixel mean or the coefficient variances. There are 256 symbols. First the frequency of symbol occurrence is measured for each processor. The time to do this is 256 initialization assignments, and approximately 8 indexed lookups and 8 increments per block (still ignoring control overhead!):

$$T_{17} = N_b * 16 t_a + 256 t_a \tag{8.36}$$

The 256-word frequency of symbols is collected across processors. Total communication time is:

$$T_{18} = 256*\left(n_x + n_y\right)*t_w \tag{8.37}$$

At each communication step, statistics are accumulated. The time to do this is:

$$T_{19} = N_b*256*t_a \tag{8.38}$$

Once the symbol statistics have been calculated, a Huffman tree may be constructed. This is a O(nlog(n)) algorithm, and the constant factor will be estimated at 20. Thus Huffman tree construction is estimated at:

$$T_{20} = 40960t_a \tag{8.39}$$

The resulting coding table is 256 words long. This table is distributed to all processors. This takes:

$$T_{21} = 256*\left(n_x + n_y\right)*t_w \tag{8.40}$$

6. Encode symbols. Each block of symbols is encoded on its processor. There are approximately 8 symbols per block. Each symbol requires a lookup in the Huffman coding table, and about 4 shift and or operations to insert the lookup bits into the compressed bitstream. Another add is needed to count the number of bits in the bitstream. The time to perform this encoding is then:

$$T_{22} = N_b*40t_a \tag{8.41}$$

The encoded bitstreams must be collected into a single compressed bitstream. If the compression ratio after encoding is 32:1, the length of the bitstream will be 9 words on average (1 extra word for the length of the data stream). Collecting this bitstream will take:

$$T_{23} = N_b * 9 * (n_x + n_y) * t_w \qquad (8.42)$$

At each step in the collection process, the two bitstreams are concatenated. For step i, the bitstreams are of length $8*2^i$, and the number of operations to concatenate is roughly four shifts and two logical operations per byte:

$$t_i = N_b * 2^i * 6 t_a \qquad (8.43)$$

The total number of steps is nx+ny, making the total number of logical operations:

$$T_{24} = N_B * 6 t_a \qquad (8.44)$$

The total number of operations is summarized in Table 8.3. If we assume $t_w=t_a$, $t_m=16t_a$, N_B=10,800, N_p=4096, N_b=3, $n_x=n_y$=32, we get the results in Table 8.4.

Table 8.3 Summary of expected complexity of adaptive XYZ algorithm on mesh architecture.

T_n	**t_a**	**t_m**	**t_w**	**loop**
1	$512N_b$			$512N_b$
2		1	$2n_x+2n_y$	$\log(n_x)+\log(n_y)$
3	$2\log(n_x)+2\log(nv_y)$	1		
4			n_x+n_y	
5	$512N_b$			$512N_b$
6	$5568N_b$	$1472N_b$		$704N_b$
7		$512N_b$		$512N_b$
8	$512N_b$			$512N_b$
9			$512n_x+512n_y$	$\log(n_x)+\log(n_y)$
10	$512\log(n_x)+512\log(n_y)$	512		
11	8192			8192
12	4296	60		20
13	20	8704		512
14			$1024n_x+1024n_y$	1024
15	$1024N_b$	$512N_b$		512
16	$1496N_b$		512	
17	$16N_b+256$			512
18			$256n_x+256n_y$	256
19	$256N_b$			256
20	40960			16384
21			$256n_x+256n_y$	256
22	$40N_b$			8
23			$9N_b$	8
24	$6N_B$			n_x+n_y

Table 8.4 Complexity of XYZ adaptive and non-adaptive algorithms for one video block 8x8x8.

COMPRESSION ALGORITHM	NUMBER OF OPERATIONS [PER BLOCK OF 8x8x8 FRAMES]	NUMBER OF OPERATIONS [PER SECOND]
Adaptive XYZ	454,855	1,705,706
Non-adaptive XYZ	116,180	435,675

8.6— XYZ Codec Based on Fast 3D DCT Coprocessor

From the previous analysis, 64% of the overhead the non-adaptive compression process is due to the 3-dimensional DCT. Optimizing this transformation is a key part of developing a practical real-time algorithm.

Applying a Fourier-based fast DCT separately to the 3-D case will require 3X64X13 multiplications, and 3x64x29 additions: 2,496 multiplications and 5,568 additions. However, the Fourier-based DCT supports folding in of constants into the quantizing phase. This results in a total of 3x64x5 multiplications: 960 multiplications and 5,568 additions. Table 8.5 summarizes the complexity of two 3D DCT algorithms, first based on direct 1D DCT, and second one based on direct fast 1D DCT.

Table 8.5 Complexity of two 3D DCT algorithms for various image formats.

3D DCT Algorithm	Format	Number of video cubes/second	Complexity [MOPS]
Direct 1D DCT	CIF 320x240 at 30fps	9,000	220
Direct 1D DCT	NTSC 720x480 at 30 fps	40,500	988
Direct 1D DCT	HDTV 1280x1024 at 60 fps	307,200	7,490
Direct Fast 1D DCT	CIF 320x240 at 30 fps	9,000	60
Direct Fast 1D DCT	NTSC 720x480 at 30 fps	40,500	256
Direct Fast 1D DCT	HDTV 1280x1024 at 60 fps	307,200	2,000

Clearly, a single-processor system is unworkable for any high-resolution system (but may be appropriate for quarter-screen motion). Thus we investigate parallel architectures. The use of modern DSP processors is a particularly interesting investigative angle, as new core processors are capable of over 100 MIPS, overlap address calculations with processing of data, and make an inexpensive engine for computationally complex problems.

The first problem that has to be resolved is real-time access to inexpensive (say 70 ns) memory. Frames of video are digitized and written to alternate buffers, as shown in Figure 8.14. A simple state machine is used to reorganize the digital data into XYZ sequence. Video data is cached in fast shift registers and written to slower banks of memory.

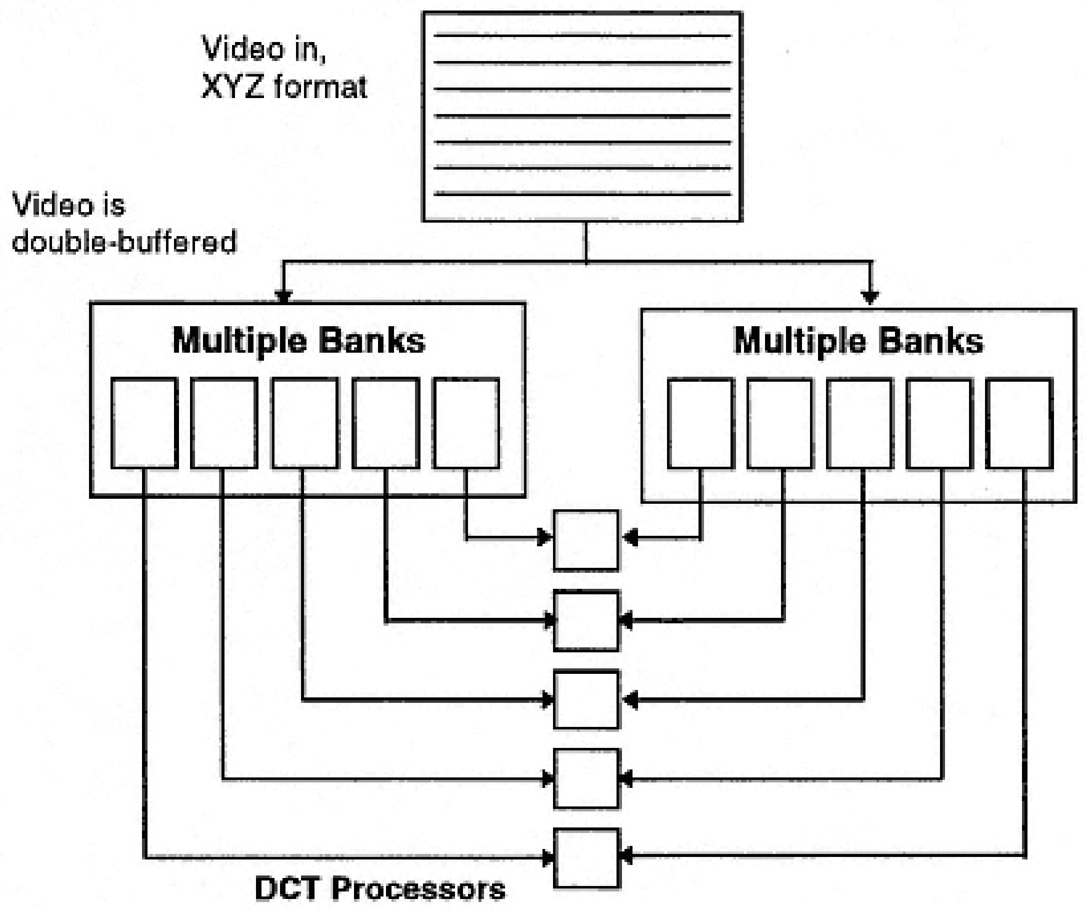


Figure 8.14
Buffering the input to the DCT processors.

Each memory buffer must be large enough to hold 8 frames of video. When full, the processing array can start retrieving data from the frame buffer. Contention is resolved by subdividing the frame buffer into separately addressable banks of memory. Fast interface to each memory bank is designed as shown in Figure 8.15.

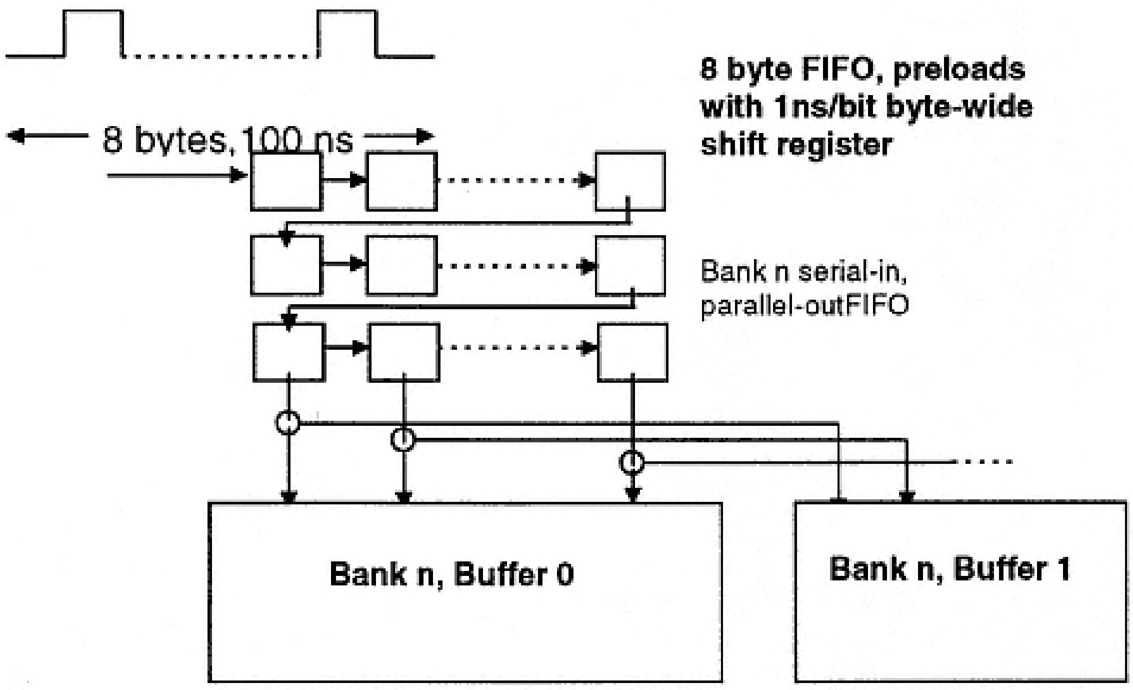

8 byte FIFO, preloads with 1ns/bit byte-wide shift register
8 bytes, 100 ns
Bank n serial-in, parallel-outFIFO
Bank n, Buffer 0
Bank n, Buffer 1

Figure 8.15
Fast interface to each memory bank.

The DCT processors are assumed to have 512 words of local memory. Each processor reads a cube for processing, and performs the DCT on the cube. Timings are given for the DSP fused multiply-accumulate architecture in Table 8.6.

The time to perform the 3D DCT on a "typical" DSP is then:

$$T_{3D} = 9088t_p + 1024t_m \tag{8.45}$$

where:

t_p is the processor cycle time,

and t_m is the memory cycle time.

Table 8.6 Implementation of the fused multiply-add FDCT on a typical DSP.

Operation	**Number of loops**	**Time per loop** [memory/processor cycles]	**Total Time** [memory/processor cycles]
Read 512 words from bank	512	1	512
3 passes: one per X, Y, Z direction	3	2688	8064
64 DCTs	64	42	2688
Address calculation	8	2	16
DCT calculation	8	2	26
Post scaling	512	2	1024
Write 512 words to output bank	512	1	512

The overhead of address calculation of the data points (representing access of data points) is 3072 processor cycles. This is a considerable portion of the overall DCT overhead, and is worth eliminating.

One means of elimination of address computation overhead is construction of a hardwired "shuffle" interconnection scheme, presented in Figure 8.16. This architecture associates 64 DCT processors with a single 512 word processing cache. A 512 word cache is constructed to hold the contents of one 8x8x8 cube. The cube is initially loaded from the memory bank it services. The contents of the cube are distributed to each of 64 DCT processors, eight consecutive words of the X direction per processor. After processing, the new values are redistributed for processing in the Y direction, and finally in the Z direction. The timing for receipt and transmission of data is illustrated in Figure 817.

The time required to process the 3D DCT (T) is a function of the initial load time (T_L), the processing time (T_{Pr}) and the permutation time (T_{Pe}). Assuming the time required to move the data out to the next stage after DCT processing (i.e., quantization) is equal to the permutation time:

$$T_{3D} = T_L + 3T_{Pr} + 3T_{Pe} \qquad (8.46)$$

Load and permutation time can be reduced by wiring each word of temporary storage individually to the input processor and output processor it services. In this case, although complex wiring is required (512 individually wired 16-bit words), the load time can be pipelined with processing time, and shuffle time reduced to a single load/store time. The shuffle pattern is simply a transposition in three dimensions:

$$S'[j][k][i] = S[i][j][k] \tag{8.47}$$

The latest generation of DSPs support an alternative, more scaleable solution. The ADSP-2106x family of Super Harvard Architecture Computers (SHARC) and the TMS320C8x are used to benchmark a fast FDCT design.

