

DIGITAL VIDEO COMPRESSION

- Compression fundamentals
- MPEG-2 and MPEG-4
- H.264 codec
- Much more!

FEATURING:

JVT / H.264 / MPEG-4 Part 10-
THE NEW COMPRESSION
STANDARD



PETER SYMES

CD
INCLUDED



Digital Video Compression

Humber College Library
205 Humber College Blvd
Toronto, ON M9W 5L7

Peter Symes

McGraw-Hill

New York Chicago San Francisco Lisbon London Madrid
Mexico City Milan New Delhi San Juan Seoul
Singapore Sydney Toronto

Cataloging-in-Publication Data is on file with the Library of Congress

Copyright © 2004, 2001 by The McGraw-Hill Companies, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 9 8 7 6 5 4 3

P/N 142494-6
PART OF
ISBN 0-07-142487-3

The sponsoring editor for this book was Stephen S. Chapman and the production supervisor was Sherri Souffrance. It was set in Vendome by Patricia Wallenburg. The art director for the cover was Anthony Landi.

Printed and bound by RR Donnelley.

Portions of this book were previously published in *Video Compression Demystified*, © 2001.

McGraw-Hill books are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. For more information, please write to the Director of Special Sales, McGraw-Hill Professional, Two Penn Plaza, New York, NY 10121-2298. Or contact your local bookstore.



This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

Information contained in this work has been obtained by The McGraw-Hill Companies, Inc. ("McGraw-Hill") from sources believed to be reliable. However, neither McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein and neither McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

CONTENTS

Preface	xv
Acknowledgments	xvii
1 What Is Compression?	1
Introduction	2
Information and Data	2
Information Reduction	4
Lossless Compression	5
Run-Length Encoding	5
Entropy Encoding	6
Lossy Compression	8
Image Compression Standards	10
Symmetric and Asymmetric Systems	11
Why Do I Care?	12
Upcoming Topics	12
2 An Introduction to Images	15
Introduction	16
Video or Computer Graphics?	16
Sampling and Quantization	17
Sampling Rate and Aliasing	18
One-Dimensional Sampling	18
Two-Dimensional Sampling	26
Temporal Sampling	29
3 Entropy Coding	35
Introduction	36
Entropy	36
Huffman Codes	41
Dangers of Variable-length Coding	43
Modified Huffman Codes	44
Arithmetic Encoding	47

4	Predictive Coding	49
	Markov Sources	50
	Differential Pulse Code Modulation	54
	Predicting Image Values	56
	Adaptive Prediction	59
5	Transforms	67
	Introduction	68
	Time, Space, and Frequency Domains	70
	Frequency and Spatial Frequency	72
	The Discrete Cosine Transform	74
	The Fourier Approach to DCT	76
	DCT in Terms of Basis Functions	81
	DCT as Axis Rotation	85
	DCT Examples	87
	DCT Failure	88
6	Quantization	91
	Introduction	92
	Mean Square Error	92
	Types of Quantizer	93
	Uniform Scalar Quantizer	94
	Nonuniform Scalar Quantizers	95
	The Lloyd-Max Quantizer	97
	Entropy-constrained Quantizers	97
	Vector Quantization	98
	Applications of Quantization	100
	Direct Image Quantization	100
	Quantization with Predictive Coding	103
7	JPEG	107
	Introduction	108
	Baseline JPEG	109
	Image Data Encoding	109
	JPEG Baseline Encoding and Decoding	110
	DCT Transform	113
	Quantization	116
	Scanning, Descriptors, and Entropy Coding	119

Assembling the Bit Stream	125
Parsing the Received Bit Stream	125
Recovering the Quantized Coefficients	126
Dequantization	127
Inverse DCT	127
Comparison	129
Examples of Baseline JPEG	129
JPEG Extensions	130
Motion JPEG	131
8 Motion Compensation	137
Introduction	138
Temporal Redundancy	138
Motion Aliasing	138
The Motion Compensation Approach	139
Static Backgrounds	139
Motion Vectors	141
Block Matching	143
Matching Criteria	143
Optimum Block Size	143
Full-Search Block Matching	145
Hierarchical Block Matching	146
Residuals	147
Other Motion Estimators	148
Restricted Search Systems	148
Phase Correlation Motion Estimation	149
9 MPEG-1	151
Introduction	152
What MPEG Defines	153
Hierarchy and Terminology	154
Frame Types in MPEG	156
Intraframes (I-frames)	156
Non-intra Frames (P-frames and B-frames)	156
MPEG Group of Pictures	158
Coding of Macroblocks	160
Intracoding of Macroblocks	161
Non-intracoding of Macroblocks	162

Rate Control	165
The MPEG Encoder and Decoder	169
10 MPEG-2	171
Introduction	172
MPEG-2 Enhancements	173
Color Space	173
Slice Structure	173
Quantization	173
Concealment Motion Vectors	174
3:2 Pulldown	175
Pan and Scan	175
MPEG-2 Profiles and Levels	175
Interlace Tools	178
Frame and Field Pictures	179
Frame and Field DCT	180
Frame and Field Prediction	182
Scalable Coding Profiles	183
MPEG-2 System Layer	184
Packetized Elementary Stream	185
Program Stream	185
Transport Stream	186
Practicing the Art of MPEG	189
Contributors to Poor Performance	189
MPEG Artifacts	190
Tips for Higher Quality	191
Update 2003	192
11 MPEG-4	195
Introduction	196
Video in MPEG-4	200
MPEG-4 Video Hierarchy	200
Shape Coding	201
Texture Coding	203
Boundary Coding	205
Coding of Arbitrary-Shaped Video Objects	205
Sprites	206
Static Texture Coding	208

Animations	208
Scalability	209
Advanced Coding Extensions (ACE)	211
Visual Profiles	212
Scene Compositing and Interaction	216
Scene Modeling	216
Interaction	217
Work in Progress	217
Studio Profiles	217
Fine Grain Scalability	218
Conclusion	220
Update 2003	221
12 Joint Video Team (JVT)	223
Introduction	224
JVT Requirements	226
A Summary of the JVT Codec	229
Similarities of JVT to MPEG-2	230
New Concepts and Technologies	230
JVT Tools	232
Transform	232
Logarithmic Quantization	235
Spatial Prediction	236
Deblocking Filter	236
Motion Estimation	238
Temporal Prediction	240
Macroblock-adaptive Frame/Field Coding	242
Lossless Coding	242
Profiles and Levels in JVT	243
Baseline Profile	243
Main Profile	244
Extended Profile	244
Levels	244
Future Work of JVT	245
13 Windows Media	247
Introduction	248
Windows Media History	248

Windows Media 9 Codecs	249
Windows Media Audio 9	250
Windows Media Audio 9 Professional	250
Windows Media Audio 9 Voice	250
Windows Media Audio 9 Professional Lossless	251
Windows Media Video 9 (WMV-9)	251
Windows Media Video 9 Screen	251
Windows Media Video 9 Image	251
Windows Media Video 9	252
Frame Interpolation in WMV	254
Windows Media 9 Licensing	256
14 Intellectual Property and Licensing	257
Introduction	258
Standards Bodies and Patents	258
Licensing Bodies	259
Windows Media 9 Licensing	261
JVT Licensing	262
15 MPEG-7 and MPEG-21	267
Introduction	268
MPEG-7	268
Concepts of MPEG-7	268
MPEG-7 Terminology	271
MPEG-7 Structure	272
MPEG-7 Visual	273
Summary	274
MPEG-21	276
16 Pro-MPEG and MPEG Operating Ranges	279
Introduction	280
Goals of Pro-MPEG	281
Update 2003	285
17 DV Compression	287
Introduction	288
Basic Concepts of DV Compression	289
Detailed Description	290

25 Mbits/s Compression	290
50 Mbits/s Compression	298
Update 2003	299
18 Wavelets	301
Introduction	302
More about Fourier Transforms	302
Wavelets Concept	306
Wavelets as Filters	307
Wavelet Compression	312
19 JPEG2000	317
Introduction	318
Limitations of the Original JPEG System	318
Goals of JPEG2000	319
20 Audio Compression	321
Introduction	322
Masking in Human Hearing	323
Simple Audio Compression Schemes	324
Quadrature Mirror Filters	326
Subband Coding	330
Bit Allocation	332
Transform Coding	332
Example Compression Systems	333
Audio Compression in MPEG	333
Audio Compression for ATSC	334
21 Streaming Media	339
Introduction	340
Applications for Streaming Media	341
Standards for Streaming Media	343
Update 2003	345
22 Closing Thoughts	347
Introduction	348
Fractal Compression	348
Statistical Multiplexing	349

Concatenated Compression Systems	351
Switching MPEG	354
MPEG Applications	358
Some Solutions	359
Mezzanine Compression Systems	361
A Glimpse into the Future	364
A Glossary	365
B Bibliography	377
C Internet Resources	381
D About the CD-ROM	385
Index	387

CHAPTER **1**

What Is Compression?

Introduction

Compression is the science of reducing the amount of data used to convey information. In this book we will examine a wide range of techniques—some simple, some very complex—and ways in which these techniques may be applied to video.

Compression relies on the fact that information, by its very nature, is not random but exhibits order and patterning. If this order and patterning can be extracted, the essence of the information can often be represented and transmitted using less data than would be needed for the original. We can then reconstruct the original, or a close approximation of it, at the receiving point.

There are several families of compression techniques, fundamentally different in their approach to the problem. These techniques are in fact so different that they can often be used sequentially to good advantage. Sophisticated compression systems use one or more techniques from each family to achieve the greatest possible reduction in data.

There are arguments about the semantics of compression. Some authorities use the term *compression* only when there is some resultant impairment, and they prefer terms like *bit rate reduction* for techniques that involve no loss. In this book *compression* is used to cover all techniques that reduce data, whatever the nature of the process and whether or not errors are introduced.

Before examining compression techniques, let us look at the general issue of information and the data used to represent it.

Information and Data

A major concern in today's world is the handling of information. The information may be of many types—written text, the spoken word, music, still pictures, and moving pictures are just a few examples. Whatever the type of information, we can represent it by electrical signals or data, and transmit it or store it.

The more complex the information, the more data is needed to represent it. Plain text can be represented by 8 bits/character, or about 20 kbits for a page. CD-quality music requires nearly 1,500 kbits for each second, and full-motion 525-line video (as used in North American television systems) needs about 200 Mbits (200,000 kbits) for each second.

When we need to transmit a given amount of data, we use a certain bandwidth (measured in bits per second) for the time necessary to transmit all the bits. In the real world, there is always some restriction on the available bandwidth. For example, if the transmission must be over regular telephone lines using modems, it is difficult to achieve more than 50 kbits/s (kilobits per second). This is fast compared to the 300-bits/s modems of a few years ago, but even at 50 kbits/s it would take some two hours to transmit a single second of high quality video.

Sometimes it is a practical restriction, like the use of a modem, that sets the requirements for compression. We could send high-quality uncompressed audio data over a modem, but each second's worth of audio would take about 30 seconds to transmit. In other words, we would have to receive the data gradually, store it away, then play the resulting file at the correct rate to hear the sound. However, we may want to transmit sound so that we can listen to it as it arrives in "real time." We could choose to reduce the quality we are trying to send, but even telephone-quality speech needs a data rate substantially higher than our high-speed modem can provide. If we want to send real-time audio over a modem link, we have to compress the data.

Another example of this type of requirement is the digital television system developed for North America. When the study was started in 1977, few thought it would be possible to transmit high-definition television (characterized by the 1,125-line system) over a single 6-MHz channel as used for today's 525-line television. When digital techniques were considered, the problem looked even worse, because the high-definition signal represented over 1 Gbits/s of data. In the event, two teams of engineers combined to produce the answer: The transmission engineers designed a system to deliver nearly 20 Mbits/s reliably over a 6-MHz channel, and the compression engineers achieved data reduction of about 60 to 1, while maintaining excellent picture quality.

At other times the motive for compression is financial. Bandwidth costs money, and so compression reduces the cost of transmission. A satellite link with a data bandwidth of 40 Mbits/s will earn more money if it can carry ten television channels instead of just one. I may want to download a 6-MB file from a computer online service; if a compressed version is available that is only 1.2 MB, I will save 80 percent of the online charges.