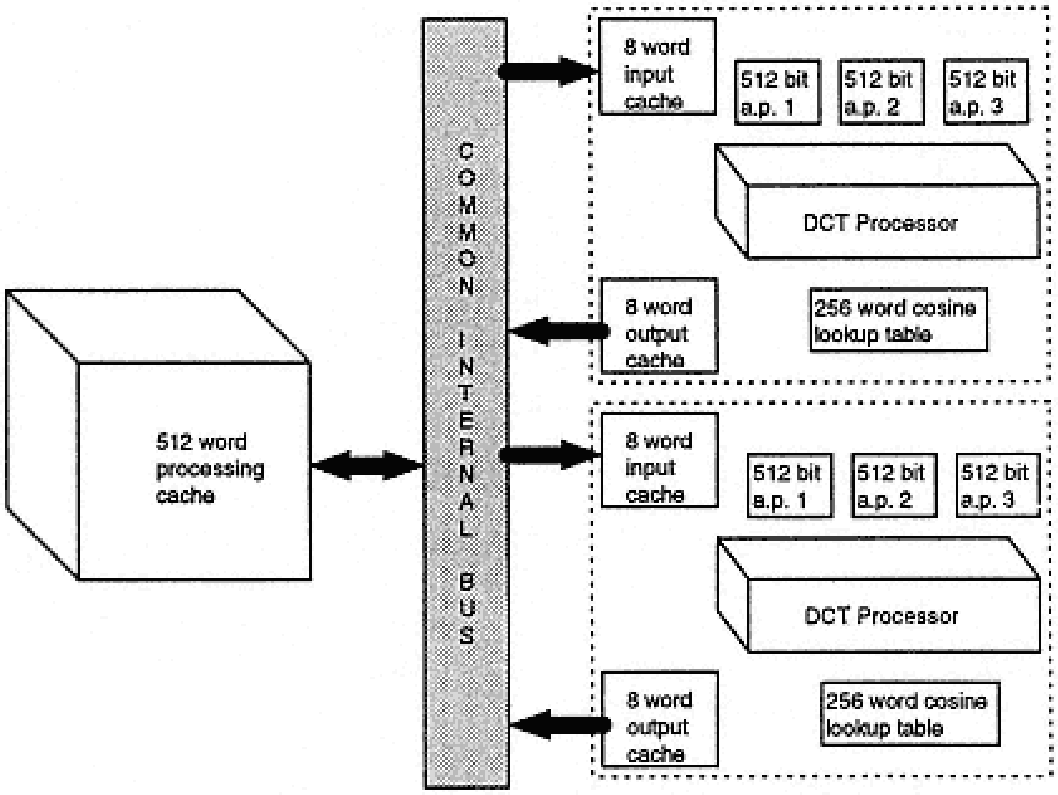


Figure 8.16
Shuffle interconnection of DCT processors.

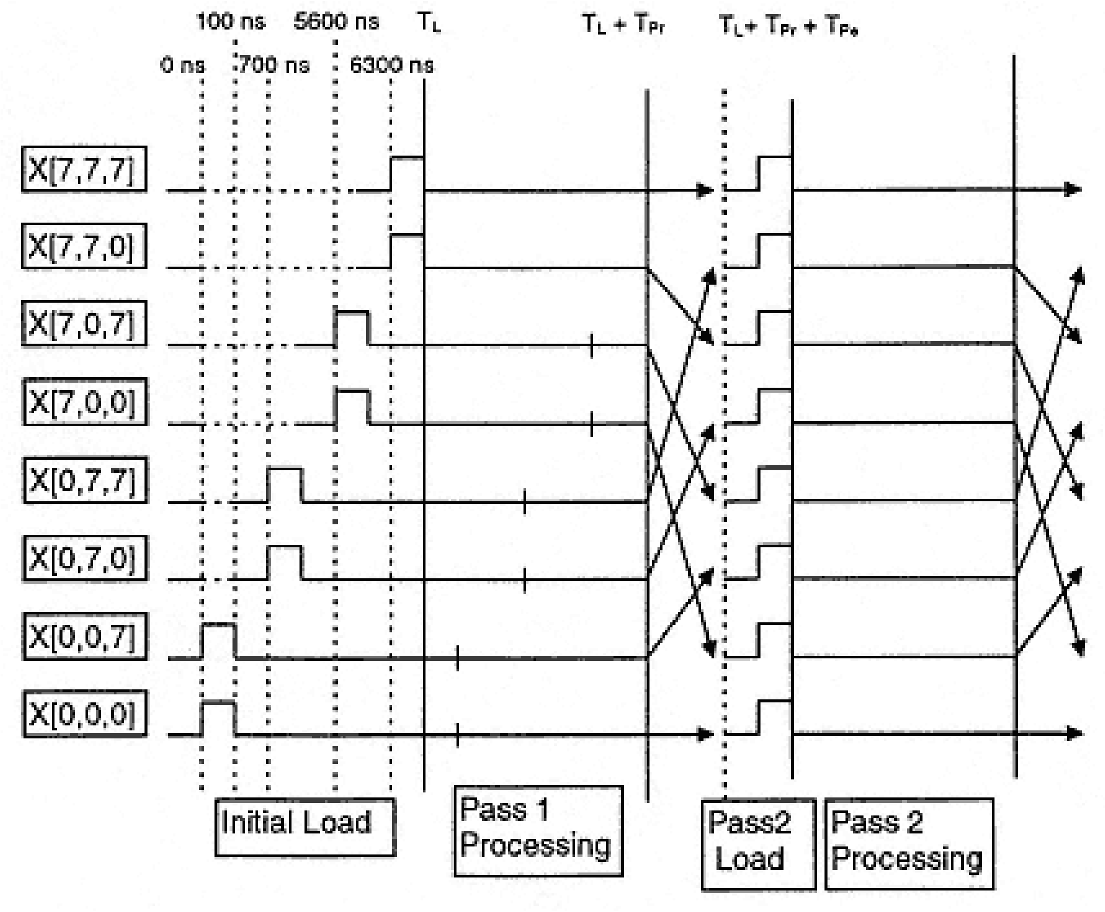


Figure 8.17
Timing diagram for shuffle interconnect between DCT processors.

8.6.1— Fast DCT Based on the SHARC Architecture

The SHARC architecture incorporates processing capability, RAM, DMA capability, multiple buses, address computation logic, and a single-tick multiply into a single chip. This architecture is depicted in Figure 8.18.

Two banks of dual ported SRAM each support simultaneous access from the DMA controller and one of the two Data Buses. The Sequencer fetches a single instruction from the Instruction Cache or the Program Memory bus. The instruction may:

• compute with registers,

• perform two move operations between registers and the Data Memory or Program Memory bus,

• perform address calculation for the next instruction.

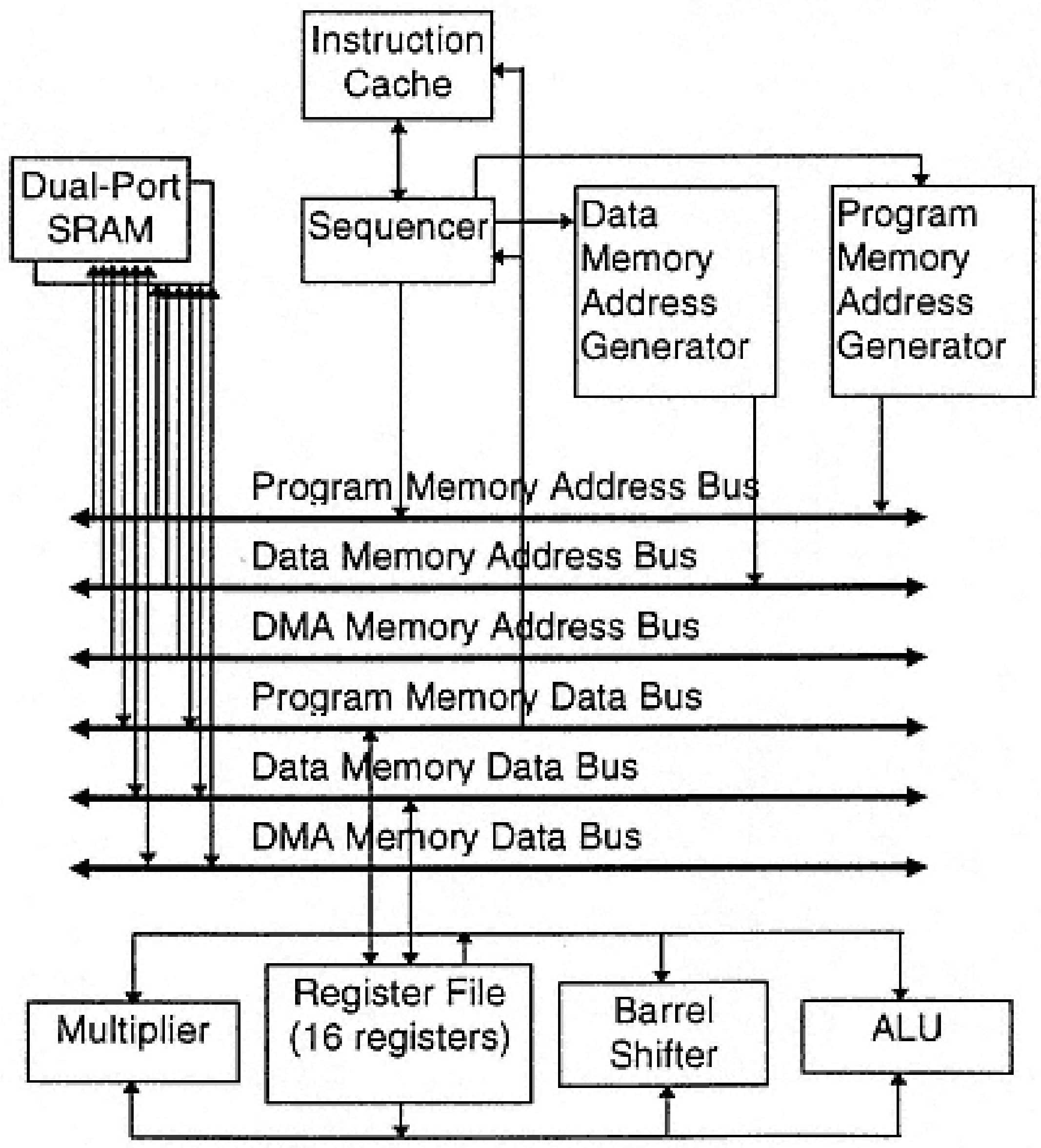


Figure 8.18
SHARC architecture.

Any instruction that is fetched from Program Memory and references Program Memory requires a wait state. Thus instructions that reference Program Memory are cached (Instruction Cache is associative). Calculations are performed between registers. Supported calculations include one tick multiply/add, shifting, general ALU operations, variable-length bit insertion/deletion. Address calculations allow increment of an address base by a programmable value. This increment may support circular queues. Eight address calculations are simultaneously active (each instruction chooses an address calculation reference).

The SHARC architecture is well-suited to implementation of the FDCT. The DMA controller can write video data into memory while the SHARC is calculating the previous DCT. Constants used in multiplications are read from Program Memory, and are therefore cached and available in a single tick. Address calculation is done in the same instruction as the multiply/add, and so the overall time to perform the DCT is:

$$T_{3D} = \max\left(4992t_p, 1024t_m\right) \tag{8.48}$$

Table 8.7 summarizes memory and processor cycles needed to implement 3D DCT on SHARC processor. At NTSC rates, five 40 MHz SHARC chips should be sufficient to perform the DCT.

Table 8.7 Implementation of the fused multiply-add FDCT on a SHARC DSP.

Operation	**Number of loops**	**Time per loop** [memory/ processor cycles]	**Total time** [memory/processor cycles]
Read 512 words from bank	512	1	512
3 passes, one per X, Y, Z direction	3	1664	4992
64 DCTs	64	26	1664
Address Calculation	8	0	0
DCT Calculation	8		26
Post Scaling	512	2	1024
Write 512 words to output bank	512	1	512

8.6.2— Fast DCT Based on the TMS320C80 Processor

The TMS320C80 is a single chip that incorporates many of the system features of video distribution and parallel processing. The TMS320C80 is composed of a master RISC processor, four DSP processors, a smart DMA controller (transfer controller), and a video controller, as shown in Figure 8.19.

Each pixel processor incorporates processing capability, multiple buses, address computation logic, and a single-tick multiply into a single chip. This architecture is depicted in Figure 8.20. The external (to the pixel processor) crossbar enables access to "local" memory or to external memory. In the absence of contention, each pixel processor can access local memory in a single tick. Instructions are cached in standard LRU fashion. Instructions perform the following operations in one tick:

• A multiply/accumulate operation between registers.

• Two move operations between registers and the Local Memory Bus or the Global Memory Bus. These buses' names are by convention - in practice they are interchangeable. Move operations include variable-length bit extraction.

• Address calculation for the data move operations.

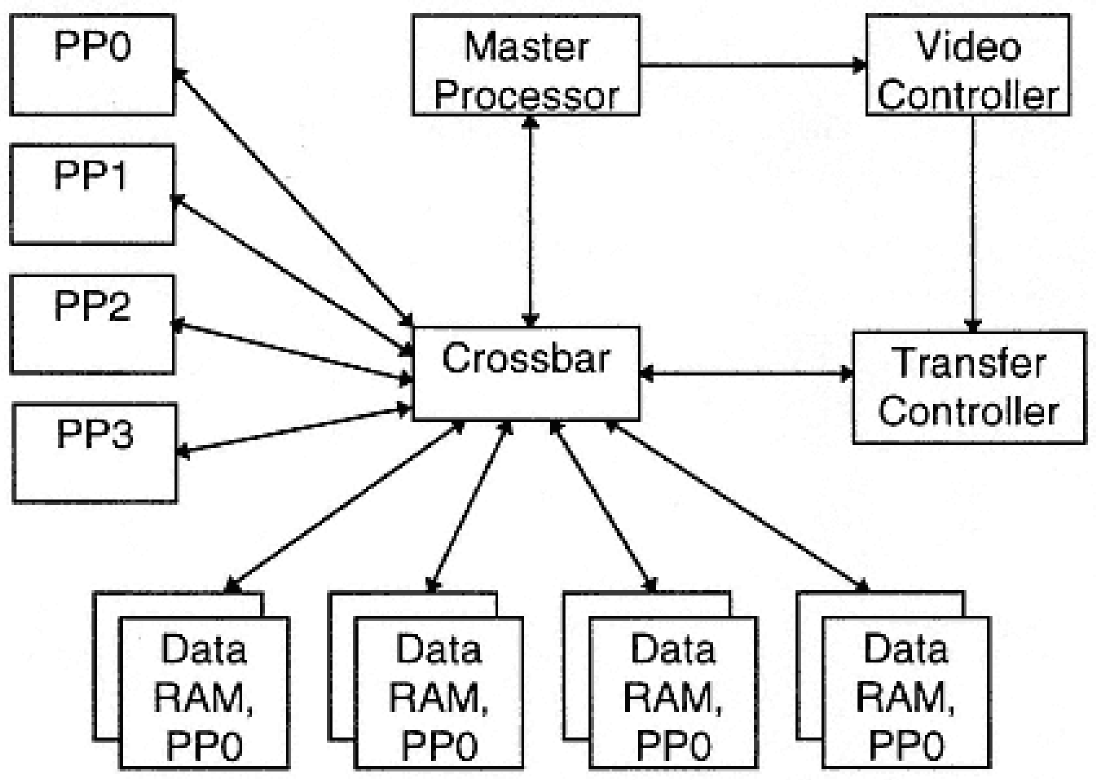


Figure 8.19
Architecture of the TMS 320C80.

Any instruction that is fetched from Program Memory and references Program Memory requires a wait state. Thus instructions that reference Program Memory are cached (Instruction Cache is associative). Calculations are performed between registers. Supported calculations include one tick multiply/add, shifting, and general ALU operations. Address calculations allow indexing of an address base by a programmable value. 3 address calculations are simultaneously active (each instruction chooses an address calculation reference).

The TMS320C80 video controller performs all of the timing operations needed to capture video images. The video controller issues orders to the transfer controller to store the captured video into the memory buffer. The transfer controller supports transfer of packets of 3D blocks of memory, using the concepts of pitch and patches. Scan lines in buffer memory are separated by the pitch, and a collection of scan lines is organized into a patch. A packet can be programmed to communicate a collection of patches, each separated by the patch offset. In the case of frames composed of 480 lines of 720 bytes, eight bytes would be transferred per line with a pitch might of 720, a patch would be composed of eight lines separated by 480 lines, and the entire transfer would be eight patches. This correctly reorganizes eight frames of eight lines of eight bytes in scan-line order into a packed cube of 512 bytes for 3D DCT processing.

Timings for the TMS320C80 are exactly the same as those for the SHARC processor, reported in Table 8.7.

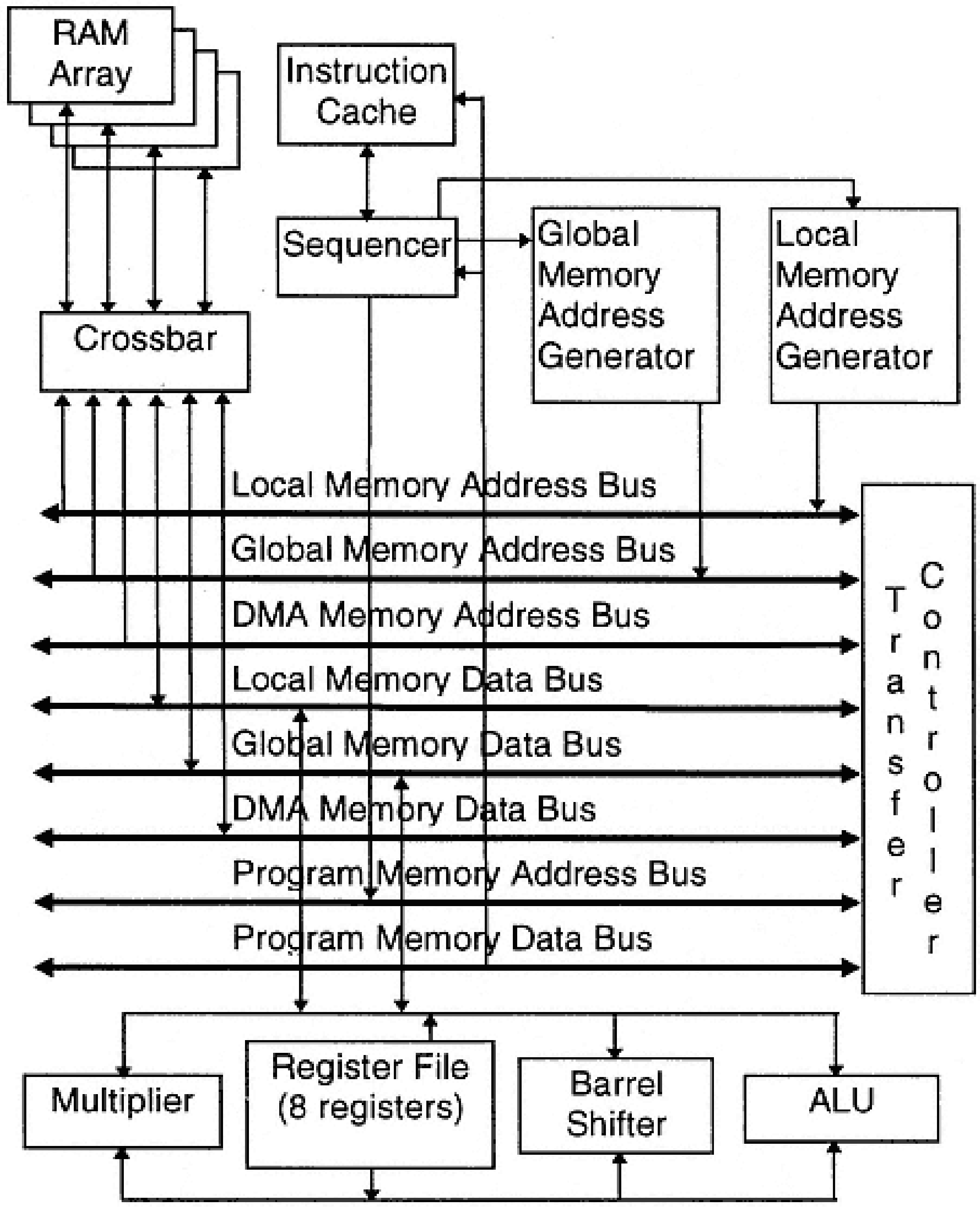


Figure 8.20
Architecture of Pixel Processors TMS320C80.

9— Experimental Results Using XYZ Compression

On the basis of the analysis, performed in Chapter 8, we summarize the complexity of the XYZ algorithm compared to H.261/H/263 and MPEG algorithms for different video formats. Results are shown in Table 9.1 for CIF format, Table 9.2 for CCIR 601 (NTSC) format, and Table 9.3 for HDTV format.

Table 9.1 Complexity of video compression algorithms in MOPS for CIF format (288x352) at 30 frames/sec.

Compression Algorithm	**Encoder Complexity**	**Decoder Complexity**	**Total Complexity**
H.261/H.263	970	200	1,170
MPEG No B Frames	750	100	850
MPEG 70% B Frames	1,120	120	1,240
XYZ	240	240	480

In summary, the XYZ decoder is more complex than H.261/H.263 and MPEG decoders, and its complexity is about 1.5-2 times higher than the complexity of the other two algorithms. However, the complexity of the XYZ encoder, based on a fast 3D DCT algorithm, is superior compared to the other two algorithms. The complexity of the XYZ encoder is about 4-6 times lower than the complexity of H.261/H.263 and MPEG encoders.

The XYZ compression algorithm was implemented on the PC, and on the MasPar parallel computer. Both implementations are described in this Chapter, and timings are shown. Based on these experiments, the final performance of the XYZ compression algorithm are determined.

Table 9.2 Complexity of video compression algorithms in MOPS for CCIR 601 format (480x720) at 30 frames/sec.

Compression Algorithm	**Encoder Complexity**	**Decoder Complexity**	**Total Complexity**
MPEG No B Frames	3,000	400	3,400
MPEG 70% B Frames	4,600	470	5,070
XYZ	980	980	1,960

Table 9.3 Complexity of video compression algorithms in MOPS for HDTV format (1152x1440) at 30 frames/sec.

Compression Algorithm	**Encoder Complexity**	**Decoder Complexity**	**Total Complexity**
MPEG No B Frames	14,500	1,900	16,400
MPEG 70% B Frames	22,000	2,300	24,300
XYZ	4,600	4,600	9,200

9.1— PC Implementation

We have performed a variety of experiments on a PC to demonstrate the XYZ video compression algorithm and evaluate its features. We also compared the XYZ algorithm with the MPEG standard. For these purposes, we have developed a testbed for both XYZ and MPEG video compression techniques, and have implemented both techniques in software.

The testbed features a user-friendly interface. The user can select a video file which is to be compressed, and a set of quantization tables to be used in the experiment. After performing the experiment, which consists of running both the encoder and decoder, the obtained results are written in the report file, while the error file contains the differences between the original and decompressed frames. The user can display each of frames individually, both the original and the decompressed frame, and compare them visually. The user can also view the difference between the original and the decompressed frames, as well as the obtained DCT coefficients.

9.1.1— Demonstration of the XYZ Algorithm

We demonstrate the performance of the XYZ video compression algorithm by applying it to a video clip chosen from the movie 'Dick Tracey'. The clip consists of eight frames with resolution 320x240 pixels. The moving figures consume about 40% of the frame, and the figures move in opposite directions. The clip has been captured and stored in uncompressed form. The quantizer, referred as QT1, is generated from one-dimensional quantizer motivated by JPEG:

$$OneDQ[x] = (0,1,2,3,6,11,20,25) \qquad (9.1)$$

where x=0 to 7.

The 3D quantizer is then developed using the following expression:

$$ThreeDQ[x,y,z] = 5 + OneDQ[x] + oneDQ[y] + OneDQ[z] \qquad (9.2)$$

where x,y, and z are in the range from 0 to 7.

The Normalized Root Mean Square Error (NRMSE) is calculated as:

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{n}(Xi - \hat{Xi})^2}{\sum_{i=1}^{n}(Xi^2)}} \qquad (9.3)$$

where:

Xi are original pixel values,

$\hat{Xi}$ are pixel values after decompression, and

n is the total number of pixels in the 8-frame video sequence.

Figure 9.1 shows the original sequence of 8 frames, the decompressed sequence, and the error between frames. The error frames are multiplied by 16 in order to become visible for evaluation purposes.

The obtained results indicate that the XYZ algorithm can effectively be used for compression of full-motion video. Besides presenting 8 frames of the video, we also played back the decompressed video in a loop. The obtained quality of the video was very good.

9.1.2— Comparison with MPEG Standard

We compared the XYZ compression algorithm with the MPEG standard. Motion estimation in MPEG is performed by 2D logarithmic search and by exhaustive search of a region of 16 pixels wide. First frame (frame 0) is compressed as I frame using MPEG-recommended quantizer. Frame 7 is compressed as P frame using forward motion prediction. Error terms are calculated and encoded using the DCT. Frames 1 to 6 are compressed as bidirectional (B) frames. Motion estimation is done using bidirectional prediction. Four experiments were performed using MPEG: two including error correction with two different search algorithms, and two with no error correction for both search algorithms.

In addition to the previous experiment, reported in Section 9.1.1, we applied XYZ compression for four additional sets of quantization tables, referred as QT2 to QT5 in [WF95]. The quantization tables are selected in such way that QT1 tables contain the smallest coefficients, thus achieving the best quality of the video and the lowest compression ratio. On the other hand, the QT5 tables have the largest coefficients, thus producing the highest compression ratio and the lowest video quality. The results are summarized in Table 9.4. Figures 9.2 shows the XYZ decompressed sequences for additional tow sets of quantization tables, QT2 and QT3, while Figure 9.3 shows the MPEG decompressed video sequences.

Table 9.4 Comparison of XYZ and MPEG video compression algorithms.

Video Compression Technique	**Compression Ratio**	**Normalized RMS Error**	**Execution Time [min]** (8 frames, 320x240)
XYZ (QT1)	34.5	0.079	6.45
XYZ (QT2)	57.7	0.097	6.45
XYZ (QT3)	70.8	0.105	6.45
XYZ (QT4)	101.7	0.120	6.45
XYZ (QT5)	128.1	0.130	6.45
MPEG Logarithmic Search and Error Correction	11.0	0.080	21.35
MPEG Exhaustive Search and Error Correction	15.6	0.080	163.0
MPEG Logarithmic Search and No Error Correction	27.0	0.140	21.35
MPEG Exhaustive Search and No Error Correction	32.9	0.125	163.0

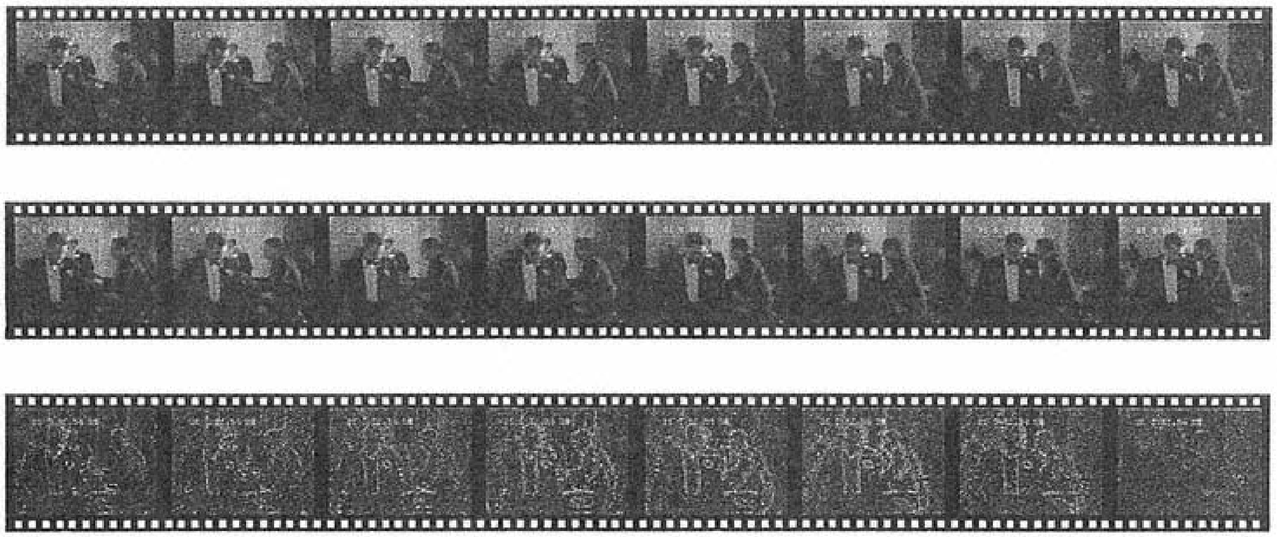


Figure 9.1
Demonstration of the XYZ video compression algorithm: (a) original sequence of 8 frames (240x320),
(b) decompressed sequence using QT1 - compression ratio=34.5, NRMSE=0.079, (c) error between frames multiplied by 16.