Transmission and storage of information might seem to be very different problems but, in terms of quantity of data, the economics are the same. Just as there is a cost for each bit transmitted, there is a

cost for each bit stored in memory, on disk, or on tape. If we can use less data, both transmission and storage will be cheaper. There are some differences in how compressed data is best structured, and these will be mentioned in Chapter 10. Otherwise, everything we say about compression applies equally to transmission or storage.

The motives for wanting to reduce the amount of data should be clear. Now we will examine some of the techniques.

Information Reduction

One important step in compression technology is easy to ignore because it lacks the glamor of some of the sophisticated techniques. It is extremely important to ensure that the signal or data stream we want to compress represents the information we want to transmit, and *only* the information we want to transmit. If the data stream contains surplus information of any nature, this information will take bits to transmit and result in fewer bits being available for the information we do need. Surplus information is *irrelevant* because the intended recipient can make no use of it.

Surplus information can take many forms. It can be information in the original signal or data stream that exceeds the capabilities of the receiving device. For example, there is no point in transmitting more resolution than the receiving device can use, and no point in transmitting color information if the receiver is known to be monochrome.

Another form of surplus information is *artifacts*—features or elements of the input that are not truly part of the information. *Noise* is the most obvious example; almost anything derived from a physical transducer has some component of noise. Noise is by nature random or nearly so and, as we shall see, this makes it essentially incompressible. In video, noise may come from many sources including the transducer, analog processing, and film grain. When the signal has been converted to digital form, it may have additional *quantization noise*, and digital processing may add noise through truncation.

Many other types of artifacts exist, ranging from filter ringing to film scratches. Some may seem trivial, but in the field of compression they can be very important. Compression relies on order and self-consistency in a signal. An artifact—anything that does not “belong”—tends to require a disproportionately high number of bits to transmit, just because it destroys this order and self-consistency.

This book concentrates on the techniques of compression. Preprocessing of video to remove irrelevant information and artifacts is a very complex subject in itself. A preprocessing unit developed by Snell & Wilcox includes no less than seven different filters!

Lossless Compression

Some compression techniques are truly lossless. In other words, when we have compressed some data, we can reverse the process (*decompress*) and get the same data we started with, exactly and precisely. Lossless compression works by removing *redundant* information—information that, if removed, can be recreated from the remaining data.

In one sense, this is the ideal of compression; there is no cost to using it other than the cost of the compression and decompression processes. Unfortunately, lossless compression suffers from two significant disadvantages. Typically, lossless compression offers only relatively small compression ratios, so used alone it often does not meet economic needs. Also, the compression ratio is very dependent on the input data. Used alone, lossless compression cannot guarantee a constant output data rate such as might be required for a transmission channel.

For some applications, lossless is the only type of compression that can be used. If we wish to transmit a binary file such as a computer program, receiving an approximation to the original is of no use whatsoever. The program will run only if replicated exactly. Fortunately, many computer files have a high degree of order and patterning, so lossless compression techniques do yield useful results. Compression programs like the well-known PKZIP use a combination of lossless techniques and are particularly effective on graphics image files.

One advantage of lossless compression is that it can be applied to any data stream. Most video compression schemes use lossy techniques to achieve large degrees of compression. Lossless techniques are then applied to the resulting data stream to reduce the data rate even further. Two lossless techniques are in general use.

Run-Length Encoding

Many data streams contain long runs of a value. For example, a graphics file will typically contain values for each pixel, ordered line by

line. A red object will likely be represented by several long runs of the value “red.” With appropriate coding it can use much less data to say “red 23 times” instead of “R R R R R R R R R R R R R R R R R R R R R R R.” This is run-length encoding—we code the length of the “run” of a certain value. In video compression, various tricks can be played to make the data suitable for run-length encoding.

Entropy Encoding

Entropy encoding is usually the last step in compression. Whereas run-length encoding relies on adjacent values being the same, entropy encoding looks at the overall frequency of specific values, wherever they are located. The implementation of entropy encoding is quite complex, but the principle is easy to understand if we define our terms carefully.

If the data is represented by bytes (groups of 8 bits), each byte can have any *value* between 0 and 255_{10} . When we come to transmit this data we will choose a *symbol* to represent each possible value, and we transmit a sequence of *symbols* until all the *values* have been sent.

The simplest way to code the data is to send the bytes themselves; in other words, each symbol is the same as the value it represents. This is so obvious that we tend not to think of anything else, and make no distinction between the value and the symbol. But it does not have to be like this. So long as both the transmitting and receiving points know and apply the same set of rules, we can use any set of 256 unique symbols to represent the 256 possible values. This is not very helpful until we add another factor: Provided we have a set of rules that lets us know when one symbol ends and another begins, *not all the symbols need to be the same length*.

This is the essence of entropy encoding. It compresses the data by using short symbols to represent values that occur frequently, and longer symbols to represent the values that occur less frequently. Unless the input data is very close to random in its distribution of values, this technique means that total data used for the symbols will be less than the total of the original values.

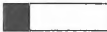
Optimization of entropy encoding is complex, but the basic idea is neither difficult nor new. Let us compare two ways of coding the alphabet. Anyone who has spent much time with computers has come across the ASCII encoding system. In ASCII we represent any letter by a particular set of 8 bits, or 1 byte. It does not matter what *value* (letter)

we choose, the *symbol* is 8 bits long.¹ A much earlier method of encoding the alphabet was invented by Samuel Morse in 1838. The Morse code is a simple but useful example of an entropy coding system.

The Morse code uses combinations of dots and dashes as symbols to represent letters. A dash is 3 times as long as a dot; each dot or dash within a symbol is separated by one dot length; each symbol is separated by one dash length. Morse designed his code to transmit English-language text, and to maximize the speed of transmission he looked at the frequency with which the various letters of the alphabet appear in typical text. The letter *e* is the most common letter in English, so Morse assigned the shortest possible symbol—one dot (actually four dots in length because of the intersymbol space). The next most common letter *t* received the next shortest symbol, one dash. At the other extreme, an infrequent letter like *y* receives three dashes and a dot, a symbol four times as long as that used for *e*, as shown in Figure 1-1.

Figure 1-1

Morse Code provides a simple example of variable length encoding.

e		4
t		6
c		14
y		16
z		14

The Morse code also illustrates one of the problems of entropy encoding. For transmission to be successful, obviously both the transmitter and the receiver must use the same mapping of symbols to values. The relationship between values and symbols is known as the *code*

¹ Of course, ASCII encodes numbers as well as letters; it encodes uppercase and lowercase letters, and it encodes punctuation and other symbols. If we need to encode the letters only, without regard to case, we do not need 8 bits because there are not 256 letters. The old telex coding using 5-bit symbols would be sufficient. However, the comparison of fixed and variable symbol lengths is still valid.

table. But the most efficient mapping can be determined only when the frequency of values is known. The code table for Morse is derived from the frequency of letters in English text. If we wanted to send Polish-language text, where the letters *c*, *y*, and *z* are used more frequently than in English, this code table would not be very efficient because those letters have long symbols. If we wanted to transmit a few lines of Polish in the middle of many pages of English, this is probably not a major concern. However, if we had to transmit a very large volume of Polish, we might want to change the code table. We can do this if we have established rules at the transmitter and receiver to allow for different code tables, but we would need to evaluate this decision carefully. If we decide to change code tables, we have to transmit the new table, and that means transmitting a considerable amount of additional data!

There is no perfect solution to this, and the best compromise depends on the application. Some coding schemes use only one code table; this may not always be the most efficient table, but the coding system does not carry any of the overhead necessary to change tables. Other systems have a small number of fixed code tables, known to both transmitter and receiver. The encoder can examine the data to be transmitted, and select the most efficient code table. The only overhead is the provision of a message that tells the receiver "Use table number 3." Still other schemes allow for the transmission of custom code tables whenever the characteristics of the data change substantially.

Lossy Compression

Lossless compression would be an ideal answer, except that it rarely provides large degrees of compression, and it cannot be used alone to guarantee a fixed bit rate. Lossless compression is an important part of our tool kit, but on its own it does not provide a solution to many practical problems. If we want to put digital audio over a modem link, or high-definition television through a 6-MHz channel, we have to accept that the compression process will result in some loss; what we get out will not be exactly the same as what we put in. This is the field of *lossy compression*. Ideally, lossy compression, like information reduction, removes *irrelevant* information. Some information is truly irrelevant in that the intended recipient cannot perceive that it is missing. In most cases we also look for information that is close to irrelevant, where the quality loss is small compared to the data savings.

The objective of lossy compression is simple. We want to get maximum benefit (compression ratio, or bit rate reduction) at minimum cost (the loss in quality). However, the realization of this objective is not simple. As we will see when we look in detail at lossy compression techniques, a very large number of parameters have to be chosen for any given implementation, and many of these parameters must be varied according to the dynamic characteristics of the data. Even with a simple measure of the resulting quality (or lack of it), optimizing such a large number of variables is a complex task. Because these schemes rely on a knowledge of how the information will be perceived by the recipient, they are known as *perceptual coding systems*.

Unfortunately, there is as yet no simple measure that can be applied to the quality of a compression system, particularly for video. The only measure that really matters is the subjective effect as perceived by a viewer, and this is an extremely complex function. If compression loss were confined to one characteristic of a picture, it would be relatively simple to derive a way of measuring this characteristic objectively. We could then perform careful subjective tests on a representative sample of viewers and arrive at a calibrated relationship between this characteristic and subjective quality. Unfortunately, compression loss results in not one but many changes to the characteristics of the picture; each of these has a complex, nonlinear relationship to subjective quality, and they interact with each other! (As an indication of the difficulty of quantifying compression loss in images: Workers in the field consistently observe that the perceived quality of a picture improves as the quality of the accompanying sound improves.)

Compression loss in images involves two main components: things that should be in the picture, but are lost, and things that are added to the picture (artifacts) that should not be there. Areas where we would expect loss include spatial resolution (luminance and color, probably to different degrees) and shadow and highlight detail. Where extreme compression is needed for applications like video conferencing, or the Internet, temporal resolution would also be sacrificed. Artifacts that might be added include blocking, "mosquito noise" in edges, quantization noise, stepping of gray scales, patterning, ringing, and the like. As a further complication, in the presence of loss—say of resolution—the addition of small amounts of artifact such as noise or ringing may actually improve the subjective quality of the picture.

Even within "standardized" compression schemes such as MPEG, the tuning of a system to yield the maximum quality per bit is a black art involving models of the human psychovisual system that are closely guarded commercial secrets. Evaluation of the effect of parameter changes is very complex for the reasons discussed above.

There may be light at the end of the tunnel. In the fall of 1997, Tektronix released a product called the *signal quality analyzer*. This device uses a technology known as *just noticeable difference* (JND) developed jointly by Tektronix and Sarnoff Corporation. It compares an output sequence from a compression system with the known input sequence, and weights each error according to its type, severity, location, and the masking effect of other nearby picture elements. The intent is to model the behavior of the human eye-brain combination, and initial tests show an encouraging degree of correlation between the measurements made by the system and subjective testing with observers.

Image Compression Standards

Any practical compression system requires the use of a number of compression tools. These tools are combined to provide a cost-effective and optimized solution for the particular problem at hand. In proprietary systems where compression is used only internally, the designer has a free hand in selecting what tools will be used, and how. In a professional videotape machine such as the Sony Digital BetaCam®, the compressed data is used only within the machine. All connections to the outside world are made as standard uncompressed video, so that there is no need for anyone outside the manufacturing organization to know the details of the compression scheme.

This approach works as long as the compressed data stays within one manufacturer's system. We may wish to send compressed data over a network or a broadcast channel, or place it on a CD, and have it usable by a different manufacturer's equipment at the other end. In this case, everyone needs to know how to decompress the data, even if they do not know exactly how the compression was achieved. In other words, we need one or more standardized compression systems.

In the field of electronic imagery, standardization work has been undertaken by the International Organization for Standardization (ISO) in cooperation with the International Telecommunications Union and the International Electrotechnical Commission (IEC). The

Joint Photographic Experts' Group and the Motion Picture Experts' Group produced the well-known JPEG, MPEG-1, and MPEG-2 standards that form the basis of most image compression today. In 1996, the National Academy of Television Arts and Sciences recognized the importance of this work by the award of an Emmy statue to ISO and IEC.

The JPEG and MPEG standards, and the continuing work of the MPEG committee, are discussed in later chapters.

Symmetric and Asymmetric Systems

Compression systems are, in general, complex and quite expensive. One characteristic important in the choice of a system is the degree of *symmetry* or *asymmetry*. This is best illustrated by examining two different scenarios.

Consider the application of videoconferencing, a subject of great interest to compression specialists. To gain popularity for other than special events, videoconferencing must be inexpensive and convenient. To be inexpensive, it must use a minimum of bandwidth, and the transmitting and receiving equipment must be affordable. To achieve low bandwidth, we need compression, and every conference station needs the ability to compress and decompress. This is a symmetrical situation—there is no benefit in designing a system where compression is more or less complex and expensive than decompression.

In contrast, some applications, such as television broadcasting, are characterized by a single compressor feeding a very large number of decompressors. CD-ROMs and digital video disks (DVDs) have similar economics; one compression system is used to produce the master, but millions of players decompress the data. These are asymmetric models. If we can reduce the data rate, or simplify the decompressor, by additional complexity in the compressor, the system as a whole benefits economically.

Baseline JPEG is very close to a symmetric system. Encoder and decoder complexities are very similar. In contrast, MPEG, with its use of motion compensation, is an asymmetric system. The bit rate is reduced with relatively little increase in decoder complexity, but at the expense of much greater encoder complexity and cost.

Why Do I Care?

It is tempting for the television engineer to conclude that compression is a subject for experts, and that it takes place within a black box. In many ways this is valid; an engineer designing or operating a television facility does not need to be able to design a television camera, or a routing switcher, and the same is true for compression systems. However, good engineers understand in principle what is happening in each of the major components of a system, even if they could not design that component.

Compression will exist at some point in virtually all television systems. The performance of the compression is affected, sometimes dramatically, by characteristics of the video signal. A signal that has been compressed carries a signature, often invisible, that can radically alter the behavior of downstream operations.

I believe that video engineers must understand the processes that are being applied to their images. Only in this way can they protect quality and diagnose problems. Anyway, compression is a fun subject!

Upcoming Topics

In Chapter 2 we look at many of the fundamentals of the sampled images that form the actual input to any compression system. Many of the topics covered are not specific to compression, but all are fundamental to its operation. Most engineers will be familiar with this material, but I encourage you to read the chapter, anyway. These ideas are absolutely essential to the operation of compression systems, and many so-called experts go wrong because they fail to remember or apply the fundamentals discussed here.

Chapters 3 to 6 cover the basic tools of compression systems for static images, and Chapter 7 brings these tools together in a description of the JPEG system. Chapter 8 introduces the special characteristics of moving images and the technology of motion compensation. Having dealt with that, we can then move on, in Chapters 9 and 10, to MPEG-1 and MPEG-2.

Digital Video Compression concentrates largely on JPEG and MPEG because these are still the most important systems for the video engineer today. Although there are more modern techniques, the large number of MPEG-2 decoders deployed in digital TV receivers (includ-

ing satellite receivers) and in DVD players is sufficient to ensure that this standard will be important for many years. MPEG-4, in its early form, is described in Chapter 11. The latest work of the standards bodies, and the latest part of MPEG-4, comes from the Joint Video Team (JVT) of MPEG and ITU-T, and is discussed in Chapter 12. Another new contender for attention is Windows Media 9, covered briefly in Chapter 13. Chapter 17 describes the DV compression system, which uses similar technology to MPEG, but in a quite different way. Later chapters explore the principles and application of other technologies, including wavelets and fractals, and the fundamentals of audio compression.

No book on compression would be complete without a look at streaming video and audio on the Internet, and this leads to a description of the contents of the CD-ROM included with this book. Here you will have the opportunity to explore some of the concepts directly through demonstration programs and sample images and sequences.

The book is organized in a progressive manner—each chapter assumes knowledge of at least the basic elements of previous chapters—so it is a good idea to work through the book in the correct sequence. However, each chapter covers the basics first, and then goes into greater detail. The fundamentals are important, and much of the detail is provided for example, and because I find it fascinating. If the detail gets too much, move on to the next chapter—I had fun writing this book, and I want you to enjoy reading it!

CHAPTER **11**

MPEG-4

Introduction

The wheels of international standardization grind slowly, and to ensure that a standard is eventually achieved there are strict rules that prohibit substantive change after a certain point in the process. By the time a standard is officially adopted there is often a backlog of desired enhancements and extensions. So it was with MPEG-2. As discussed in Chapter 10, MPEG-3 had been started and abandoned, so the next project became MPEG-4.

MPEG-4 exists in two parts, Version 1 and Version 2, which is a superset of Version 1. In other words, all Version-1 constructs are valid within Version 2. Work began in 1993, and was formalized in July 1995 with a call for proposals. The various proposals for audio and video were evaluated by subjective testing and by experts. Drafting of the Version 1¹ followed, leading to a Committee Draft in November 1997, and became an ISO/IEC International Standard in 1999. Meanwhile, work continued on the Version-2 extensions and the Draft Amendment was frozen in December 1999, with various phases of formal adoption during 2000. As of the summer of 2000, work on extensions continues forming the basis for Amendments 2 and 3, representing Versions 3 and 4 respectively. Even more work is planned, to determine and meet the needs of digital cinema.

At first, the main focus of MPEG-4 was the encoding of video and audio at very low rates. In fact, the standard was explicitly optimized for three bit rate ranges:

- Below 64 kbits/s
- 64 to 384 kbits/s
- 384 kbits/s to 4 Mbits/s

Performance at low bit rates remained a major objective and, as we shall see, some very creative ideas contributed to this end. Great attention was also paid to error resilience, making MPEG-4 particularly suitable for use in error-prone environments, such as transmission to personal handheld devices. Other profiles and levels use bit rates up to 38.4 Mbits/s, and work proceeds on studio-quality profiles and levels using data rates up to 1.2 Gbits/s.

More important, MPEG-4 became vastly more than just another compression system—it evolved into a totally new concept of multi-

¹ Except for Conformance Testing, which followed about a year later.

media encoding with powerful tools for interactivity and a vast range of applications. Even the official “overview” of this Standard spans 67 pages, so in this chapter I can only attempt a brief introduction to the system. I hope I will manage to convey some of the new concepts and demonstrate the enormous richness of MPEG-4; it really is a Standard with something for everyone, and each facet of the Standard offers a wealth of tools and options. Despite the focus on the low bit rate applications, MPEG-4 includes important extensions for high-end studio operations and high-definition television.

The most significant departure from conventional transmission systems is the concept of *objects*. Different parts of the final scene can be coded and transmitted separately as *video objects* and *audio objects* to be brought together, or *composited*, by the decoder. This raises a number of interesting possibilities. Most obviously, different types of objects may be coded using different techniques, and each may be coded with the tools most appropriate to the job. The different objects may be generated independently, or in some cases a scene may be analyzed to separate, for example, foreground and background objects. An interesting example was demonstrated recently. Video coverage of a soccer game was processed to separate the ball from the rest of the scene. The background (the scene without the ball) was transmitted as a “teaser” to attract a pay-per-view audience. Anyone could see the game, but only those who paid could see the ball! Figure 11-1 shows the general concept.

As we have seen throughout this book, no coding system is universally efficient. DCT and quantization are effective for continuous-tone, bandwidth-limited images, but certainly not optimal for everything that we may wish to display. As a simple example, if a scene is to be overlaid with text, conventional systems key the text into the video, often filling the body and edges of the letters with contrasting colors. The resulting combination is then coded by normal means. Superimposing text in this manner adds a good deal of high-frequency energy to the signal, and these edges are not well coded by DCT. They consume a disproportionate number of bits, and the result is often less than pleasing. One of the very noticeable artifacts of DCT-based encoding is the effect known as *mosquito noise* on sharp edges such as overlaid text, caused by excessive quantization of the high-frequency coefficients.

Thus, the addition of text has a significant impact on the efficiency of the video coding. Yet we know that we could code that text in a very simple manner—the letters can be specified in ASCII (or some more efficient code), and position, font, size, color, and spacing information

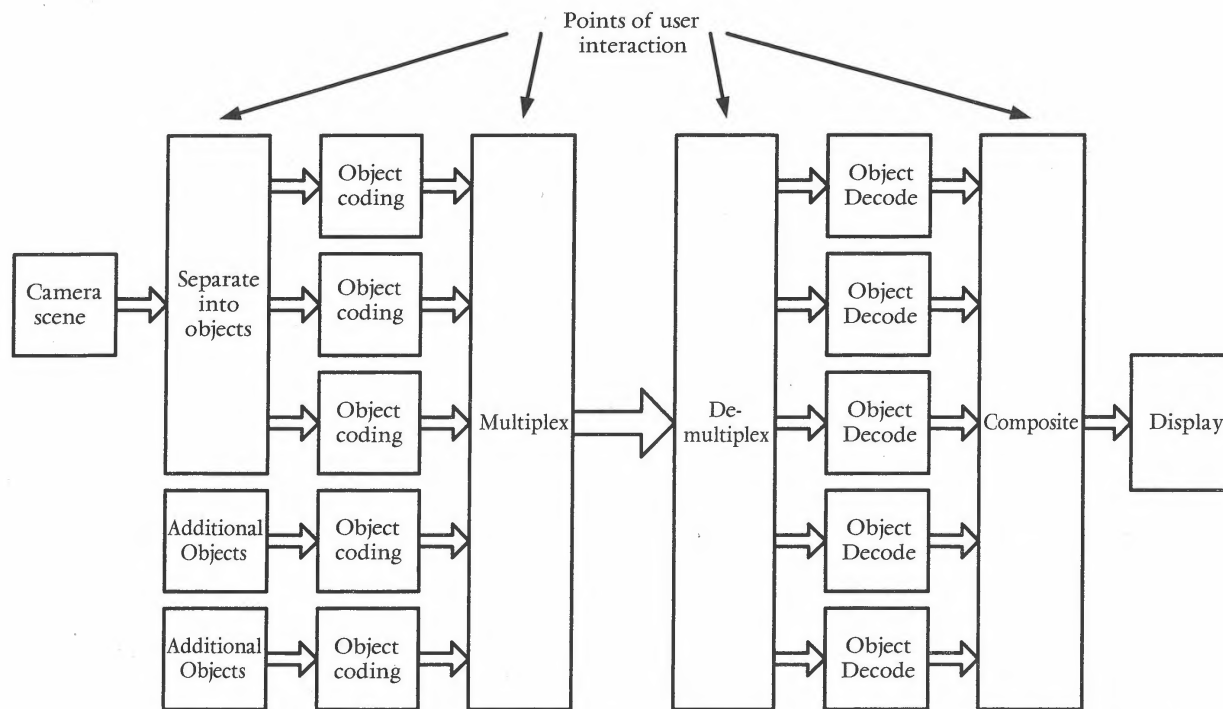


Figure 11.1 Objects and object coding in the MPEG-4 video system.

can be added with a fairly small number of bits. But for this to work, the decoder needs the ability to generate the titles from the information supplied, and to key the titles over the decoded video prior to display. Decoding of MPEG-4 video bitstreams may require a decoder with multiple decode engines and the ability to perform compositing operations.

MPEG-4 is not confined to video and text. We discuss some of the other options later, but for the moment let's continue with the text example. The ability to composite elements at the decoder provides far more than coding efficiency. In conventional systems, the image to be displayed is determined and constructed prior to transmission; overlaid titles become a part of the video prior to encoding and transmission. In MPEG-4, it is possible to transmit several text streams and to choose at the decoder which stream or streams (if any) to combine with the video. The choice may be entirely that of the viewer, or may be constrained by other information in the transmitted bit stream.

A simple and crude example of this capability exists in today's analog television system. "Closed captions" for the hearing impaired are coded into one line at the top of the picture, and the viewer may

choose whether or not to overlay them onto the video. MPEG-4 offers vastly more flexibility and efficiency even with simple text; color, position, and font can all be specified by the author, and the viewer may have the choice, for example, of several languages.