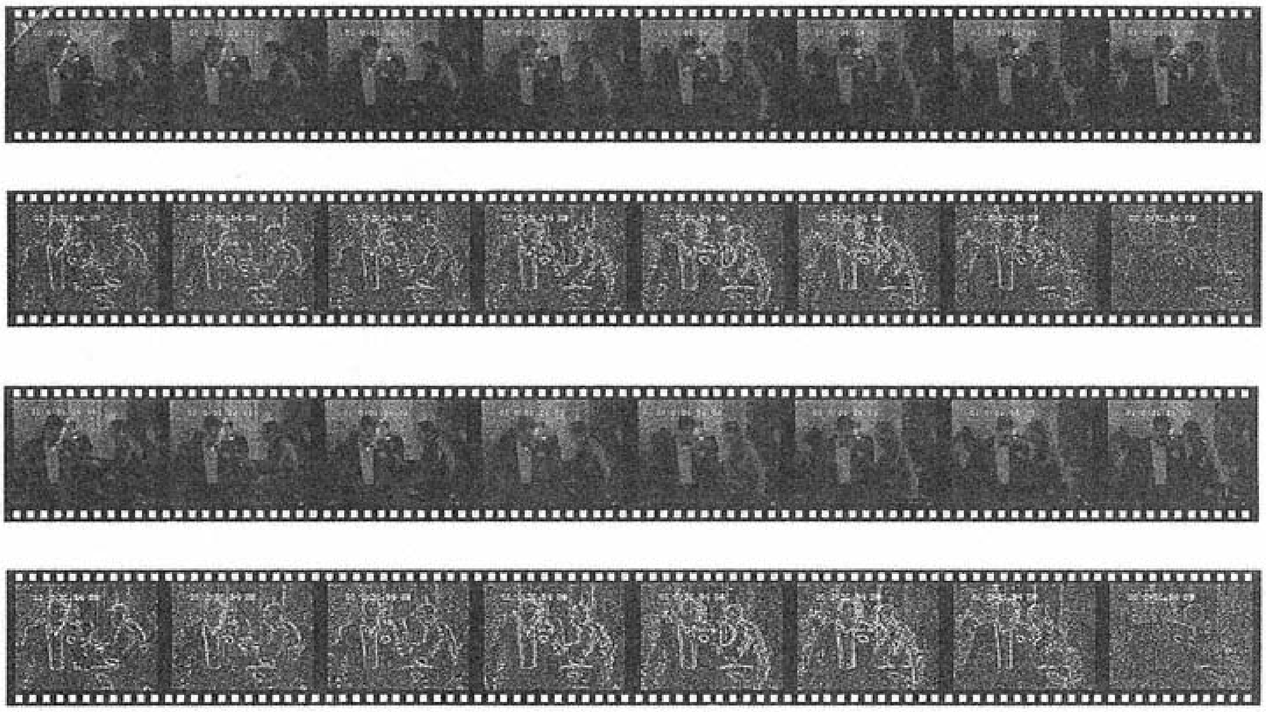


Figure 9.2
The XYZ compression algorithm: (a) Decompressed sequence using QT2, compression ratio=57.7, NRMSE=0.097, (b) error function for case a, (c) decompressed sequence using QT3, compression ratio=70.8, NRMSE=0.105, (d) error function for case c.

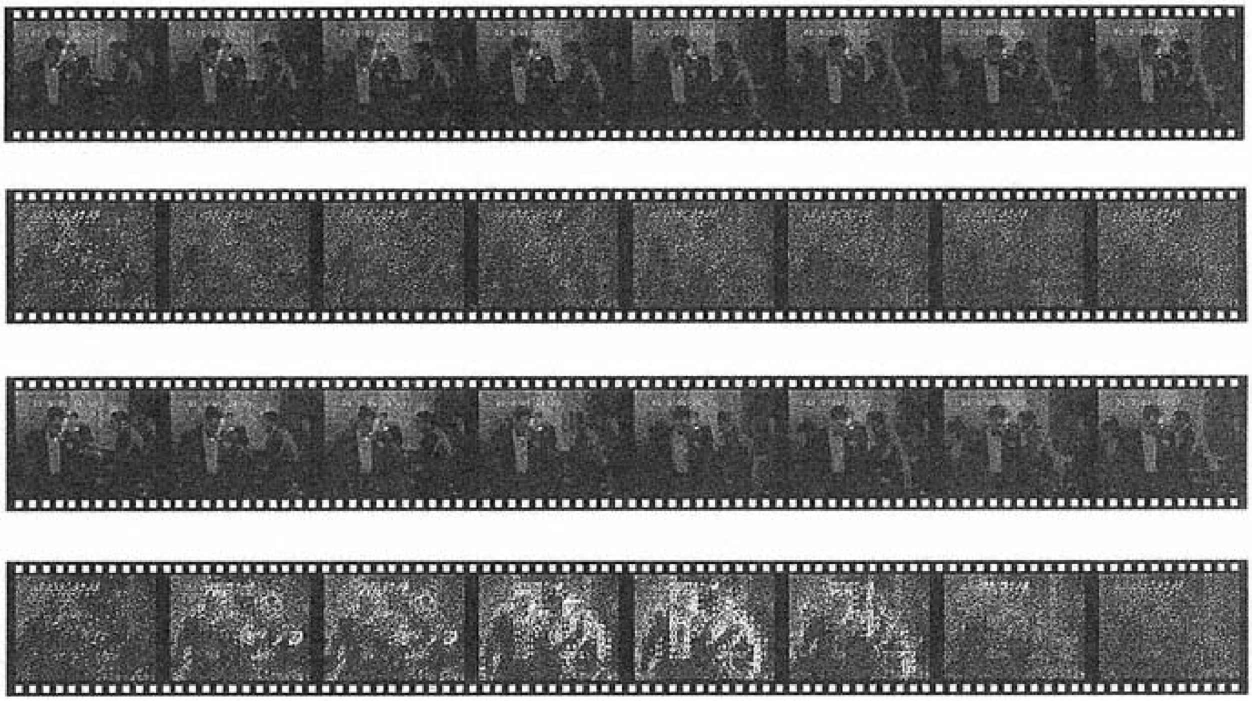


Figure 9.3
MPEG compression algorithm: (a) decompressed sequence using MPEG including B error correction, compression ratio=15.6, NRMSE=0.080, (b) error function for case a, (c) decompressed sequence using MPEG excluding B error correction, compression ratio=32.9, NRMSE=0.125, (d) error function for case c.

Figure 9.4 shows an uncompressed frame drawn from the eight frame sequence, rendered in gray scale for ease of reproduction. The character in the foreground is moving to the reader's right, the character behind him is moving to the left.

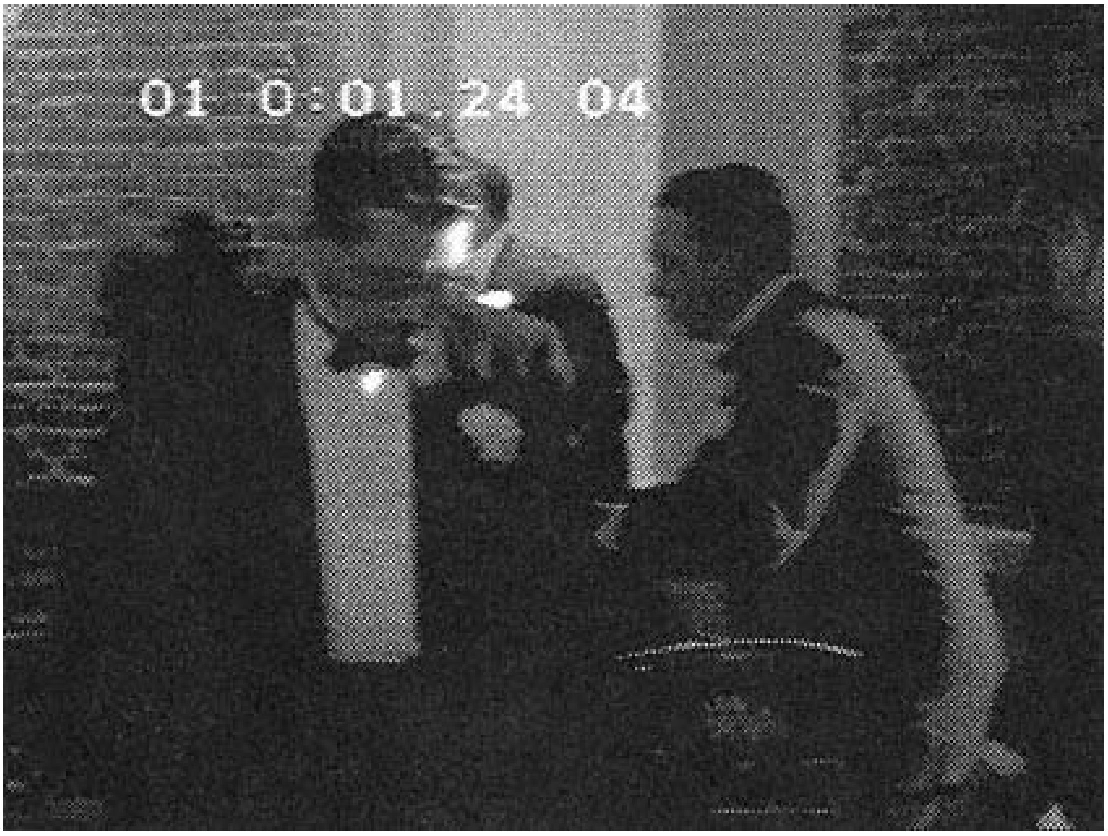


Figure 9.4
Raw captured frame from the movie 'Dick Tracey'.

Figure 9.5 is decoded frame using MPEG compression associated with MPEG-1 recommended quantizers and Huffman coding tables. An exhaustive search for inter-frame motion estimation was used with p=8. The same frame is displayed – in this case a B-frame. The compression ratio for the sequence of frames is 11:1, and the NRMS error is 0.08.

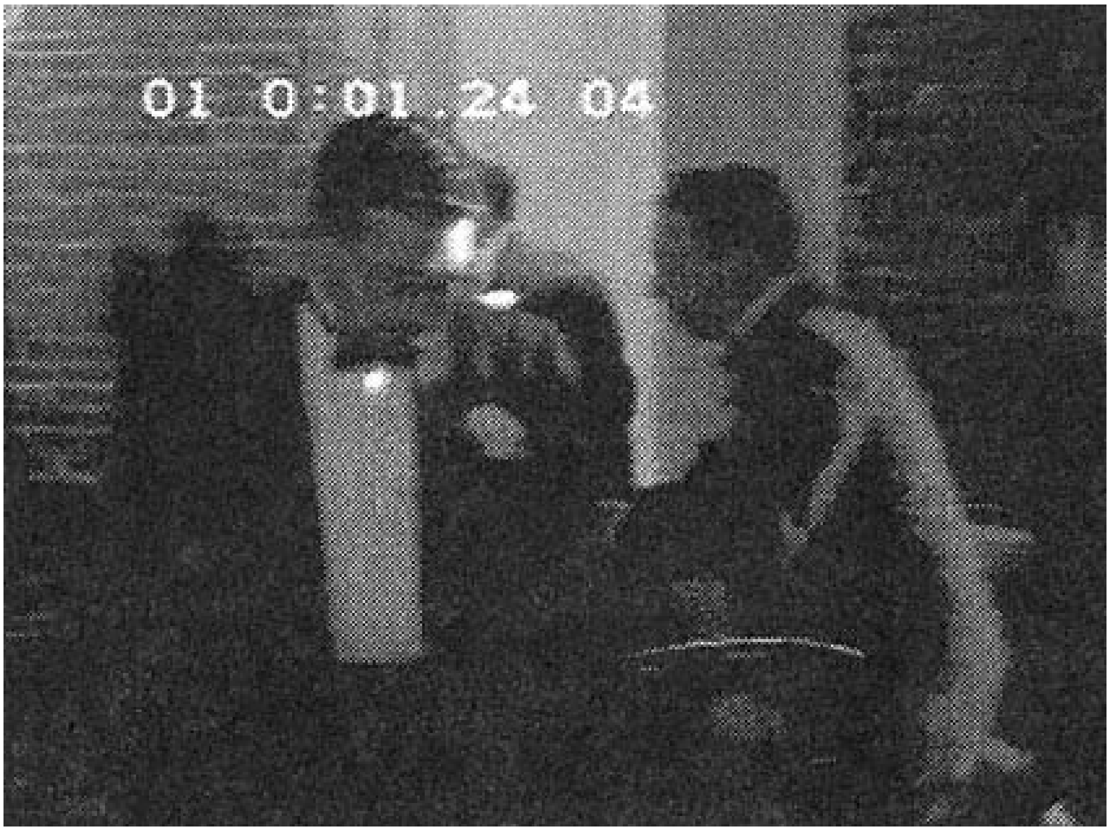


Figure 9.5
MPEG-decompressed frame, Compression ratio=11, NRMSE=0.08.

Figure 9.6 displays the same frame after MPEG compression without error residue. That is, simple motion estimation is used on the B frames. The compression ratio for this sequence is 27:1, with NRMSE of 0.14. Experience indicates that NRMSE values above 0.08 result in objectionable artifacts.

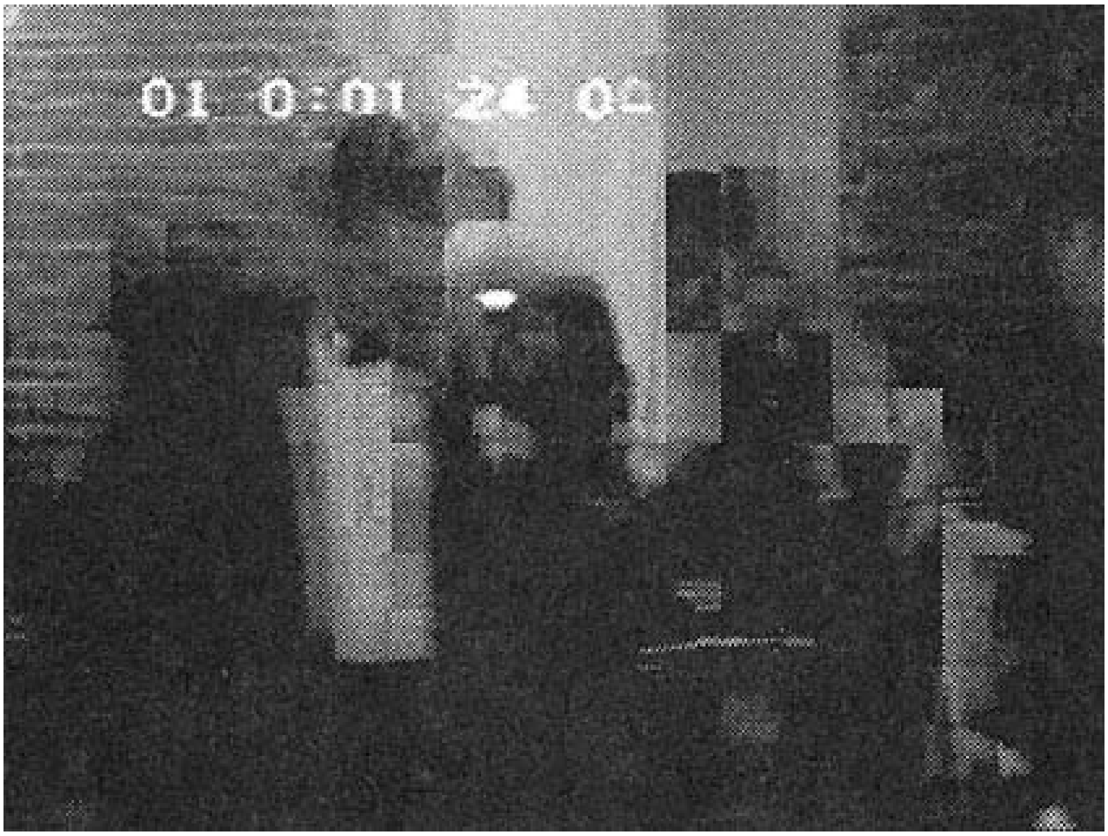


Figure 9.6
MPEG-decompressed frame, motion estimation only.
Compression ratio=27, NRMSE=0.14.

Figure 9.7 displays the same frame after compression with the XYZ compression algorithm. The MPEG quantizers were used (after extension into three dimensions), and the MPEG Huffman tables were used for encoding. This sequence is compressed at 34.5:1 with NRMSE of 0.079.

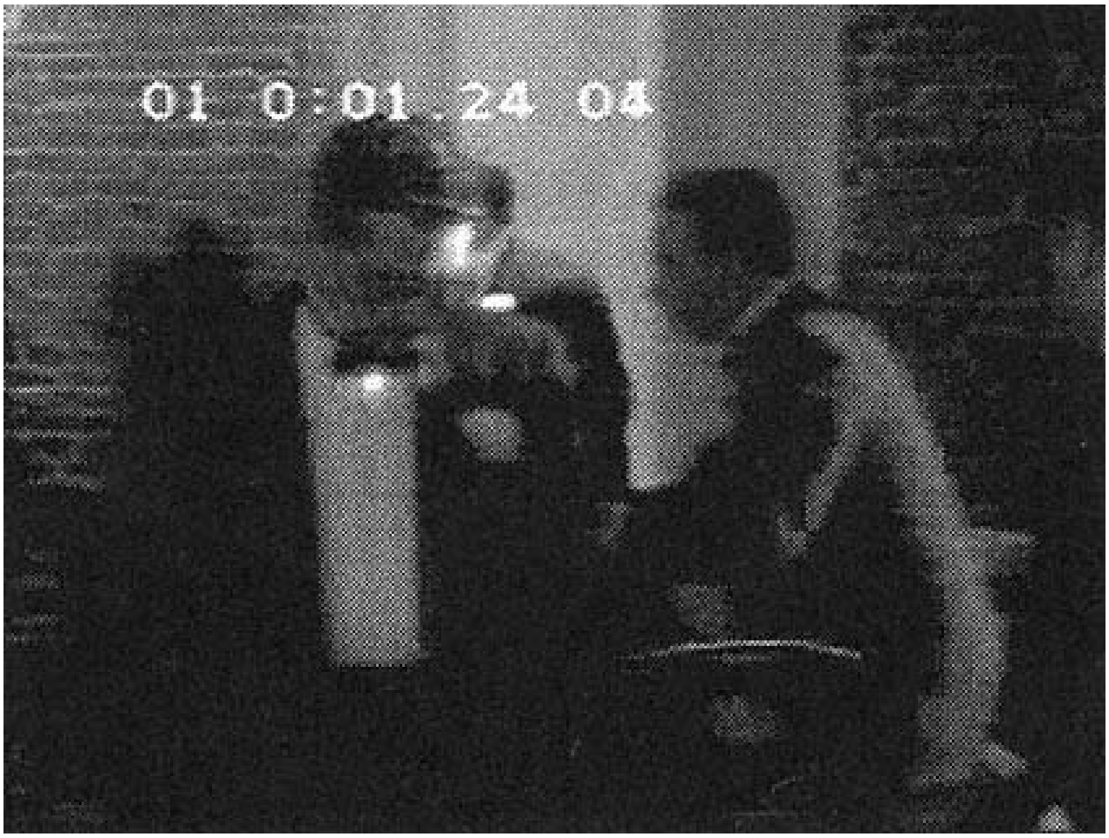


Figure 9.7
XYZ decompressed frame, Compression ratio = 34.5, NRMSE=0.079.

The following conclusions can be made [WF95]:

• The XYZ video compression algorithm gives significantly better compression ratios than the MPEG algorithm for the same quality of video. For example, XYZ result 1 and MPEG result 1 (see Table 9.3) give similar NRMS errors (0.079 and 0.08, respectively) and reconstructed sequences show similar image quality. However, the XYZ algorithm provides much higher compression ratio (34.5 versus 15.6).

• For similar compression ratios, the XYZ video compression algorithm gives much better quality of the decompressed video than the MPEG algorithm. For example, XYZ result 1 and MPEG result 4 (see Table 9.3) give similar compression ratios (34.5 and 32.9, respectively), but XYZ algorithm gives much better quality (NRMS error for XYZ is 0.079, while for MPEG is 0.125).

• The obtained results suggest that XYZ video compression algorithm is faster than the MPEG algorithm (including both compression and decompression).

• The XYZ results 4 and 5 (Table 9.3) suggest that very high compression ratios (greater than 100) can be achieved using the XYZ video compression algorithm, while the NRMS error is still kept relatively small (in the range from 0.120 to 0.130). In this case, for videos with the fast camera movement the visual artifacts are significant. However, the algorithm gives very good results for videos with little movements, which is the case in videoconferencing applications.

• Finally, the MPEG technique is based on three different algorithms: one for I, another for P, and the third algorithm for B frames. MPEG is also asymmetrical algorithm, requiring a complex encoder and a simple decoder. On the other hand, the XYZ technique applies only one algorithm for all frames and is a symmetrical algorithm requiring the same complexity for both encoder and decoder. This fact is beneficial for VLSI implementation of the algorithm, discussed in Chapter 8.

9.1.3— The Sensitivity of the XYZ Algorithm to Various Video Effects

In the previous sections, we demonstrated the XYZ video compression algorithm by applying it to a single 8-frame clip, which includes the large area of motion. In this section, we present the results obtained when applying the XYZ algorithm to various video clips, which show different video effects, such as camera break, camera panning and zooming, and fast camera movement. Pan and zoom sequences are chosen from the movie 'Total Recall', another pan and camera break sequences are from the movie 'Interview with a Vampire', and a fast camera movement sequence is from the movie 'Interceptor'. These sequences are compressed using the XYZ algorithm and the MPEG-inspired quantization tables, given by the equation 9.2. The results are summarized in Table 9.5.

Table 9.5 XYZ algorithm applied to various video effects.

Movie Clip Video Effect	**Compression Ratio**	**Normalized RMSE Error**
Dick Tracy Typical Motion	34.5	0.079
Interview with the Vampire Camera Break	32.5	0.087
Interview with the Vampire Camera Panning	26.1	0.085
Total Recall Camera Panning	17.5	0.049
Total Recall Camera Zoom	23.9	0.042
Interceptor Fast Motion	26.0	0.025

Note from Table 9.5 that the XYZ algorithm shows very little sensitivity to camera break achieving almost the same compression ratio and NRMSE as in the case with no camera break. For illustration purposes, Figure 9.8 shows two frames from a sequence which includes a camera break - the original frames and frames after decompression.

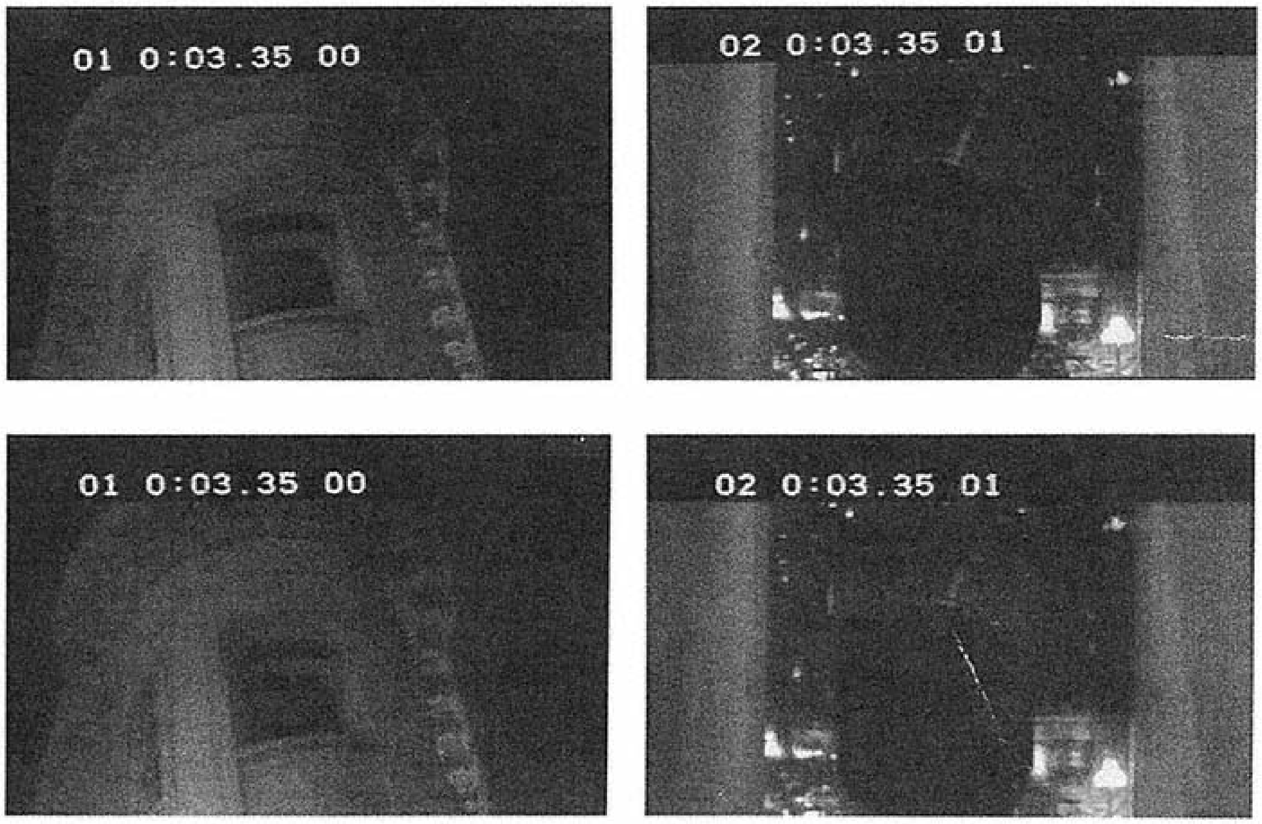


Figure 9.8
Applying the XYZ video compression algorithm to a sequence with camera break: (upper row): original frames, (lower row): frames after decompression., Compression ratio=26, NRMSE=0.025.

However, in the other three cases: camera panning, camera zooming, and fast camera movement, the XYZ algorithm shows a decrease in the compression ratio compared to a normal sequence without these effects, which is an expected result.

In Section 9.3, we present the XYZ algorithm using quantization based on human visual factors, which is capable of achieving high compression ratios in all these cases.

9.2— MasPar Implementation

The full adaptive XYZ compression algorithm, described in Chapter 8, was implemented on the MasPar parallel computer. The MasPar was considered a good benchmark to predict the performance of DSP implementations, as modern DSPs are about 1000 times more powerful than the MasPar processors. This was also a good testbed for longer test video sequences (5 seconds or more).

9.2.1— Architecture of the MasPar

The MasPar is a massively parallel SIMD (single instruction, multiple data) computer. In SIMD computers, all Processing Elements (PEs) execute the same instruction simultaneously, but operate on different data. Instructions are decoded and broadcast by the Array Control Unit (ACU). The general architecture of the MasPar is shown in Figure 9.9.

The ACU is a general-purpose processor with its own data and instruction memory. Programs are loaded and executed on the ACU. When parallel instructions are

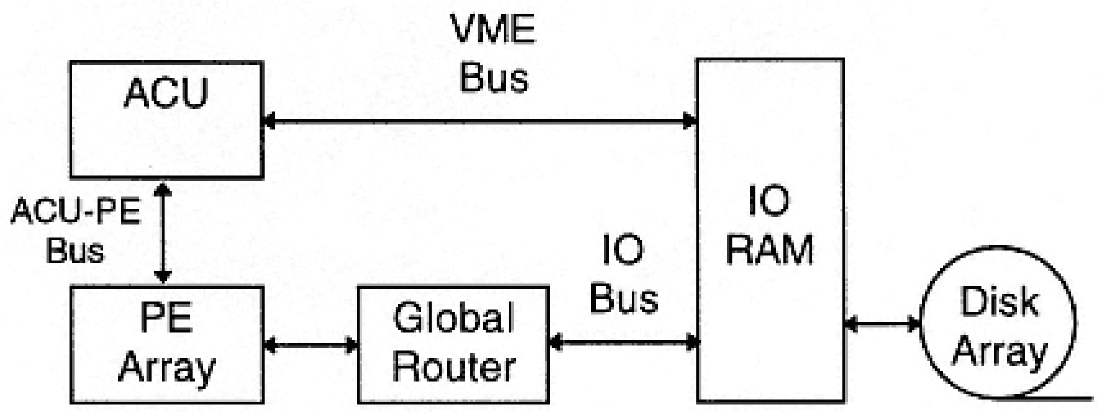


Figure 9.9
The architecture of MasPar.

interpreted, they are broadcast to the PE array. All code was written in MPL, a parallel version of ANSI C. In MPL, data that is declared "plural" is stored on the PEs. Operations on plural data generate parallel instructions which are executed on the PEs.

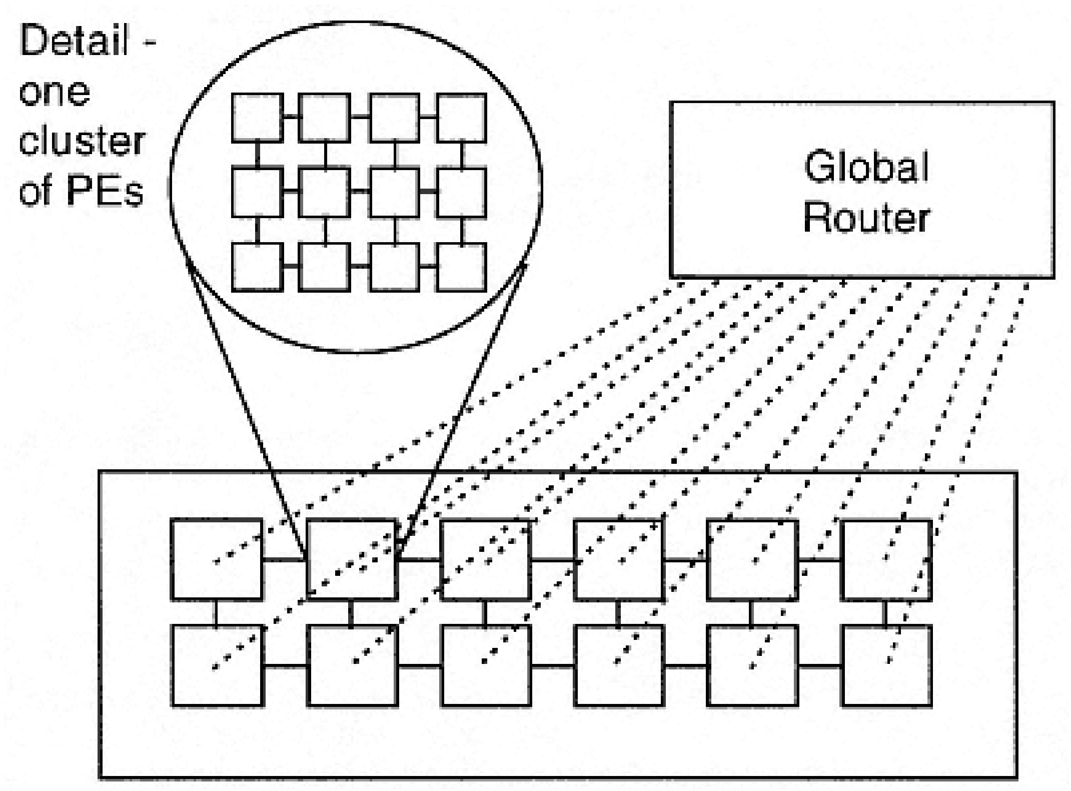


Figure 9.10
Communications of the PEs in MasPar computer.