We will look in more depth at the capabilities of the system, but we have already identified three key characteristics of MPEG-4 streams:

- Multiple objects may be encoded using different techniques, and composited at the decoder.
- Objects may be of natural origin, such as scenes from a camera, or synthetic, such as text.
- Instructions in the bitstream, and/or user choice, may enable several different presentations from the same bitstream.

These capabilities do not have to be used—MPEG-4 provides traditional coding of video and audio and improves on MPEG-2 by offering increased efficiency and excellent resilience to errors. However, the true power of MPEG-4 comes from the new applications made possible by the architecture described above. The independent coding of objects offers a number of advantages. Each object may be coded in the most efficient manner, which more closely emulates the processes of the human psychovisual system; the separation of objects allows interaction with these objects—particularly powerful in games and educational programs. Also, different spatial or temporal scaling may be used as appropriate. As an example, if bandwidth and/or computational resources are limited, it may be appropriate to retain full temporal resolution for an important foreground object, but to update to the background at a lower rate.

There are also a number of disadvantages to the object approach. The decoder must be capable of decoding all the possible bit streams it supports, and it must have a compositing capability. Thus a hardware MPEG-4 decoder is necessarily more complex than a comparable MPEG-2 decoder. Similarly, more code is necessary to implement a software decoder, but this decoder can have considerable flexibility in using the limited hardware resources to the best advantage.

Up to now, we have discussed only conventional video images and text. MPEG-4 provides other video objects, some of which are far more innovative; these are discussed in the next section. Clearly a proliferation of object types complicates decoder design, and it would be uneconomical to require all decoders to support all possible constructs and objects. Not surprisingly, MPEG-4 defines a number of profiles and levels, in a similar manner to MPEG-2, permitting implementation with different tool sets and various parameter limitations.

Video in MPEG-4

Before looking at the techniques used for video compression in MPEG-4 we must examine the structure of a video scene, as defined for MPEG-4. A typical scene might include a background, one or more foreground objects such as furniture, one or more people, and some graphic elements. In conventional video thinking, and in MPEG-1 and MPEG-2, the complete scene is sampled once per frame, producing a bitmap that is coded by the techniques discussed in earlier chapters. MPEG-4 can also work this way, but it can also handle each of the objects separately, if the information is presented in the correct way.

To make life simple, let's forget about the furniture for now; we will just assume it is a part of the background. So, apart from any graphic elements, the scene consists of a background and (say) one person as a foreground object. It is well known that the scene may not be that way in the studio—the person, such as a weather forecaster, may be standing in front of a plain blue or green background and *chroma keyed* over a different background (such as a weather map). In the studio, the image of the person in front of the plain-color background is processed to eliminate the background color and to generate a *key signal* or *alpha channel* representing the shape (and, perhaps, the transparency) of the foreground person. This shape information is used to composite the two scene elements together. Where the foreground person stands, the background scene is replaced by the image of the person. Elsewhere (outside the shape) the background is unchanged. If the person is holding a semi-transparent object like a glass of water, the final image must be a mix of foreground and background scenes, determined by transparency information contained in the key signal. For realism, a mix is also needed along the edges of the foreground shapes.

In MPEG-4 terminology, the person in the foreground is known as a *video object*, represented by two elements, the video image of the person, known as the *texture*, and the key signal or alpha channel, known as the *shape*.

The background image does not really need a shape associated with it, but MPEG-4 treats this as having a rectangular shape, a degenerate case.

MPEG-4 Video Hierarchy

The starting point for video in MPEG-4 is a visual concept rather than a signal, such as a foreground person in a scene. Before we can

do anything with the object it must be sampled. Most objects are sampled at regular time intervals (frames), and each time sample (a complete spatial representation at one instant in time) is known as a *video object plane* (VOP). So, each object in the scene is represented by a series of video object planes—except a sprite or other static object, which may be represented by a single video object plane.

In more familiar terms, a camera views a scene and captures the information by sampling the scene (by scanning, or shuttering and scanning). The camera provides its output as a sequence of frames or, in MPEG-4 terminology, the texture part of a sequence of VOPs.

A VOP contains texture data and either rectangular shape information or more complex shape data associated with the object. VOPs, like frames in earlier versions of MPEG, may be coded with intradata, or by using motion compensation; this is discussed below.

Next up is the optional *group of video object planes* (GOV) that groups together video object planes. GOVs are similar to the groups of pictures (GOPs) in earlier MPEGs, and provide points in the bitstream where VOPs are encoded independently from each other, and so provide random access points into the bitstream.

The *video object layer* (VOL) permits scalable coding of a sequence of VOPs or GOVs. Scaling may be spatial or temporal and is discussed below. Multiple VOLs correspond to multiple scalings of a sequence. Each scaling is matched to a set of available resources—usually limited bandwidth and/or limited computational power.

The *video object* (VO) level includes everything in the bitstream about a particular video object. It includes all VOLs associated with the object. Multiple video objects are represented by multiple VOs.

Finally, the *video session* (VS) is the top video level of the MPEG-4 scene, and includes all video objects, natural or synthetic, in the scene.

Shape Coding

There are two types of shape for video objects in MPEG-4, rectangular and arbitrary. *Rectangular shapes* are trivial and merely indicate the extent of an image. Note, however, that this still represents a significant increase in flexibility over earlier standards. In MPEG-2, for example, the extent of the image is hard-coded into the bitstream headers. In MPEG-4, the rectangular size of a simple background video object is comparable, but there can still be other rectangular objects (of differing size) in the same session, such as picture-in-picture, and sprites.

Arbitrary shapes are the MPEG-4 equivalent of the graphics alpha channel. The shape again represents the extent of a video object, and at any point in the image plane it determines whether its associated object may be visible. The shape is defined over a rectangular area, known as the *mask*, sized to accommodate the greatest horizontal and vertical dimensions of the object itself. Both the horizontal and vertical sizes of the shape mask are multiples of 16 pixels.

Arbitrary shapes may be coded as binary data or grayscale data. Binary shapes are the simplest and indicate whether the object is transparent or opaque at any given point. The edge between the object and background is always hard. This normally represents a violation of Nyquist, and aliasing is likely to be visible. Grayscale shapes, on the other hand, permit a smooth, bandwidth-controlled transition or "blend" between objects and can yield much greater realism.

Binary shapes are used for simple applications and can represent total transparency or total opacity. In image areas where the video object is determined to be transparent, it cannot be seen under any circumstances. The background, or the composite formed by the background and any lower-level video objects, will be seen (unless occluded by some other higher-level object). Where the video object is opaque, the background and any lower-level objects will be invisible, and the video object associated with the shape will be visible, unless occluded by a higher-level object (something in front).

Binary shapes are coded into blocks 16 pixels square, like the macroblock used for textures, but they are known as *binary alpha blocks* (BABs). There are three classes of block in a binary mask; those where all pixels are transparent (not part of the video object); those where all pixels are opaque (part of the video object); and those where some pixels are transparent and others opaque. Coding of the first two cases is trivial. Blocks of the third type are those representing the boundary of the video object and are coded using techniques derived from arithmetic coding, discussed briefly in Chapter 3.

The algorithm used is known as *context-based arithmetic encoding* (CAE), extended to include the use of motion estimation. As in the conventional arithmetic encoding algorithm, coding is based upon a continually updated probability estimate. In the basic interCAE, the probability estimate is computed from ten pixels, above and to the left of the pixel being coded, see Figure 11-2a. For intraCAE, where motion estimation and prediction are used, the context includes some pixels from the current frame and some from the reference frame, as seen in Figure 11-2b.

Figure 11-2a
Context for intraCAE.

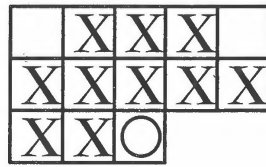


Figure 11-2b
Context for interCAE.



Reference VOP

Current VOP

Grayscale shapes are represented much as a luminance signal is, usually quantized to eight bits, with values from 0 (totally transparent) to 255 (totally opaque). The shape is coded using a philosophy similar to that used for binary shapes. Again there are macroblocks that are totally outside the boundary of the object, and those that are totally within the boundary of the object. Blocks outside are marked as “all zero” (fully transparent); those inside are marked as “all 255” (fully opaque). Blocks that are neither fully transparent nor fully opaque must be coded in a manner similar to that used for texture; that is, motion compensated DCT.

Texture Coding

Texture coding, the MPEG-4 term that corresponds to coding of conventional moving-image data, is built on MPEG-2 coding with extensions and improvements, as MPEG-2 was built on MPEG-1 and JPEG. In fact, a rectangular VOP is the closest equivalent to a “video frame” in pre-MPEG-4 terminology. The same concepts of intra- and intercoding apply, and video objects may be coded with I-, P-, and B-VOPs. All MPEG-4 profiles, except for the studio profiles described below, use a 4:2:0, *YUV* representation of video object textures.

In MPEG-4, not all video objects are the same size, and texture coding is necessary only for those areas that are part of the object. For rectangular objects this is simple; the size must be a multiple of 16 pixels

(one macroblock) in each direction, and all macroblocks are processed. For objects with a more complex shape, the boundary is defined by the shape signal. The extent of the object is still defined by a rectangular array of macroblocks, but texture coding is performed only for those macroblocks wholly or partially inside the object boundary.

I-VOPs are coded much as MPEG-2 I-frames are, but several new techniques have been introduced to improve efficiency. In Chapter 4, we looked at predictive coding, and in Chapter 10, we saw that MPEG-2 used the simple $\hat{X} = A$ predictor for encoding DC coefficients and motion vectors. Chapter 4 also showed that there are much better predictors, and that the performance of a simple adaptive predictor was substantially higher than that of other one- or two-dimensional predictors. MPEG-4 uses an adaptive predictor for the DC values. The predictor measures horizontal and vertical brightness gradients and predicts the DC value from the block above or the block to the left, in the direction of the lesser gradient.

Just as the correlation of the image provides benefit by predicting DC coefficients, it can also help in the coding of some AC coefficients. Areas of similar texture (in the conventional sense of the word) generate similar arrays of AC coefficients after the DCT transform. The similarity (and, therefore, the coding benefit) is greatest in the most significant coefficients; those that represent the greatest energy of the texture. These coefficients are normally the non-zero coefficients in the first row and/or first column. These are also the coefficients that will be quantized least aggressively and use up the most bits for transmission and thus offer the greatest potential for improved coding.

In MPEG-4, the AC coefficients of either the first row or the first column are predicted from those of the block immediately above, or immediately to the left, or above-left.

Quantization of the coefficients is similar to the method used by MPEG-2, but the mechanisms for coefficients scanning, and for variable-length encoding, are both improved.

The chosen method for coefficient readout is determined by the DC prediction. When there is no DC prediction, the zigzag scanning, as described under MPEG-2, is used. If the DC coefficient was predicted from the block to the left, *alternate-vertical* scan is used, a scanning system that emphasizes reading out the vertical direction first. If, however, the DC coefficient was predicted from the block above, *alternate-horizontal* scan is selected, which results in a readout of coefficients with an emphasis on the horizontal direction first.

To improve the efficiency of variable-length encoding, two different VLC tables are provided, and the quantization level determines which table is used. The VLC codes themselves are reversible. This is part of the error-resilience strategy of MPEG-4; if there is an error in the received bit stream, decoding can continue up to the error. Data after the error may be decoded by starting at the end of the block and decoding the VLC codes in reverse until the error is reached.

Boundary Coding

The ability to code objects of arbitrary shape creates an interesting situation at the object boundaries. Blocks that lie outside the boundary of the object need no texture coding. Those that are totally inside the boundary are coded normally according to the techniques we have discussed. Texture coding is needed for boundary blocks, but the object only exists for part of the block. If we chose to make the pixels outside the boundary of the object black, this would introduce a good deal of coefficient energy when the block was DCT transformed, yet this energy would not represent a meaningful part of the image. To avoid this inefficiency boundary blocks are *padded*. First, all pixels that are not part of the object are given a value equal to the average value of all the pixels that are part of the object. The padding is refined by stepping through the pixels outside the object and performing a correction based on the average value of any neighbors that lie within the object. Note that no values within the object are changed. Changes to values outside the object do not affect the final result because these pixels will never be visible. The process described minimizes the energy of the coefficients when the block is DCT transformed.