The PE Array is organized into matrices of 32 PEs called clusters. Each cluster is a source or destination for global routing, supporting communication between arbitrary processors or to/from the ACU. In addition, a mesh architecture ("X-net") connects adjacent processors. Communications of the PEs is illustrated in Figure 9.10.

The mesh connection supports higher bandwidths than does the more general global routing connection. Measured execution times on the MasPar computer are summarized in Table 9.6.

Table 9.6 Measured execution times (in milliseconds) on the MasPar computer.

	Loop	**Integer Multiply**	**Float Multiply**	**Integer Add**	**Integer Multiply**
ACU	.002	.006	.024	.004	.017
PE	.002	.022	.030	.015	.024

9.2.2— Algorithms on the MasPar

Several standard problems must be solved in order to implement adaptive XYZ compression on the MasPar computer. In particular, data on the ACU has to be distributed to the PEs, data on the PEs has to be "reduced" (collected) into the ACU, and data on the PEs has to be sorted. Fast algorithms are developed for each of these problems.

Distribution of individual data from the ACU to the PEs is done through the global router. The ACU simply sends each data element to each PE (the MPL command **proc**). However, a fast distribution strategy exists for broadcast of the same data item to all PEs, as illustrated in Figure 9.11.

The X-Net copy command (the MPL command **xnetc**) transmits data along the xnet, depositing copies of the data at each PE on the route. Since the PE array is organized as a two-dimensional array, the broadcast can be accomplished with only two xnet transfers. Broadcast from the ACU is then accomplished by setting the data item into PE#0, and performing an xnet copy transfer to the last processor on row 0. Then each processor in row #0 performs an xnet copy to the corresponding processor in the last row along each column.

Reduction of data across PEs is done with the X-Net "pipe" send (the MPL command **xnetp**). Pipe sends are high-speed point-to-point sends (without copies to the intermediate nodes), as shown in Figure 9.12. After each transfer of data from one PE to another, reduction of the data is performed on the target PE. For example, in the case of pixel averaging, the reduction operation is accumulation.

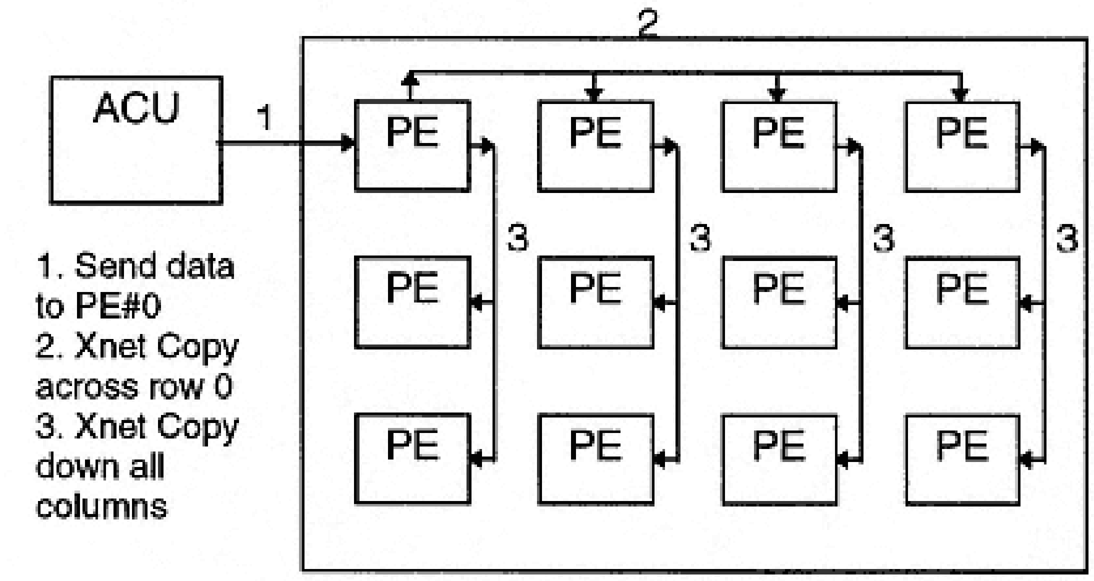


Figure 9.11
Broadcast of data from the ACU to PEs.

Performance is gained by increasing the length of each successive transfer. The first transfer is to the nearest neighbor, the second transfer reduces every fourth PE, etc. At the end of the process, the reduced data resides in PE#0. This data can then be transferred to the ACU.

Sorting of data is done with a parallel version of the bubble sort, as illustrated in Figure 9.13. This technique was chosen because of the relatively small number of data items to be sorted (512), and its ease of implementation.

The PEs are logically organized as a line of processors. At each even step in the sorting process, each even numbered processor (in the logical organization) compares its data value to the processor on the right. The larger value is sent to the right. At each odd step in the process, each odd numbered processor compares its data value to the processor on its right. At the end of N steps, N data items are sorted.

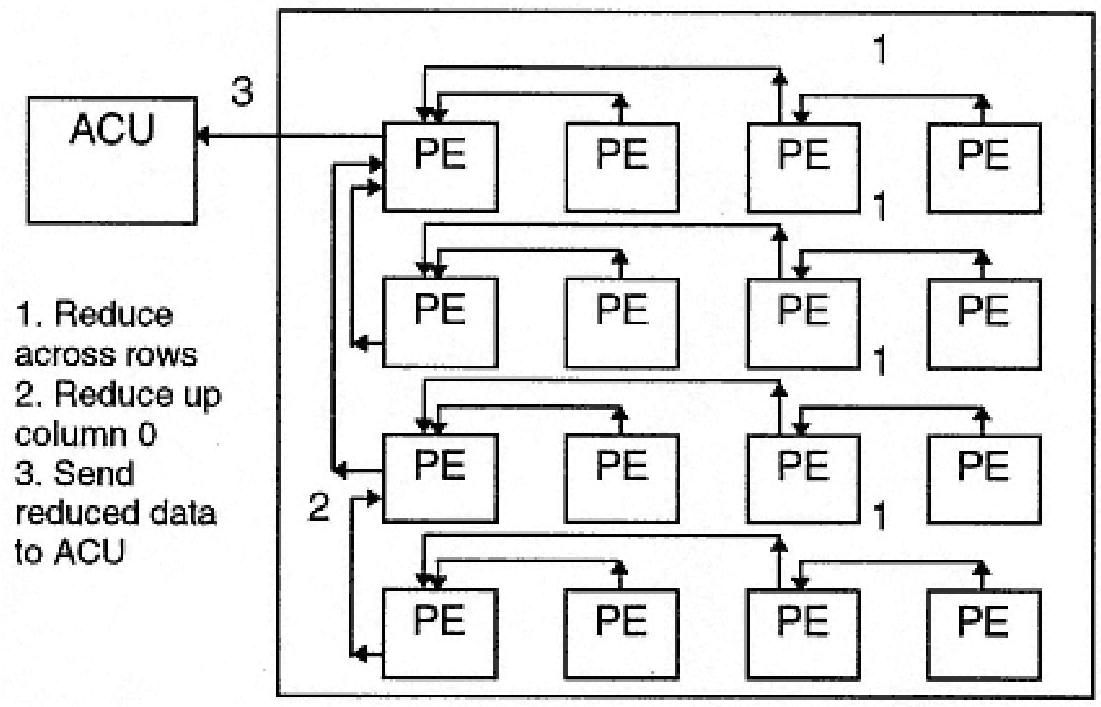


Figure 9.12
Reduction of data from PEs to the ACU.

9.2.3— Adaptive XYZ Compression on the MasPar

The full adaptive XYZ video compression algorithm was implemented on the MasPar computer. Timing of the XYZ adaptive compression algorithm on MasPar is reported in Table 9.7. Full-motion video clips used by the MPEG committee to assess the MPEG compression algorithm were used to benchmark the XYZ compression algorithm. Adaptive quantization was applied based on returning a desired NRMSE figure. The quantizers were chosen to minimize overall mean square error. This work duplicates earlier work [RPR77]. The resulting compression ratios were disappointing, and it was not possible to arrive at the compression ratios advertised in that work.

Adaptive Huffman coding can be expected to increase the compression ratio without decreasing the visual quality. However, adaptive Huffman coding is computationally complex. This technique was applied to CCIR clip "Susie", and to MPEG-2 clips "Carousel", and "Cheerleaders".

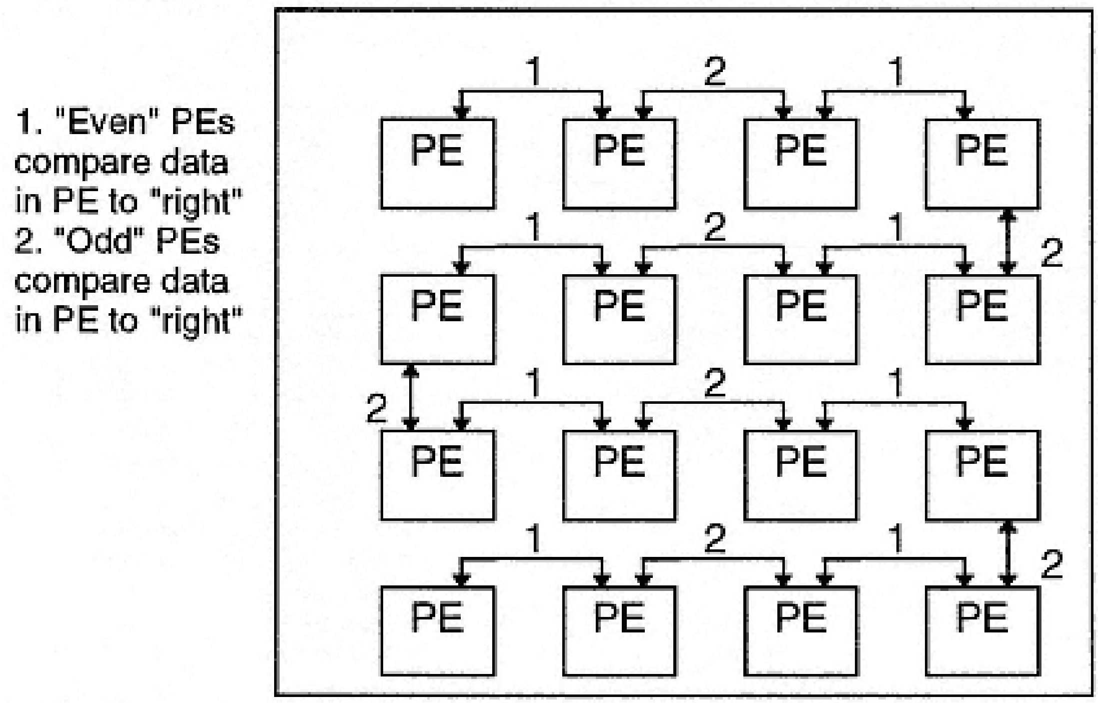

1. "Even" PEs compare data in PE to "right"
2. "Odd" PEs compare data in PE to "right"
1
2
1
2
PE

Figure 9.13
Sorting data on PEs.

Somewhat surprisingly, the results of adaptive Huffman coding actually show a reduction in compression ratio when compared to the MPEG-1 coding tables. This is due to the different strategy used by the MPEG-1 tables (the ESC code is not supported), and the different maximum length of the MPEG-1 code words (16 bit vs. 12 bit).

It seems clear that then benefits of adaptive compression are outweighed by the computational expense of this procedure. It seems clear that non-adaptive compression will give 50% to 75% of the compression ratios at 1/4 the computational cost of adaptive compression. Except in the most demanding environments, the advantages of real-time non-adaptive compression will probably outweigh the cost of adaptive compression.

This combined failure of adaptive quantization and adaptive Huffman coding to generate high compression ratios, coupled with the prohibitively high computational cost of the adaptive algorithms, lead to abandonment of the adaptive approach. This leaves the human visual acuity experiments as the last hope for developing a truly reliable and competitive compression algorithm.

Table 9.7 Timing of the XYZ adaptive compression on MasPar.

Function	CPU Time	Comments
Estimated Adaptive Compression Time	9 seconds	
Estimated Non-adaptive Compression Time	2 seconds	
Measured Overall Adaptive Compression Time	8.9 seconds	Time to read/write, encode and decode, compute adaptive quantizers for ¼ sec. Note that all numbers are for RGB (4:4:4) data. Real-life applications would use YUV12 (4:2:0) data, and should run twice as fast, and get double the compression ratios.
Read	1.4 seconds	Read in pre-formatted data, 8x8x8 blocks distributed to all processors. This corresponds to an actual throughput rate of only 15 MB/s, the maximum rating for the MasPar disk array. FAU's MasPar lacks overlapped file I/O capability.
Average Pixel	.047 seconds	Time to compute average pixel across all processors. This indicates the efficiency of the mesh interconnection.
Forward DCT	1.126 seconds	Fast Forward DCT calculation
Calculate Quantizers, Sort Order	1.030 seconds	
Entropy Coding	1.477 seconds	Does not include actual encoding time (only size of encoded data is collected)
Inverse DCT	1.149 seconds	Fast Inverse DCT
Write Frame Data	1.107 seconds	Writes data in same pre-formatted format
Overhead	App. 2 seconds	Overhead includes measurement of actual NRMSE, some loop control and initialization code

8.3— Non-adaptive XYZ Compression

The thrust of the work was then changed to emphasize quantization based on Human Visual Factors. Threshold visibility is defined as the magnitude of the DCT component at which artifacts are first visible. This test was run by comparing the DCT component to be tested against a window containing the DCT component with magnitude 0. The visibility was taken to be the greatest magnitude at which the windows could not be differentiated (at the 6X viewing distance). The visibility of DCT coefficients was tested at a distance of 6X the screen size. Threshold visibility was compared to relative visibility. The resulting figures are summarized in Table 9.8.

Table 9.8 Measured threshold visibility of DCT coefficients.