Coding of Arbitrary-Shaped Video Objects

Now that we have looked at coding the different elements, it is worthwhile to examine all the possible cases for block coding within the mask of a video object.

In the simplest possible case, the block is flagged as transparent in the shape part of the VOP. This means that all the pixels in the block lie outside the boundary of the object, so that no texture coding is required.

The block may be flagged as opaque in the shape part of the VOP. In this case, all pixels in the block lie inside the boundary of the object. No boundary coding is required, but texture coding is needed. The texture may be intracoded or predicted with motion compensation, as appropriate.

The remaining blocks are in some way part of the boundary area of the object. In the case of a binary shape this means there are some pixels that are transparent and some that are opaque. In the case of a grayscale shape, it means that not all pixels are transparent and not all pixels are opaque. Here both shape and texture must be coded. Coding of both the shape and texture may be intra or predictive.

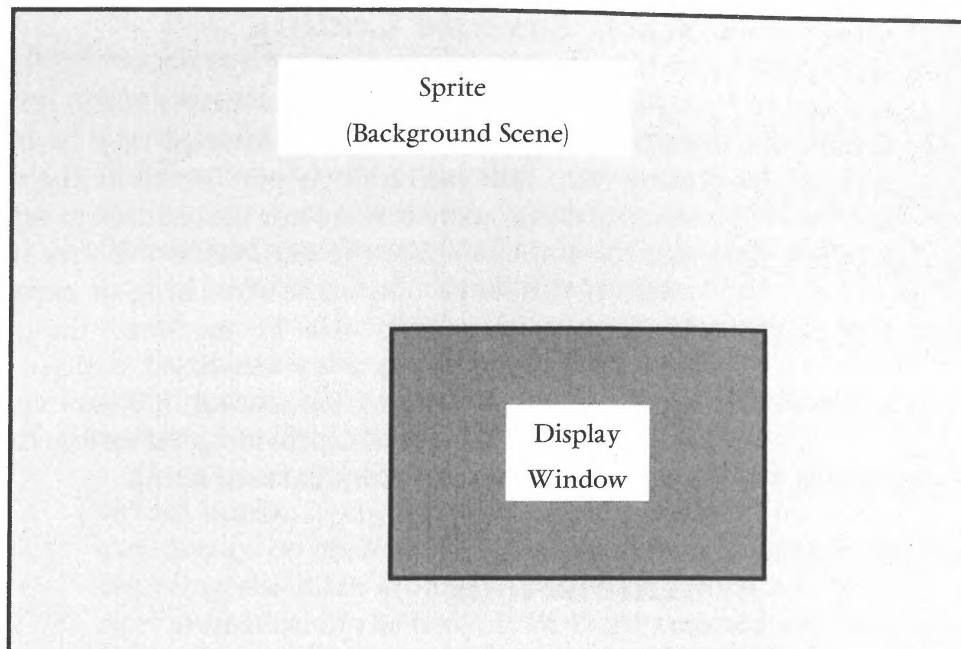
Just as in earlier systems, the syntax must support many coding possibilities. Any block may be intracoded; in a P-VOP or B-VOP, blocks may be predictively coded. In the simplest case, the *motion vector difference* (MVD) is zero (i.e., the motion-vector prediction is perfect), and the motion vector points to a good match for the block, so that no residuals are required. This block just requires a “skip” code. If the match is not sufficiently good, residuals must be transmitted. In other cases, the MVD will be nonzero and residuals may or may not be required.

Sprites

MPEG-4 has another object type that is particularly useful for backgrounds; the sprite. A *sprite* is a video object, usually larger (perhaps considerably larger) than can be displayed at any one time. It is used for an object (like a static background) that is persistent through a scene. Although sprites are not restricted to games, this application provides their most obvious use. Generally a game scene consists of a background plus a number of (usually) synthetic objects that move according to the script of the game and the actions of the player. During the action, the viewed scene will likely pan around the background; the player sees different parts at different times, but they are all parts of the same static image. MPEG-4 provides the ability to transmit the whole of the background once as a sprite, and to generate scenes by sending cropping or warping information to determine which part of the sprite is visible at any given instant (Figure 11-3). Once the sprite has been transmitted, only the cropping/warping information for the sprite and the foreground objects need be transmitted. In a typical game, each section of the sprite will likely be used many times, so this approach represents a substantial reduction in the total data used.

Figure 11-3

The sprite provides a background scene larger than the display window; once all the sprite has been transmitted, any part of it may be displayed.



Transmitting the whole of a sprite at the beginning may be very efficient, but will need either extra bandwidth or a period of time before the action can start. MPEG-4 has techniques to avoid this problem as well. A sprite can be transmitted in sections, as needed. The opening scene would require that only the area of the sprite needed for that scene be transmitted immediately. If the “camera” then panned to the left, a new strip of information would be transmitted every frame, but these would all be stored at the decoder as part of the sprite. So, if the pan reversed, no new background information would need to be transmitted until the camera moved to the right of the original scene. However, a certain amount of sprite data could be transmitted with each frame anyway, as bandwidth permitted, to continue building the complete image at the decoder. This method still requires that a full screen of data be transmitted before the first scene may be viewed. If this is unacceptable, the sprite may be *progressively* encoded, so that only a low-resolution version is transmitted at first; more area and more resolution are sent later.

Sprites are encoded as luminance plus two color components, as we have seen in JPEG and earlier versions of MPEG, and are always intra-coded, since the image is essentially static. However, I-encoding uses the improved techniques introduced in MPEG-4, discussed in the previous section.

Static Texture Coding

MPEG-4 includes the ability to map a static texture onto a varying shape. For example, a logo or message may be mapped onto a swimming fish. This operation is performed at the decoder; the moving shape and the static texture are transmitted as separate objects. In this case the ability to scale the static texture is very important. The movement of the shape may cause stretching or expansion of the texture far beyond its nominal size. To maximize the quality of this operation, MPEG-4 provides a wavelet-based tool for compression of the static texture. Wavelets are discussed in a later chapter; the important point here is that wavelets provide good scaling capability; objects that are expanded display minimal artifacting.

Animations

As discussed above, one of the great strengths of MPEG-4 is the ability to transmit both natural and synthetic objects that may be composited at the decoder. One of the more interesting capabilities using synthetic objects is *facial animation*. This is another example of mapping a texture onto a moving shape, but in this case the shape is specified by a mesh or 3-D model specified by a number of *nodes*. The position of each node is coded, and again predictive coding may be used to increase the efficiency of the coding as the face-shape changes. The standard includes all the syntax necessary to code the position and movement of the nodes. It also includes higher-level constructs, such as *visemes*, or visual lip configurations equivalent to speech phonemes.

With sufficiently good implementation, this technique opens up some fascinating (or frightening) possibilities. A real or imaginary person may be represented by a mesh and a texture. In combination with speech synthesis, the animated face can be coded to “read” any textual message.

Version 2 of MPEG-4 goes on to add body animation capability. The following extract from the official Overview of the MPEG-4 Standard is reproduced by kind permission of the MPEG Convener. Not only does this short quote give a description of the body-animation capability, but I think it provides an excellent example of the thoroughness of every aspect of MPEG-4.

The Body is an object capable of producing virtual body models and animations in the form of a set of 3-D polygonal meshes ready for rendering. Two sets of parameters are defined for the body: body definition parameter (BDP) set, and body animation parameter (BAP) set. The BDP set defines the set of parameters to transform the default body to a customized body with its body surface, body dimensions, and (optionally) texture. The body animation parameters (BAPs), if correctly interpreted, will produce reasonably similar high level results in terms of body posture and animation on different body models, without the need to initialize or calibrate the model.

Upon construction, the Body object contains a generic virtual human body with the default posture. This body can already be rendered. It is also immediately capable of receiving the BAPs from the bitstream, which will produce animation of the body. If BDPs are received, they are used to transform the generic body into a particular body determined by the parameters contents. Any component can be null. A null component is replaced by the corresponding default component when the body is rendered. The default posture is defined by standing posture. This posture is defined as follows: the feet should point to the front direction, the two arms should be placed on the side of the body with the palm of the hands facing inward. This posture also implies that all BAPs have default values.

No assumption is made and no limitation is imposed on the range of motion of joints. In other words the human body model should be capable of supporting various applications, from realistic simulation of human motions to network games using simple human-like models. The work on Body Animation includes the assessment of the emerging standard as applied to hand signing for the listening-impaired.

Scalability

MPEG-4 provides both spatial and temporal scalability at the object level. In both cases, this technique is used to generate a *base layer*, representing the lowest quality to be supported by the bitstream, and one

or more *enhancement layers*. These layers may all be produced in a single coding operation. The scaling can be implemented in two different ways. When there are known bandwidth limitations, versions of the bitstream may be used that include only the base layer, or the base layer plus lower-order enhancement layers. Alternatively, all layers may be sent and the scaling decision left to the decoder. If the display device is low resolution, or if computational resources are insufficient, enhancement layers may be ignored.

Figure 11-4 shows the concept of an encoder implementing spatial scalability; in this case at just two levels. The input VOP is down-converted to a lower resolution, the base layer. This layer is encoded, and then a decoder constructs the base-layer VOP as it will appear at the decoder. This VOP is then up-converted to the same resolution as the input, and a subtraction process yields the differences from the original image, these are separately encoded in an enhancement-layer encoder. Note that each stream of encoded VOPs forms a video object layer. The base-layer VOL uses both intra- and intercoding, but the enhancement layer uses only predictive coding. The base-layer VOPs are used as references, as shown in Figure 11-5.

Figure 11-4
Spatially scalable
encoder for a single
enhancement layer.

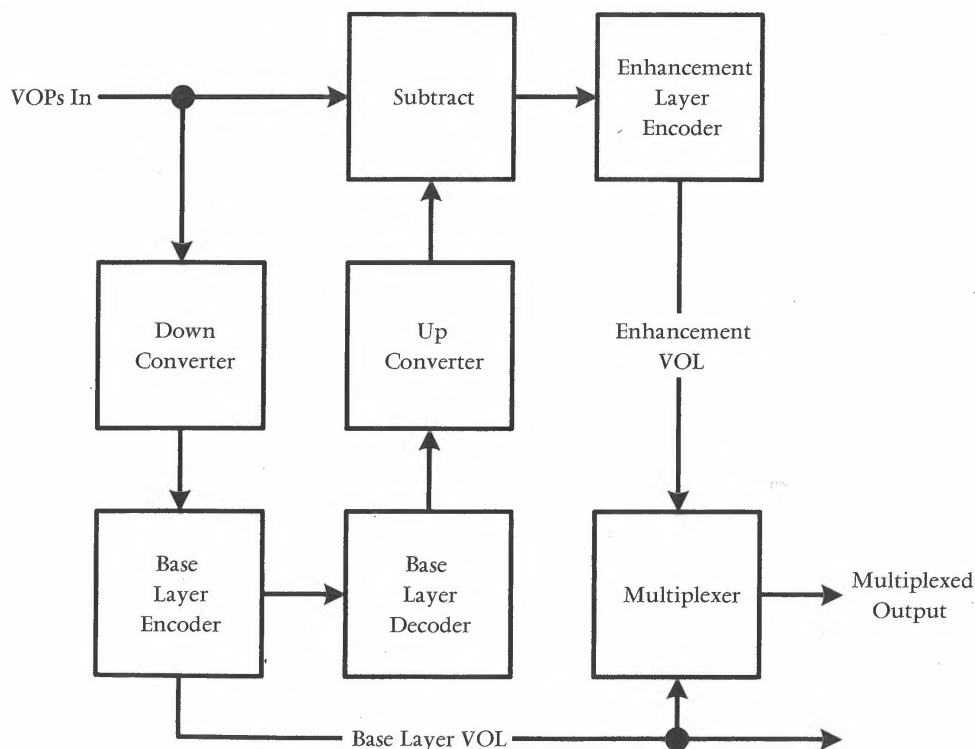
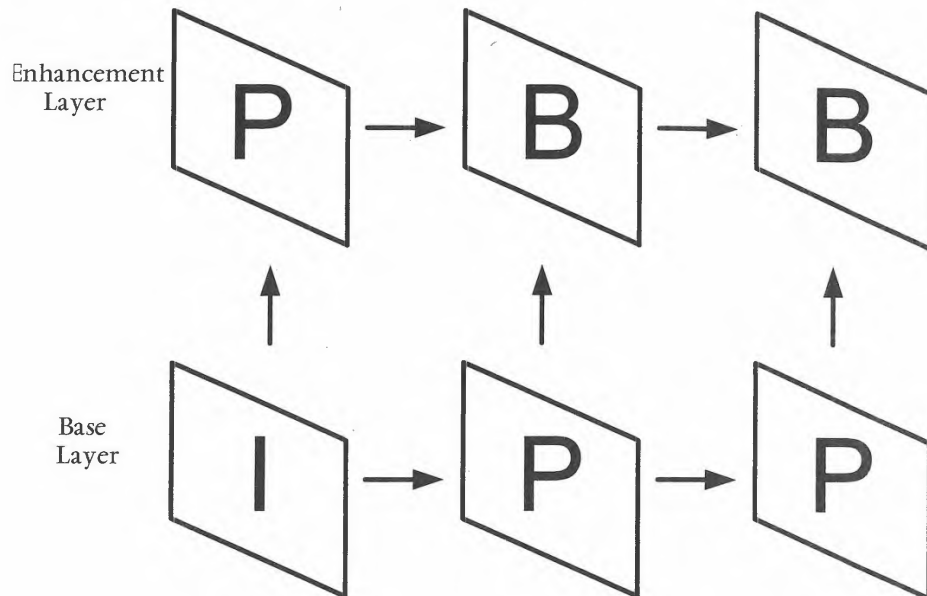


Figure 11-5
Prediction of a spatial
enhancement layer.