	Y=0	Y=1	Y=2	Y=3	Y=4	Y=5	Y=6	Y=7
X=0, T=0..7	1	0	17	32	67	49	209	208
	2	0	14	41	70	113	138	258
	0	22	27	62	122	124	284	254
	0	21	38	111	81	176	570	216
	1	50	51	175	114	277	450	242
	32	57	86	227	129	289	403	263
	33	61	131	179	150	176	310	227
	32	64	159	166	132	155	236	172
X=1, T=0..7	16	28	41	89	196	260	255	220
	17	46	39	91	172	200	175	205
	35	55	62	115	219	225	235	235
	45	92	108	166	215	225	250	225
	75	114	114	126	222	250	235	230
	110	82	135	176	228	235	230	230
	41	75	139	207	238	240	230	225
	71	171	179	152	165	200	195	210
X=2, T=0..7	17	58	72	88	80	130	190	240
	16	71	85	89	130	150	155	160
	28	113	115	145	160	205	185	170
	25	170	165	195	175	230	240	220
	43	160	225	245	175	285	295	240
	68	225	215	885	230	265	270	220
	103	220	205	235	225	220	210	205
	125	185	170	150	140	150	180	180
X=3, T=0..7	27	49	86	115	145	225	155	235
	13	67	71	155	185	150	180	165
	30	110	110	210	255	260	250	240
	37	170	135	270	260	370	335	330
	57	215	185	325	475	425	435	280
	59	295	205	380	355	375	380	255
	135	260	215	250	260	245	280	250
	200	175	195	155	160	210	210	215
X=4, T=0..7	47	67	72	180	345	365	540	245
	52	100	86	165	190	175	185	200
	46	145	150	225	255	225	245	260
	62	125	155	290	365	360	330	330
	74	220	235	280	675	1000	610	285
	84	175	260	285	355	385	360	330
	88	240	235	265	255	270	265	270
	93	200	190	185	195	200	205	205
X=5, T=0..7	65	200	130	295	365	670	625	295
	80	200	170	225	240	190	225	230
	105	240	235	305	305	280	315	295
	145	345	350	450	385	460	450	345
	185	364	415	805	885	875	560	385
	255	355	460	405	400	435	450	380
	285	285	320	315	275	305	315	325
	225	220	225	225	230	230	230	225
X=6, T=0..7	140	285	325	425	385	585	560	240
	95	220	225	220	210	180	205	235
	175	320	295	330	320	260	320	285
	270	330	435	455	435	375	425	360
	315	360	545	655	615	640	545	325
	335	355	425	390	420	425	400	330
	310	325	285	260	275	270	285	280
	215	210	215	185	215	205	225	185
X=7, T=0..7	145	315	360	380	385	385	420	330
	250	220	265	260	280	255	240	225
	270	315	290	315	355	335	330	285
	370	365	335	345	430	440	365	355
	410	345	360	315	440	450	410	365
	425	340	345	335	415	455	430	370
	355	300	350	325	360	320	335	355
	265	250	260	225	235	235	235	250

Similarly, the relative visibility was tested by putting up a window at a reference amplitude, and displaying a second window at another amplitude. The amplitude at which the windows can be consistently differentiated was recorded.

The results showed a surprising trend - while the threshold visibility was strongly related to the frequency of the DCT component in all three dimensions, relative visibility was virtually independent of time, and only slightly related to spatial frequency. The primary relationship proved to be between intensity of the reference frame and relative visibility coefficient (about 25% of the intensity proved visible).

Thus a non-linear quantizer was used to model human visual acuity. The first quantizer step was taken to be a function of the threshold visibility (plus 1). Subsequent steps were taken, for these experiments, to be a uniform step size. This size was chosen differently in the experiments.

The following series of experiments were performed:

• **EXPERIMENT 1**. First step is threshold visibility+1. Second and following steps are taken to be the minimum of the first step and 20.

• **EXPERIMENT 2**. First step is maximum of threshold visibility+1 and 8. Second and following steps are taken to be the minimum of the first step and 20. This experiment is expected to factor in relative insensitivity of the human eye to large areas of constant color.

• **EXPERIMENT 3**. First step is maximum of (threshold visibility+1)/2 and 4. Second and following steps are taken to be the minimum of the first step and 10.

This is expected to create artifacts invisible past 3X viewing distance. The video clips selected for these tests were:

• **Susie**. A test sequence of 150 frames released by the CCIR was selected as typical of "talking head" video sequences.

• **Cheerleaders**. A test sequence of 150 frames released by the MPEG-2 committee was selected as typical of motion video sequences.

• **Carousel**. A test sequence of 150 frames released by the MPEG-2 committee was selected as typical of fast, large-scale motion.

• **Dick Tracy**. A test sequence of 8 frames showing motion of two bodies on the screen was selected as typical of motion video.

• **Total Recall**. A test sequence of 8 frames was taken as typical of zoom in motion videos.

• **Interview with the Vampire**. A test sequence of 8 frames was taken as typical of a scene break in motion videos.

• **Vampire Interview with the Vampire**. A test sequence of 8 frames was taken as typical of camera panning in motion videos.

• **Interceptor**. A test sequence of 8 frames was taken as typical of fast motion in motion videos.

The results are summarized in Table 9.9.

Table 9.9 Results of the XYZ video compression using Human Visual Factors-based based quantization.

VIDEO CLIP	Experiment #1 Compression Ratio / NRMSE	Experiment #2 Compression Ratio / NRMSE	Experiment #3 Compression Ratio / NRMSE
Susie	64.7/0.053	109.5/0.054	71.0/0.049
Cheerleaders	33.1/0.102	40.7/0.102	27.1/0.086
Carousel	38.1/0.209	47.6/0.209	29.3/0.175
Dick Tracy	59.4/0.131	88.5/0.131	56.5/0.112
Total Recall	50.1/0.078	69.8/0.079	43.9/0.066
Vampire/Break	57.5/0.150	83.8/0.150	53.0/0.125
Vampire/Pan	49.8/0.151	69.7/0.152	44.8/0.128
Interceptor	54.9/0.047	74.9/0.047	47.2/0.039

The accompanying figures 9.14 and 9.15 show frame 3 (the frame most susceptible to artifacts) of sequences Susie and Cheerleaders, respectively.

The 5-second clip of 'Susie' was compressed in about one minute on the MasPar, suggesting it may be compressible close to real-time on the TMS32080. The compression ratios of 64.7, 109.5, and 71.0 include YUV sub-sampling of RGB data (a 2:1 factor). A videotape of the returned data showed virtually no artifacts at about 6X the viewing distance.

This 5-second clip of 'Cheerleaders' was also compressed in about one minute on the MasPar. The compression ratios attained for the three experiments were 33.1, 40.7, and 27.1. A videotape of the returned data has been mistaken for the uncompressed clip at about 6X the viewing distance.

In summary, the XYZ algorithm, based on the human visual quantization, generates excellent visual results while requiring only calculations of the non-adaptive procedure.

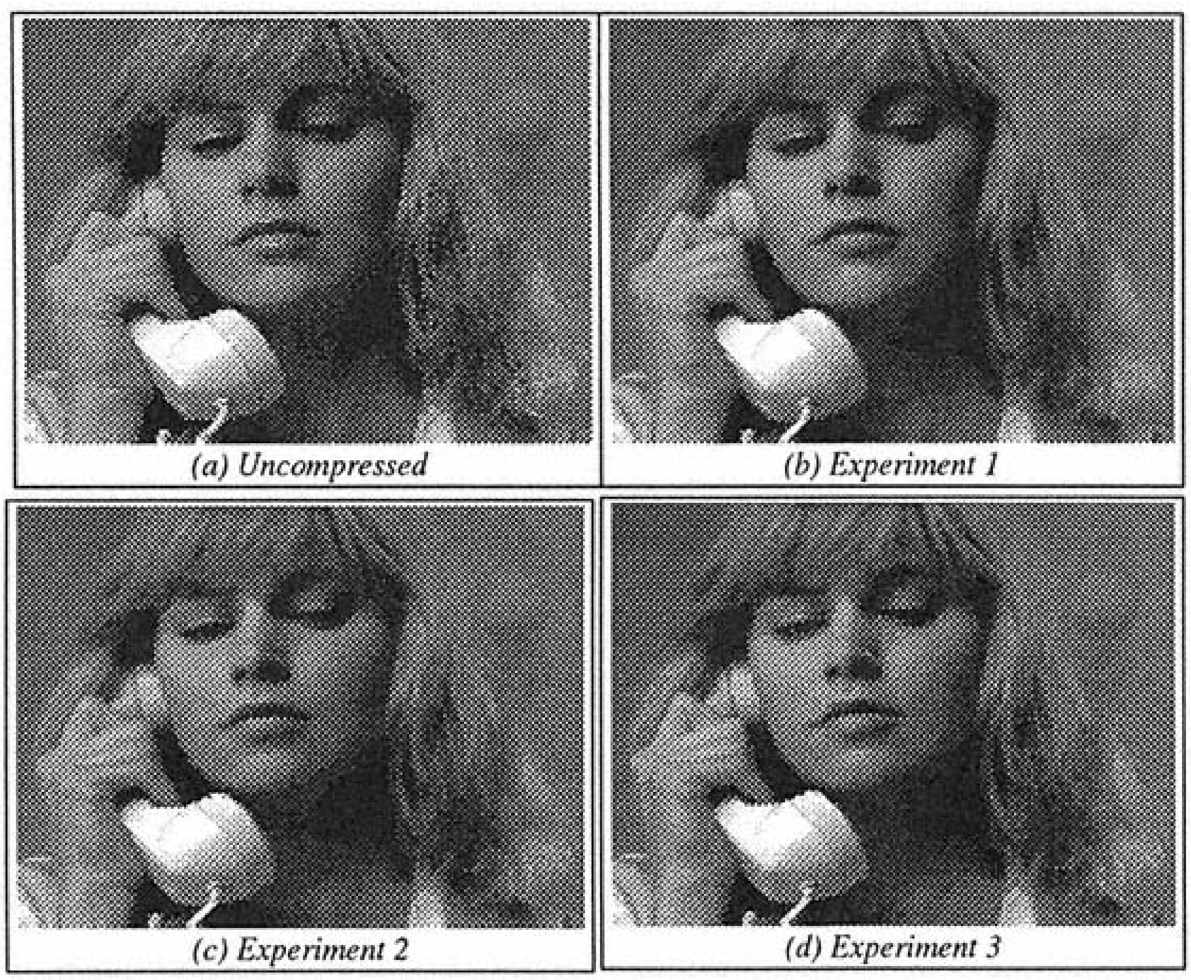


Figure 9.14
Frame 3 of 'Susie' video clip. Compression ratios obtained in three experiments are: 64.7, 109.5, and 71.0, respectively.

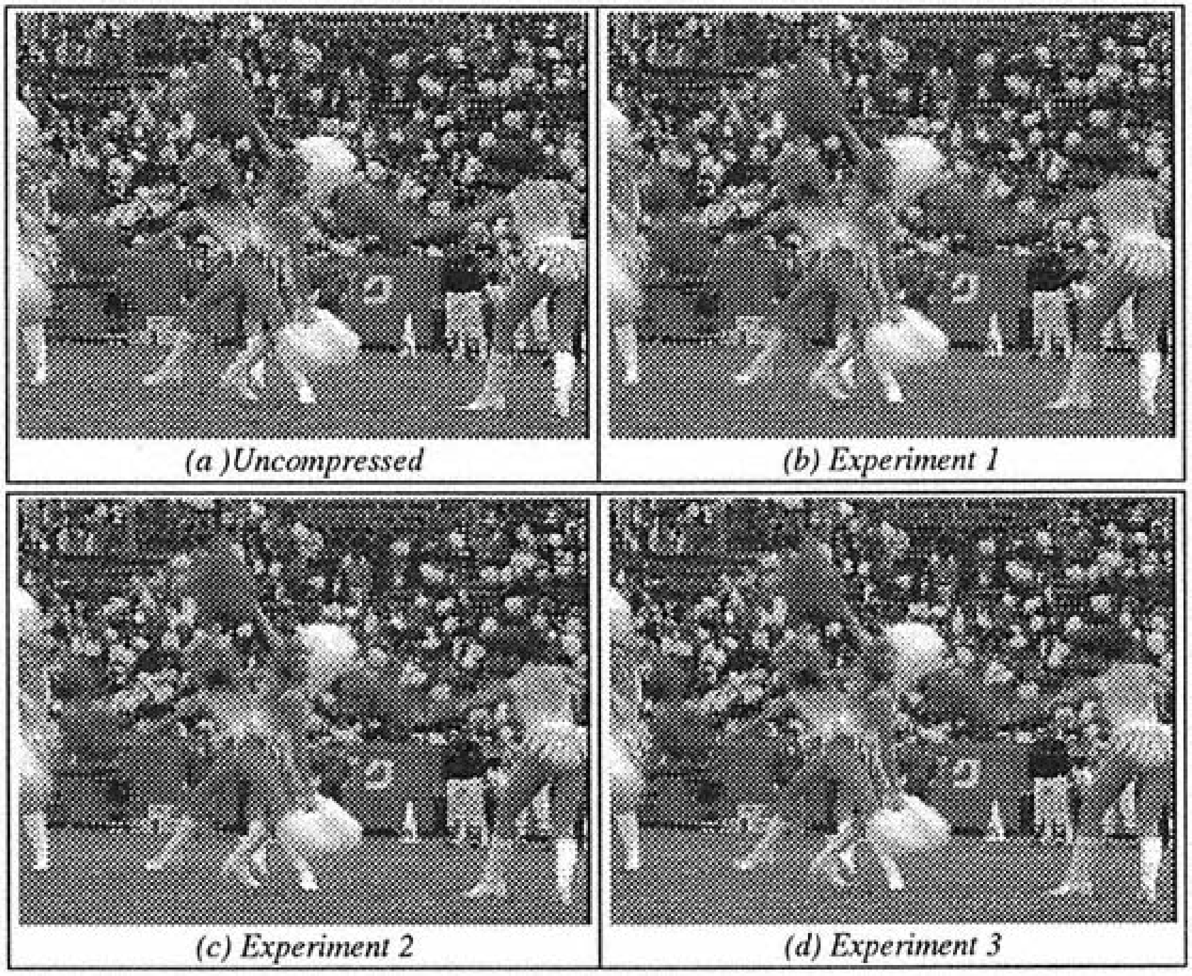


Figure 9.15
Frame 3 of 'Cheerleaders' video clip. Compression ratios obtained in three experiments are: 33.1, 40.7, and 27.1, respectively.

10— Conclusion

XYZ video compression compares favorably with other compression algorithms. Compression ratios exceed those of other algorithms, and compression times are comparable to other algorithms capable of high compression ratios. The XYZ encoder complexity is significantly lower than the complexity of the H.261/H/263 and MPEG algorithms (2.5 to 5 times), due to the fact that no motion estimation is necessary. On the other hand, the XYZ decoder complexity is about 2 times higher compared to these tow algorithms. Table 10.1 summarizes the comparison of XYZ compression with other popular video compression algorithms.

Table 10.1 Rough comparison of popular video compression algorithms.

ALGORITHM	Average Compression Ratio	Relative Encoder Complexity	Relative Decoder Complexity
XYZ	75:1	2	2
H.261/H.263	50:1	5	1
MPEG	30:1	10	1
Wavelet	20:1	1	1
MJPEG	10:1	1	1

The contributions of this text include:

1. The XYZ multimedia compression algorithm has been added to the literature. This book has justified the algorithm and explored associated issues. This contribution has made possible a fast, high-quality compression strategy.

2. The proposed XYZ algorithm is a real-time algorithm. Notably, compression takes place in real-time and does not require off-line processing. The expected time to perform this compression compares with the decompression time of other algorithms featuring lower compression performance. Thus the algorithm enables high-quality video teleconferencing.