Temporal scalability is rather more simple. The stream of incoming VOPs is split. The required number of VOPs (perhaps every third VOP) is sent to the base-layer encoder; the remainder is sent to one or more enhancement layers.

Note that scalability is applied on a per-object basis. This provides a wide range of coding and decoding capabilities. For example, a decoder in a game system might have insufficient computational power to decode all objects at the highest available rate. It might choose to decode only a low rate for the background, accepting some level of jerky motion there, but to decode a higher rate for the principal foreground object, making its motion as smooth as possible.

Advanced Coding Extensions (ACE)

Version 2 of MPEG-4 introduced three new tools to improve coding efficiency for video objects. These are *global motion compensation* (GMC), *quarter pel (pixel) motion compensation*, and *shape-adaptive DCT*. Together, these tools have been shown to improve coding efficiency by up to 50 percent over Version 1, depending on the type of material and the bit rate.

Global motion compensation and quarter pixel motion compensation are fairly self-explanatory. GMC allows the overall motion of the

object to be coded with a very few parameters, and the improved resolution of motion vectors substantially reduces prediction errors and the use of residuals.

Shape-adaptive DCT may be used to improve the coding efficiency of boundary blocks, when not all pixels belong to the image. Instead of using 8×8 two-dimensional DCT on the complete block of 64 pixels, one-dimensional DCT is applied first vertically, then horizontally, but only to the pixels that belong to the object; these are called the *active pixels*.

To start the process, each column of the block is examined, and within each column all the pixels that belong to the object are moved up, so the column is "top-justified." Then, each column is DCT-transformed, but using a DCT matched in size to the number of pixels. For example, if there are five pixels in a column that belong to the object, a 1×5 DCT would be used for that column. If there are no active pixels in the column, no transform is performed. When the vertical transforms are complete, the active pixels are shifted to the left, so that each row is "left aligned." A one-dimensional DCT is then performed on each row that has active pixels, again using the appropriate size of transform.

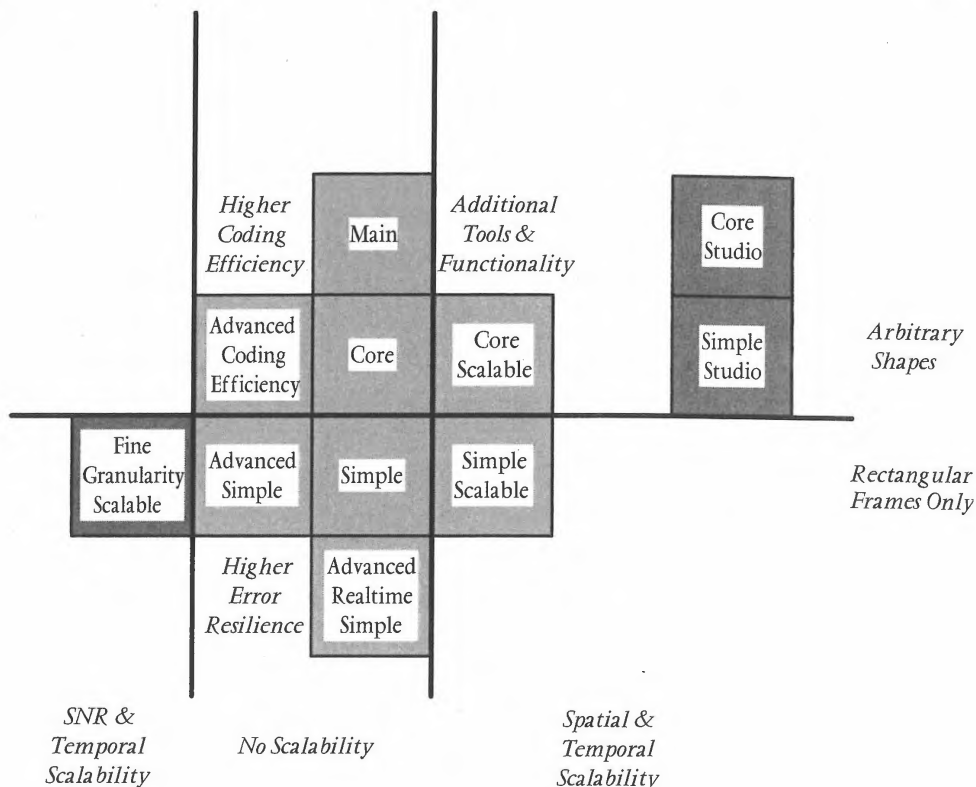
Because the contour of the object is transmitted separately by shape coding, the pixels can be moved back to the "the right place" in the decoder.

Visual Profiles

MPEG-4 includes a structure of profiles and levels using the same philosophy as MPEG-2. As with all aspects of MPEG-4 the range is vast. Rather than paraphrase, I include part of the listing from the Official Overview; this extract covers visual profiles only for Versions 1 and 2. Figure 11-6 shows the relationships among the various profiles and includes the two studio profiles defined by the proposed Amendment 3, and the Fine Grain Scalability profile that is the subject of proposed Amendment 4. These additions to the standard are discussed below. Figure 11-6 is reproduced from the work of Professor Jens-Rainer Ohm of the University of Aachen, with his kind permission.

The visual part of the standard provides profiles for the coding of natural, synthetic, and synthetic/natural hybrid visual content. There are five profiles for natural video content:

Figure 11.6
Profiles in MPEG-4.



1. The *Simple Visual Profile* provides efficient, error-resilient coding of rectangular video objects, suitable for applications on mobile networks, such as PCS and IMT2000.
2. The *Simple Scalable Visual Profile* adds support for coding of temporal and spatial scalable objects to the Simple Visual Profile. It is useful for applications that provide services at more than one level of quality due to bit-rate or decoder resource limitations, such as Internet use and software decoding.
3. The *Core Visual Profile* adds support for coding of arbitrary-shaped and temporally scalable objects to the Simple Visual Profile. It is useful for applications such as those providing relatively simple content interactivity (Internet multimedia applications).
4. The *Main Visual Profile* adds support for coding of interlaced, semitransparent, and sprite objects to the

Core Visual Profile. It is useful for interactive and entertainment-quality broadcast and DVD applications.

5. The *N-Bit Visual Profile* adds support for coding video objects having pixel depths ranging from 4 to 12 bits to the Core Visual Profile. It is suitable for use in surveillance applications.

The profiles for synthetic and synthetic/natural hybrid visual content are:

1. The *Simple Facial Animation Visual Profile* provides a simple means to animate a face model, suitable for applications such as audio/video presentation for the hearing impaired.
2. The *Scalable Texture Visual Profile* provides spatial scalable coding of still image (texture) objects useful for applications needing multiple scalability levels, such as mapping texture onto objects in games, and high-resolution digital still cameras.
3. The *Basic Animated 2-D Texture Visual Profile* provides spatial scalability, SNR scalability, and mesh-based animation for still image (textures) objects and also simple face object animation.
4. The *Hybrid Visual Profile* combines the ability to decode arbitrary-shaped and temporally scalable natural video objects (as in the Core Visual Profile) with the ability to decode several synthetic and hybrid objects, including simple face and animated still image objects. It is suitable for various content-rich multimedia applications.

Version 2 adds the following Profiles for natural video:

1. The *Advanced Real-Time Simple (ARTS) Profile* provides advanced error resilient coding techniques of rectangular video objects using a back channel and improved temporal resolution stability with the low buffering delay. It is suitable for real time coding applications; such as the videophone, teleconferencing, and remote observation.

2. The *Core Scalable Profile* adds support for coding of temporal and spatial scalable arbitrarily shaped objects to the Core Profile. The main functionality of this profile is object-based SNR and spatial/temporal scalability for regions or objects of interest. It is useful for applications such as the Internet, mobile, and broadcast.
3. The *Advanced Coding Efficiency (ACE) Profile* improves the coding efficiency for both rectangular and arbitrary shaped objects. It is suitable for applications such as mobile broadcast reception, the acquisition of image sequences (camcorders), and other applications where high coding efficiency is requested and small footprint is not the prime concern.

The Version 2 profiles for synthetic and synthetic/natural hybrid visual content are:

1. The *Advanced Scalable Texture Profile* supports decoding of arbitrary-shaped texture and still images including scalable shape coding, wavelet tiling, and error-resilience. It is useful for applications that require fast random access as well as multiple scalability levels and arbitrary-shaped coding of still objects. Examples are fast content-based still image browsing on the Internet, multimedia-enabled PDAs, and Internet-ready high-resolution digital still cameras.
2. The *Advanced Core Profile* combines the ability to decode arbitrary-shaped video objects (as in the Core Visual Profile) with the ability to decode arbitrary-shaped scalable still image objects (as in the Advanced Scalable Texture Profile). It is suitable for various content-rich multimedia applications such as interactive multimedia streaming over Internet.
3. The *Simple Face and Body Animation Profile* is a superset of the Simple Face Animation Profile, adding—obviously—body animation.

Scene Compositing and Interaction

Scene Modeling

Although it is not directly related to video compression, it would be wrong to discuss MPEG-4 without some mention of *Binary Format for Scenes* (BIFS). The introduction to this chapter highlighted the fact that MPEG-4 can handle multiple video objects, all transmitted independently, but composited at the decoder. The same philosophy is applied to audio. A complete MPEG-4 scene may include multiple audio objects, natural and synthetic, such as background sound, dialogue, sound effects, etc. There can also be multiple 2-D and 3-D graphical objects in a scene.

BIFS is the language that describes, statically and dynamically, how objects should be brought together at the decoder to form a complete scene. It is based on *virtual reality modeling language* (VRML) with extensions that provide simpler constructs for 2-D objects and coordinate spaces. BIFS provides a hierarchical or tree structure wherein objects may be combined into groups. Individual objects may be manipulated within a group, or the whole group may be manipulated. Nodes of the tree may be added or removed at any time.

Each object and each group have a local coordinate system, addressing both space and time parameters. All of these coordinate systems relate to a set of global coordinates describing the scene, so that any object may be manipulated with respect to itself, or within any layer of a group structure, or with respect to the overall scene. Thus, BIFS provides a timeline within which any part of the scene can be choreographed. BIFS data is sent in a separately identified stream or multiplexed with video or audio sessions.

MPEG-4 Version 2 includes some extensions to BIFS, known as *advanced BIFS*. To demonstrate the richness of MPEG-4, I again quote a paragraph from the Official Overview. To stress that the visual schemes discussed above are just one part of MPEG-4, this extract refers to Advanced BIFS constructs for the audio parts of a scene:

Advanced sound environment modeling in interactive virtual scenes, where properties such as room reflections, reverberation, Doppler effect, and sound obstruction caused by objects appearing between the source and the listener are computed for sound sources in a dynamic environment in real time. Also enhanced source directivi-

ty modeling is made possible enabling inclusion of realistic sound sources into 3-D scenes.

Interaction

MPEG-4 is designed for use in applications where the end-user may interact with the scene presented. The most obvious application example is games, but interaction is a powerful tool that is an essential element of any system where more information is available than can be presented at one time. Interactive television, educational systems, any sophisticated data retrieval system; all need the ability to change the presentation based upon user requests—just as on a Web site.

Interaction may be provided using many devices—mice, remote controls, and keyboards are just a few. Fortunately, the MPEG-4 constructs do not require additional standardization or a guess at future mechanisms for interaction. Any interactive device is required to produce BIFS commands, and these are combined with the BIFS content of the master scene. The decoder need not know whether the commands were part of the original coding or contributed by an interaction mechanism; the same capabilities exist.

Work in Progress

During September 2000, the MPEG committee is completing work on two more amendments to MPEG-4, scheduled to become part of the standard during 2001. As with earlier work, the new amendments are true supersets, extending the application of the Standard into new areas.

Studio Profiles

I mentioned at the beginning of this chapter that MPEG-4 had expanded far beyond its original focus of low-bit-rate applications. *Studio profiles* are intended for use in production environments and for the storage and transport of program material up to the point of transmission to the end-user, where an appropriate profile and level would be chosen for the application. It is important to note that the

MPEG-4 studio profiles are true members of the MPEG-4 system, not just extensions of MPEG-2. In fact, the object-oriented structure is seen as a natural fit with the production process. If, for example, video objects can be kept separate, and their compositing is described separately, there are no irrevocable steps. The original material may be re-edited or repurposed without limitation.

The MPEG-4 studio profiles overlap MPEG-2, in that they include the capabilities of the MPEG-2 4:2:2 profile. MPEG-4, however, goes much further, and the studio profiles include most of the powerful tools and constructs specified for the other profiles, together with extension into higher-definition formats and other areas critical to high-quality production.

The studio profiles add a few more tools to improve efficiency in the high-quality domain. In particular, DCT coefficients may be grouped by value, and variable-length coding uses recursive selection of the most efficient of a number of VLC coding tables. These tools offer an increase in coding efficiency of around 10 percent when compared to MPEG-2.

Two studio profiles are specified. Both provide for 4:2:2 or 4:4:4 coding in either the *YUV* or *RGB* domains. They provide for picture sizes of up to 4k×2k (4000×2000 pixels), progressive or interlace, and pixel depths of up to 12 bits per component. Encoded data rates can be up to 1.2 Gbits/s. Both studio profiles support arbitrarily shaped video objects.

The *Simple Studio Profile* is intended for acquisition and editing environments. The profile permits only intracoding, so that frames or VOPs are always independent, and the video can be accessed at any point by an editor without complication or loss of temporal granularity. The Simple Studio Profile also permits lossless transcoding from the 4:2:2 profile of MPEG-2.

The *Core Studio Profile* is intended for distribution and storage within production environments. It includes all of the elements of the Simple Studio Profile, plus sprites, and permits predictive encoding, using P-VOPs only. There is no bidirectional prediction within MPEG-4 studio profiles.

Fine Grain Scalability

The *Fine Grain Scalability Profile* (FGS), often known as the *Streaming Profile* is intended to support applications and environments where

the bandwidth and/or computational power available cannot be predicted and may vary dynamically. The scaling tools provided by other parts of MPEG-4 are powerful in their own way, but have two significant limitations.

The granularity of scaling using existing tools is necessarily coarse. Earlier, we saw a process that could produce a 2-layer bitstream including a base layer and one enhancement layer. The number of enhancement layers that can be used is arbitrary, but in practical terms is severely limited. Each layer adds coder and decoder complexity and overhead in the bitstream. Also, provision for all the levels of scalability must be built in at the time of encoding; any changes require recoding from scratch.

In contrast, FGS provides a mechanism that permits a single encoding process, producing a bitstream that can be modified subsequently in two different ways. Prior to transmission, the bitstream may be processed to scale it down to known bandwidth limitations. This can be performed dynamically, for example in response to the requirements of a statistical multiplexing system. Downstream distribution points may further reduce the bit rate if necessary. At the receiving end, the decoder can use all of the received bits, or it can scale the bitstream down further to match its capabilities and resources, again dynamically.

Figure 11-7 shows the concept of bit-plane encoding. The quantized DCT coefficients are indicated by the columns of bits. Clearly the most significant coefficient is represented by the tallest column. We can "slice" the bit planes horizontally. If we took only the highest plane in the figure, we would capture only the MSB of the largest coefficient. Our representation of the block would then be a single coefficient with the quantized value 1000 (instead of 1110). This is obviously a very crude representation, but it is better than nothing.² If we were able to take two bit planes, we would get three coefficients, one with the value 1100, and two each with the value 100. The important point is that from a single coding operation we can extract at the decoder as much information as we are able to use from the bitstream, and be assured that we have captured the most significant elements. This approach is also known as *signal-to-noise scalability*.

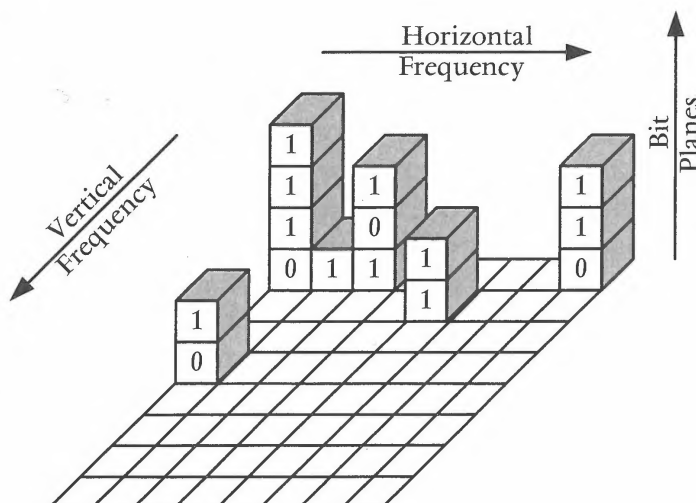
It is possible to modify this technique at the encoder. Some values may be artificially pushed up the bit-plane stack. For example, it is possible to apply frequency weighting to the DCT coefficients,

² As discussed in Chapter 6, the reconstruction value would likely be different. For an effective range of 1000 to 1111, the probable reconstruction value is 1100.

emphasizing the importance of the low-frequency coefficients, or to increase artificially the value of all coefficients that lie within a particular *region of interest* in the object.

The streaming profile is, of course, especially useful in Internet environments. Fine-grain scalability may be used alone, or combined with temporal scaling. A session may be multicast by a server at the highest quality permitted by the bandwidth available. Downstream servers may, if necessary, trim the bitstream to suit lower-bandwidth parts of the network. A decoder uses only as much of the bitstream as its resources permit. It may ignore temporal enhancement layers to minimize the number of VOPs to be processed, and it can take only as much data as it can handle from the other layers.

Figure 11-7
Bit planes in an array
of DCT coefficients.



Conclusion

I think this is the longest chapter in the book, and those who are familiar with MPEG-4 will know that I have only scratched the surface of this Standard and its various amendments. I have tried to cover, at least briefly, those elements of the standard most relevant to video compression, and to convey an impression of the vast scope of MPEG-4. For those interested in learning more, MPEG has become much more open to accessibility of documents (prior to the final Standard, which is controlled by ISO), and its Web site carries many working documents and tutorials.

CHAPTER **20**

**Audio
Compression**

Introduction

This will be a short chapter, partly because of limited knowledge on my part, but also because audio compression is built on many of the same principles as video compression. Only in one sense is audio compression radically different from video compression—it is much older and has been commercially exploited in the analog domain. It was not called *compression*; it was known as *noise reduction*, but essentially these are the same. A noise reducer modifies the signal such that when reverse processing is applied after transmission or recording, the noise of the transmission or recording channel is attenuated. The well known Dolby A, B, and C systems are actually simple, analog, sub-band companding systems—terms we encounter below.

Nevertheless, our interest is on compressing digital audio. Depending on the quality requirements, audio may be sampled at a variety of frequencies, usually in the range 8 to 48 kHz, and with sample resolutions from 8 to 24 bits. The lower limits provide telephone-grade audio; the higher can match or exceed human hearing capabilities. Of most interest in the professional world are two standards at the high end. Compact disc audio is sampled at 44.1 kHz, with a word length of 16 bits for each of two channels, a total of just over 1.4 Mb/s. Professional studio audio is usually sampled at 48 kHz with a word length of 20 bits, almost 1 Mb/s per channel. This rate can accommodate audio frequencies to over 20 kHz with a dynamic range of 120 dB.

As with any form of compression, the objective is to reduce the data by eliminating redundancy. There is “absolute” redundancy—data can be removed by an encoder then unambiguously recreated by a decoder. Removal of this redundancy is lossless compression. There is also perceptual redundancy—data can be removed without significant change to the experience of a human observer.

Perceptual redundancy itself falls into two categories. There are phenomena to which the observer is intrinsically insensitive. In video, we saw that the higher-frequency transform coefficients could be coarsely quantized without significant impact on the viewer. This is true for all conditions. Another effect, used in bit allocation in video systems, is masking. An artifact that might be visible and objectionable in an image area with little activity (empty sky, for example), becomes invisible or insignificant in a busy area of an image.

All these effects exist in audio, and all are exploited by compression systems, but masking is by far the most significant contributor to data

savings in a single-channel audio system. (Multiple channel systems also benefit from correlation between the various channels.) To understand the mechanisms of masking in the human hearing process we must examine the workings of the inner ear.

Masking in Human Hearing

The human hearing system has remarkable performance. Its useful range covers about 10 octaves and, in the more sensitive regions, it provides a dynamic range of more than 100 dB. The discriminating power of the ear—brain combination is incredible; we can resolve conversation and other intelligence from noise levels and interfering signals to a degree that seems to defy the laws of physics. This performance requires some very sophisticated processing.

One of the key elements of the ear is the basilar membrane of the inner ear. It separates two of the fluid-filled chambers of the cochlea and is in contact with the hairs of the organ of Corti. These hairs drive the actual sensors that send nerve messages to the brain. The basilar membrane varies in width, thickness, and rigidity along its length. This makes it frequency sensitive; different areas vibrate at different frequencies. The basilar is not just a passive membrane; it has an active mechanism that can provide positive feedback to low-amplitude vibrations. The combination of these characteristics leads to interesting behavior. (This explanation is somewhat simplistic. Various authorities have suggested different mechanisms for the behavior of the inner ear, particularly in respect to the active attributes. Some suggest nonlinearity in the cochlea; others attribute the active element to the organ of Corti. Suffice it to say that the behavior is complex, certainly not linear, and that there are active elements within the inner ear. We concentrate on the effects of this complexity.)

Some authorities estimate that there are 24 regions in the basilar membrane; others suggest a higher number, but it is certainly finite. Each region can vibrate over a small range of frequencies, but at only one frequency at a time within that range. When positive feedback is applied, the effect is that of a very high-Q tuned circuit. Each region vibrates at a frequency determined by the strongest stimulus within its range, *and is unaffected by any smaller stimuli*. This means that within each frequency band, only the loudest frequency contributes to the signal received by the brain, and thus what we hear.

This effect is the single most significant contributor to our ability to compress audio.

The effect is called *frequency masking*, and it contributes to our ability to compress in two distinct ways. As mentioned above, only the strongest stimulus within a region is significant. Any frequency in the signal in the same band, but lower in amplitude, need not be coded. Also, noise within the band is irrelevant, provided it is sufficiently below the prime stimulus. This allows encoding of the prime stimulus with a relatively small number of bits; the resulting quantization noise is masked just like any other noise. This effect is quite remarkable and, thanks to Dolby Laboratories, there is a demonstration on the CD-ROM that is well worth trying.

The positive feedback of the basilar contributes to another effect, again similar to that of a high-Q tuned circuit. The vibration responds slowly to changes in the amplitude of the stimulus, and this results in *temporal masking*—we fail to hear sounds a short time before, and a longer time after, a strong stimulus.