3. The algorithm takes advantage of Human Visual System features to generate highest-quality playback. Compression ratios, superior to those of MPEG, are reached with equivalent visual distortion.

4. The compression ratios for the XYZ algorithm is extremely high. Compression ratios are about 2-3 times those of MPEG at the same bit rate. Compression ratios of 30:1 show no apparent artifacts, and ratios of up to 100:1 show no artifacts at normal viewing distances. Transmission times are correspondingly reduced, and the algorithm may enable wide-area video transmission, interactive television, videoconferencing, and other video-on-demand applications.

5. The computational complexity of the 3-D DCT has been reduced by developing a fast 3-D DCT algorithm.

6. Hardware and software implementation issues have been explored. Substantial progress towards an inexpensive, real-time 3-D compression engine has been made.

7. Evaluation of the results has included a comparison with other compression schemes. The result of this comparison will be a useful contribution for further work in related issues.

Future Work

More work needs to be done in the development of optimal visually weighed quantizers. The identification of two classes of visual response has lead to the introduction of non-linear quantization. The initial threshold visibility of DCT coefficients was demonstrated to be related to the DCT frequency components, while subsequent relative visibility was demonstrated to be most closely related to the intensity of the DCT coefficient. Further research will likely lead to even greater compression ratios for the technique.

The algorithm was demonstrated to be implementable on only few DSPs. This would be a worthwhile effort, resulting in a fully real-time teleconferencing application of arbitrarily high resolution.

Further work on the three-dimensional DCT could result in a true 3-D DCT engine capable of faster computation of the DCT transform. The DCT is the primary bottleneck in the XYZ process.

The XYZ encoder uses eight consecutive frames for the encoding process. This may pose a very critical requirement for the XYZ encoder implementation, which must have a large memory for storing eight frames. One way to reduce large memory requirements would be to develop a recursive, real-time 3D DCT algorithm, which will update the values of DCT coefficients based on the last frame. In that case, only the last frame must be stored in the memory of the XYZ encoder. The challenge is to develop recursive equations for DCT coefficients in the form:

$$F_k(u,v,w) = F_{k-1}(u,v,w) + DCT_k(x,y,z) \tag{10.1}$$

where F_{k-1} are DCT coefficients calculated using (k-1) frames, F_k are DCT coefficients calculated using k frames, and DCT_k is the impact of the k-th frame on DCT coefficients.

Similarly, in the XYZ decoder all eight frames must be reconstructed before they are played back. This will also require large memory for storing eight consecutive frames during decoding process. Recursive, real-time inverse 3D DCT algorithms can resolve this problem by first calculating all coefficients of the first frame, playing back the frame, and then continuing with the subsequent frames.

Some promising early work was done in mating the wavelet "subsampling" idea with the DCT block-based transform idea. Early results seem to indicate that the combination of subsampling results in fewer block artifacts and possibly higher compression ratios, but at the cost of added computational complexity. This idea may also be useful in reducing the buffer requirements of the algorithm.

Some early work was also done in the development of fast DCT algorithms based on "Walsh-like" transform domains. This work held out the exciting prospect of integer and shift-based transforms approximating the DCT with few or no multiplications.

Bibliography

[A+87]
J. Ameye, J. Bursens, S. Desmet, K. Vanhoof, G. Tu, J. Rommelaere, and A. Oostevlinck, "Image Coding Using the Human Visual System", *Int'l Workshop on Image Coding*, August 1987, pp. 229-308.

[ADC95]
Analog Devices Corporation, "ADSP-2106x SHARC User's Manual", 1995.

[AG78]
W. Adams and C. Giesler, "Quantizing Characteristics for Signals Having Laplacian Amplitude Probability Density Function", *IEEE Transactions on Communication*, Vol. 26, August 1978, pp. 1295-1297.

[ANR74]
N. Ahmed, T. Natarajan, and K. Rao, "Discrete Cosine Transform", *IEEE Transactions on Computing*, Vol. 23, January 1974, pp. 90-93.

[AR75]
N. Ahmed and K. Rao, "Orthogonal Transforms for Digital Signal Processing", *Springer-Verlag*, 1975.

[BF91]
K.B. Benson and D.G. Fink, "HDTV - Advanced Television for the 1990s", *Mc-Graw Hill*, 1991.

[BK95]
V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", *Kluwer Academic Publishers,* Norwell, Mass., 1995.

[Boy92]
R. Boynton, "Human Color Vision", *Optical Society of America*, 1992.

[BSZ95]
B. Furht, S. Smoliar, and HJ. Zhang, "Video and Image Processing in Multimedia Systems", *Kluwer Academic Publishers*, Norwell, Mass., 1995.

[Bur89]
J. Burl, "Estimating the Basis Functions of the Karhunen-Loeve Transforms", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, January 1989, pp. 99-105.

[Cla81]
R. Clarke, "Relation Between the Karhunen-Loeve and Cosine Transforms", *Proceedings IEE*, Part F, Vol. 128, November 1981, pp. 359-360.

[Cla83]
R. Clarke, "Performance of KLT and DCT for Data Having Widely Varying Values of Intersample Correlation Coefficient", *Electronics Letters*, Vol. 19, March 1983, pp. 251-253.

[Cla85]
R. Clarke, "Transform Coding of Images", *Academic Press*, 1985.

[CR88]
O. Chantelou and C. Remus, "Adaptive Transform Coding of HDTV Pictures", *Proc. of the II Intl. Conf. on Signal Processing of HDTV Pictures*, February 1988, pp. 231-238.

[CR90]
B. Chitpraset and K. Rao "Human Visual Weighed Progressive Image Transmission", *IEEE Transactions on Communications*, Vol. 38, 1990.

[CSF77]
W. Chen, C. H. Smith, and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", *IEEE Transactions on Communication*, Vol. 25, September 1977, pp. 1004-1009.

[CT65]
J. Cooley and J. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", *Mathematics of Computing*, Vol. 19, 1965, pp. 297-301.

[F+92]
H. Fujiwara et al., "An All-ASIC Implementation of a Low Bit-Rate Video Codec", *IEEE Transactions of Circuits and Systems for Video Technology*, Vol. 2, No. 2, June 1992, pp. 123-134.

[Far90]
P. M. Farrelle, "Recursive Block Coding for Image Data Compression", *Springer-Verlag*, 1990.

[Fei90]
E. Feig, "A Fast Scaled DCT Algorithm", Image Processing Algorithms and Techniques, *Proceedings of the DPIE*, Vol. 1244, February 1990, pp. 2-13.

[Fis95]
Y. Fisher, "Fractal Image Compression", *Springer-Verlag*, 1995.

[Fur95a]
B. Furht, "A Survey of Multimedia Compression Techniques and Standards. Part I: JPEG Standard", *Journal of Real-Time Imaging*, Vol. 1, No. 1, April 1995, pp. 49-67.

[Fur95b]
B. Furht, "A Survey of Multimedia Compression Techniques and Standards. Part II: Video Compression", *Journal of Real-Time Imaging*, Vol. 1, No. 5, November 1995, pp. 319-338.

[GCT92]
J. Granata, M. Conner, and R. Tolimieri, "The Tensor Product: A Mathematical Programming Language for FFTs and other Fast DSP Operations", *IEEE Signal Processing*, Vol. 40, No. 8, January 1992.

[GG90]
A. Gersho and R. M. Gray, "Vector Quantization and Signal Compression", *Kluwer Academic Publishers*, 1990.

[GR82]
A. Gersho and B. Ramamurthi, "Image Coding Using Vector Quantization", *ICASSP*, April 1982, pp. 428-431.

[Gra81]
D. J. Granath, "The Role of Human Visual Models in Image Processing", *Proceedings of the IEEE*, Vol. 67, May 1981, pp. 552-561.

[Gri80]
N. Griswold, "Perceptual Coding im the Cosine Transform Domain", *Optical Engineering*, Vol. 19, May/June 1980, pp. 306-311.

[GVA92]
K. Guttag, R. J. Gove, and J. R. Van Aken, "A Single-Chip Multiprocessor for Multimedia - the MVP", *IEEE Computer Graphics and Applications*, Vol. 12, No. 6, November 1992, pp. 53-64.

[HM94]
A. C. Hung and T. H. Y. Meng, "A Comparison of Fast Inverse Discrete Transform Algorithms", *ACM Multimedia Systems*, Vol. 2, No. 5, 1994, pp. 204-217.

[HS63]
J. Huang and P. Schultheiss, "Block Quantization of Correlated Gaussian Random Variables", *IEEE Transactions on Communications Systems*, Vol. 11, 1963, pp. 289-296.

[Huf52]
D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of IRE*, Vol. 40, No. 9, September 1952, pp1098-1101.

[IEE87]
IEEE, "Special Issue on High Definition Television", *IEEE Transactions on Broadcasting*, Vol. 33, 1987.

[IS92a]
ISO, DIS 10918-1 (JPEG), "Information Technology - Digital Compression and Coding of Continuous Tone Images - Part 1: Requirements and Guidelines", 7/92.

[IS92b]
ISO, DIS 11172 (JPEG), "Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s", 10/92.

[IS93a]
ISO, CD 13818-1 (MPEG), "Information Technology - Generic Coding of Moving Pictures and Associated Audio Information - Part 1: Systems", 12/93.

[IS93b]
ISO, CD 13818-2 (MPEG), "Information Technology - Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video", 12/93.

[ITU95a]
ITU-T Recommendation H.324. "Terminal for Low Bitrate Multimedia Communication", November 1995.

[ITU95b]
ITU-T Recommendation H.263. "Video Coding for
Low Bitrate Communication", October 1995.

[ITU93]
ITU-T Recommendation H.320. "Narrow-Band Visual Telephone Systems and Terminal Equipment", March 1993.

[Jai76]
A. K. Jain, "A Fast Karhunen-Loeve Transform for a Class of Random Processes", *IEEE Transactions on Communications*, Vol. 24, 1976, pp. 1023-1029

[Jai89]
A. K. Jain, "Fundamentals of Digital Image Processing", *Prentice-Hall*, 1989.

[JJ81]
J. R. Jain and A. K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding", *IEEE Transactions on Communications*, Vol. 29, 1981, pp. 1799-1808.

[K+94]
V. Kumar, A. Grama, A. Gupta, and G. Karypis G, "Parallel Computing", *Benjamin-Cummings,* 1994.

[KSH77]
H. Kitajima, T. Saito, and T. Hurobe, "Comparison of the Discrete Cosine and Fourier Transforms as Possible Substitutes for the Karhunen-Loeve Transform", *Transactions of the IECE,* Vol. 60, June 1977, pp. 279-283.

[LF91]
E. Linzer and E. Feig, "New DCT and Scaled-DCT Algorithms for Fused Multiply-Add Architectures", *Proc. of the ICASSP*, 1991, pp 2201-2204.

[Llo82]
S. P. Lloyd, "Least Squares Quantization in PCM", *IEEE Transactions on Information Theory*, Vol. 28, No. 2, March 1982, pp. 129-137.

[Mas92a]
MasPar Computer Corporation, "MasPar MP-1 and MP-2 Architecture Specification", Document Part Number 9300-5001, 1992.

[Mas92b]
MasPar Computer Corporation, "MasPar Programming Language", Document Part Number 9302-0001, 1992.

[Mak80]
J. Makhoul, "A Fast Cosine Transform in One and Two Dimensions", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 28, February 1980, pp. 27-34.

[Max60]
J. Max, "Quantizing for Minimum Distortion", *Transactions of IRE*, Vol. 6, March 1960, pp. 7-12.

[NA77]
T. Natarajan and N. Ahmed, "On Interframe Transform Coding", *IEEE Transactions on Communications*, Vol. 25, November 1977, pp. 1323-1329.

[NH88]
A. Netravali and B. Haskell, "Digital Pictures, Representation and Compression", *Plenum Press*, 1988.

[NLS88]
K. Ngan, K. Leong and H. Singh, "A HVS-Weighed Transform Coding Scheme with Adaptive Quantization", *Proc. of the SPIE Visual Communication and Image Processing*, Vol. 1001, November 1988, pp. 702-708.

[Oku95]
S. Okubo, "Reference Model Methodology - A Tool for the Collaborative Creation of Video Coding Standards", *Proceedings of the IEEE*, Vol. 83, No. 2, February 1995, pp. 139-150.

[PDG95]
P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI Architectures for Video Compression - A Survey", *Proceedings of the IEEE*, Vol. 83, No. 2, February 1995, pp. 220-246.

[PM93]
W. B. Pennebaker and J. L. Mitchell, "JPEG Still Image Data Compression Standard", Van Nostrand Reinhold, 1993.

[Rao85]
K. Rao, "Discrete Transforms and Their Applications", *Van Nostrand Reinhold*, 1985.

[Ros88]
S. Ross, "A First Course in Probability", *MacMillan Publishing Company*, 1988.

[RPR77]
J. Roes, W. Pratt, and G. Robinson, "Interframe Cosine Transform Image Coding", *IEEE Transactions on Communication*, Vol. 25, No. 11, November 1977, pp. 1329-1338.

[RY90]
K. R. Rao and R. Yip, "Discrete Cosine Transform - Algorithms, Advantages, Applications", *Academic Press*, 1990.

[Sch88]
G. Schamel, "Coding of HDTV Signals with Bit-Rates Between 140 and 280 Mbit/s", *Proc. of the Picture Coding Symposium*, September 1988, pp13.10-1 through 13.10-2.

[Sea76]
A. Seagall, "Bit Allocation and Encoding for Vector Sources", *IEEE Transactions on Information Theory*, Vol. 22, March 1976, pp. 162-169.

[Sha49]
C. Shannon, "The Mathematical Theory of Communication", *University of Illinois Press*, 1949.

[SN95]
R. Steinmetz and K. Nahrstedt, "Multimedia: Computing, Communications & Applications", *Prentice-Hall*, 1995.

[TIC95a]
Texas Instruments Corporation, "TMS320C80 Master Processor User's Guide", Literature Number SPRU109, 1995.

[TIC95b]
Texas Instruments Corporation, "TMS320C80 Parallel Processor User's Guide", Literature Number SPRU110, 1995.

[VDG89]
M. Vitterli, P. Dumel, and Guillemot, "Trade-offs in the Computation of Mono- and Multi-Dimensional DCTs", *Proc. of the ICASSP*, May 1989, pp. 99-1002.

[VK95]
M. Vitterli and J. Kovacevic, "Wavelets and Subband Coding", *Prentice-Hall1*, 1995.

[VN84]
M. Vitterli and H. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations", *Signal Processing*, Vol. 6, August 1984, pp. 267-278.

[WF96]
R. Westwater and B. Furht, "The XYZ Algorithm fro Real-Time Compression of Full-Motion Video", *Journal of Real-Time Imaging*, Vol. 2, No. 1, February 1996, pp. 19-34.

[Wil91]
R. Williams, "Adaptive Data Compression", *Kluwer Academic Publishers*, Norwell, Mass., 1991.

[Win78]
S. Winograd, "On Computing the Discrete Fourier Transform", *Mathematics of Computing*, Vol. 32, 1978, pp. 175-199.

Index