Simple Audio Compression Schemes

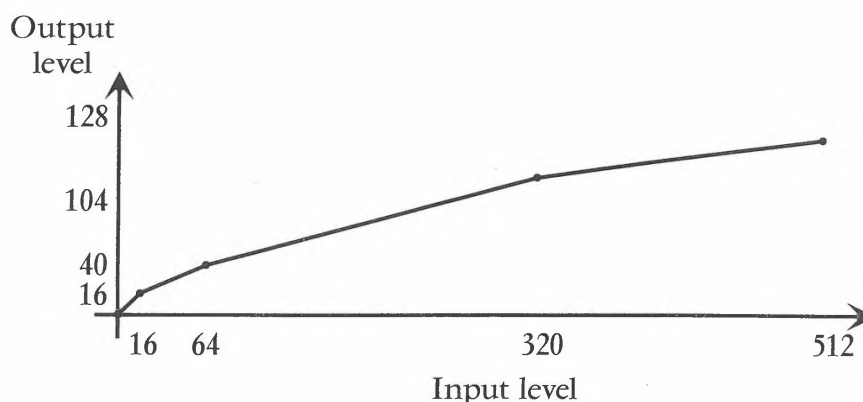
Many of today's consumer devices use some form of audio compression. Sometimes this is used to increase storage capacity. One example is found in 8-mm video recorders; compression is employed to reduce the tape area needed for audio. Audio workstations become much more cost effective if the capacity of the hard drives can be increased by using compression, provided the necessary high quality can be maintained.

In some applications, the compression is used for more than one purpose. The mini compact disc achieves its recording density by means of compression, but data reduction is also used to improve replay under adverse conditions. Data is read from the mini CD at the same speed as from a regular CD—about twice as fast as necessary for normal replay. The data enters a buffer, and after the buffer delay (typically a few seconds) it is decompressed and presented to the listener. If the listener is an energetic jogger and jars the player enough to move the read head, the player can continue to play from the buffer while the head is repositioned, and the additional data speed allows the buffer to be filled again, ready for the next “jog.”

Simple schemes such as these generally use some form of *companding*. The number of bits required by each word is reduced by dynamically altering the level or gain according to the instantaneous amplitude, or the amplitude range of a group of samples.

Video 8, for example, uses a simple four-step gain control based on instantaneous signal amplitude. Low amplitudes are passed at unity gain; higher amplitudes are gain-reduced by factors of 2, 4, or 8. The result looks to a video engineer like a gamma curve (see Figure 20-1). The higher levels are coarsely quantized, but the increased quantization noise is masked by the high signal level. (In the case of Video 8, this mild reduction in data rate is supplemented by an analog compressor.)

Figure 20-1
Transfer function for
Video 8 digital
compression.



More sophisticated companding systems usually divide the input data stream into frames, typically between 1 and 32 ms long. In a 16-bit system, the values for each sample can range from $-32,768$ to $+32,767$. However, it is rare for the sound in one frame to occupy the full range of sample values. If it does not, each sample can be expressed by a shorter word, combined with a gain or scaling factor applied to the complete frame. In other words, the levels are expressed in floating-point form, but with the restriction that the mantissa must have the same value throughout a frame. The NICAM systems, as used in stereo television, use this technique to reduce 14-bit words to 10-bit words, plus a 3-bit mantissa for each frame of 32 words.

There are two real problems with such systems. A frame with a single high sample value can cause low-level audio to be coarsely quantized. This is not usually audible if the frame length is short, typically not more than 1 millisecond. The greater problem is that they do not provide the large compression ratios we would like to see for many applications.

- A typical audio spectrum is made up of many discrete frequencies. With wide subbands, most or all subbands will contain one or more components, and so have to be encoded. With narrower bands, more subbands will fall within gaps in the spectrum and contain no components. These bands do not have to be encoded at all.
- The number of bits needed to encode a subband depends on the degree of masking. We must always consider the worst case, where the stimulus is at the top edge of the band, because of the steep slope of the masking curve below the stimulus frequency. For this reason, narrow bands mask much higher noise levels and can be encoded with fewer bits.

Band-splitting filters are an efficient means of obtaining a reasonable number of subbands, such as the 32 used by many subband encoders. Beyond this, however, use of filters becomes very cumbersome, and a frequency transform is used to analyze the signal. Very large numbers of subbands may be obtained by this method. A 256-subband transform filter has approximately the same complexity as a conventional filter for 32 subbands.

There is a drawback to transform filters. Filters with high-frequency resolution have poor temporal resolution, and this affects the coding of frames that include transients. If the full coding gain is taken, the resulting quantization noise is present throughout the frame. A transient in the frame will probably mask this noise after the transient, but there is very little "negative" temporal masking; thus, the noise may be heard at the beginning of the frame.

To counter this effect, some systems use transient detection and switch to a lower number of bands for frames that include a transient. A correctly designed transform filter bank may be switched quite easily without any significant increase in complexity.

Example Compression Systems

Audio Compression in MPEG

MPEG has standardized on compression schemes based on the MUSICAM system developed by Philips. Three coding layers are defined for MPEG-1, but Layer II is by far the most used.

Layer II supports audio sampled at 32 kHz, 44.1 kHz, and 48 kHz. It uses 24-ms frames (with 48-kHz sampling) and codes 32 equal subbands. For each subband, a 6-bit scale factor is used, giving a range of 120 dB. The scale factor can operate over the complete frame, but can be changed on 8-ms boundaries if required.

This system uses forward adaptive bit allocation and fractional bit quantization. It can code mono or stereo at bit rates of 32 to 384 kb/s. Layer II has been adopted for several direct broadcast satellite (DBS) systems and by the European Digital Video Broadcasting (DVB) system. It is also the standard for multimedia applications such as CD-ROMs.

MPEG-2 added some flexibility to MPEG-1 audio. Half-sample rates were added (16 kHz, 22.05 kHz, and 24 kHz) and provision was made for a multichannel extension. This approach provides backward compatibility with MPEG-1. If multichannel sound is required, a stereo mix must be derived and transmitted as Layer II audio. The additional information is sent in an enhancement stream. Multichannel decoders accept both streams, and remultiplex for the required number of channels. Simple decoders receive only the stereo bit stream and ignore the enhancement layer. Total bit rates of up to 1 Mbit/s are permitted for highest-quality multichannel.

The most well-known audio compression system is, of course, MP3—the designation for MPEG Audio Layer III. This uses more sophisticated algorithms to achieve greater compression efficiency, and is claimed to offer CD quality at about 64 kbits/s per channel. The availability of many thousands of MP3 compressed files on the Internet has done more to change the way we think about entertainment content than any other single development. It is far from certain how the copyright issues will be solved, and who will win the battle for control of digital entertainment distribution, but it is certain that the new techniques are here to stay.

Audio Compression for ATSC

During the process leading to the U.S. ATSC standard, it was decided to use MPEG-2 (MP@HL) for video coding. However, no agreement could be reached on the audio compression system to be used.

One proponent offered a system where six channels were individually compressed, but the main contenders were Philips with an enhanced MUSICAM system and Dolby with its AC-3 system. Both systems offered “5.1” channels. This means five full-bandwidth channels for

front left, front center, front right, left surround, and right surround. An additional channel has only narrow bandwidth and is intended to drive subwoofers, chair-shakers, and other very low-frequency devices. It is also known as the *low-frequency effects* (LFE) channel.

In a bitterly fought contest, expert listeners compared the two systems under carefully controlled studio conditions. AC-3 was selected, on the basis of its slightly superior performance on certain vibraphone passages. Philips contested the decision on the grounds that the MUSICAM system had not been performing correctly at the tests, but the decision remained. In fairness it should be said that when either system is working optimally, expert listeners cannot distinguish the results from the original recording, although Dolby claims this performance level at a lower bit rate. AC-3 supports bit rates from 32 to 640 kbits/s; the ATSC tests were performed at the intended transmission rate of 384 kbits/s (the maximum permissible rate has now been increased to 448 kbits/s, the same maximum as specified for DVDs). This represents a compression ratio of about 10:1, making the quality a remarkable achievement.

AC-3 is a transform system, switching between 256 subbands for normal operation and 128 subbands for transients. It supports the usual sampling rates of 32 kHz, 44.1 kHz, and 48 kHz, and uses 32-ms framing at 48 kHz. It uses a 4.5-bit scale factor over a range of 144 dB and forward/backward adaptive bit allocation.

Dolby incorporated a number of interesting features in AC-3, to be known as Dolby Digital in the commercial world. Although the system can encode up to 5.1 channels, the decoder can produce whatever downmix is appropriate for the receiver capabilities. Decoders can downmix to mono or to stereo or, of course, support full 5.1-channel output. In addition, each Dolby Digital (DD) decoder incorporates a Dolby Surround Pro Logic (DSPL) decoder to support the most common surround-sound installations. In fact, Dolby Digital includes a specific pass-through mode for DSPL encoded audio. The two DSPL signals are carried on the left and right front channels of the 5.1 system and passed directly to the DSPL decoder.

It is important to recognize the existence of this mode. Many videotapes, particularly film-tape transfers, have DSPL audio. DSPL is an analog compression scheme that carries center and surround information coded into the left and right channels; it is a four- to two- to four-channel system. It might be assumed that a DSPL signal could be decoded to four channels and fed to a DD system, perhaps feeding the surround signal to both left and right rear channels. This would

work if there were five loudspeakers at the receiver, but there is a hidden trap.

DSPL is an analog system where the matrixing can never be perfect, so the system is designed with this in mind. Specifically, the system is designed to avoid sensitivity to phase variations in high frequencies in the surround channel. Also, DSPL is not a true surround-sound system in that it does not attempt to place sounds accurately in a 360° stage. It provides accurate location in front of the listener, plus a general (and very effective) ambiance from behind. As a part of this process the DSPL decoder low-pass-filters and delays the derived ambiance signal. Provided the four signals are then conveyed transparently to four loudspeakers, all will be well.

However, if the DSPL signal is decoded and fed to a DD system and decoded at a receiver with only (two-channel) stereo capability, the DD decoder downmixes for the stereo output. If the signals it mixes include the filtered and delayed surround from a DSPL decoder, the downmix will be quite wrong and objectionable.

To avoid this problem, DD provides the DSPL mode described above. The two DSPL-encoded channels are fed to the left and right front channels of the DD system, and the “message” that this is DSPL is passed with the bit stream to the decoder, which passes the two channels directly to the incorporated DSPL decoder. No downmix is required for stereo, and the two DSPL channels can be combined for mono.

Dolby Digital also addresses the perennial problem of loudness variation between programs and between channels. The system uses dialog level to normalize receiver volume; the bit stream carries an indication of the dialog level and the decoder gain is varied to keep this essentially constant.

Finally, DD allows for adjustable dynamic range at the decoder. Wide dynamic range is perfect for home theater environments, but quite unsuitable for a portable television in a noisy kitchen. The decoder can reduce dynamic range as determined by the receiver manufacturer—or according to the preferences of the listener if user controls are provided—with separate control of compression above and below the nominal dialog level.

Some issues surrounding the use of DD for digital television have not yet been resolved. One is the metadata (descriptive data) that should accompany the audio to the encoder. The encoder must know if its input is DSPL, so as to invoke the pass-through mode. It also needs to know the nominal dialog level and the dynamic range setting in order to convey these to the receiver. This presents a practical issue

in a television plant where no obvious mechanism exists for routing, recording, and distributing the metadata with the audio.

Initially most potential users assumed that DD would be used for network distribution as well as final transmission to the home. However, fades, voiceovers, and the like cannot be performed on the coded signal, and such operations in a local TV station require decoding and re-encoding. As with any compression system, this results in quality loss. It has been suggested that DD coded at a higher rate than that used for transmission could be used for distribution to provide the necessary quality headroom. However, it is possible that a different coding system, Dolby-E, may be preferable for distribution; this is discussed briefly in Chapter 22 under Mezzanine Compression Systems. As with many aspects of digital television, implementation is very slow, and new decisions are being made all the time.

CHAPTER **21**

Streaming Media

Introduction

Streaming is a technique used to deliver multimedia content, often including video and audio, usually compressed. It is used on networks, including private intranets and the Internet. Streaming brings together characteristics of two fundamentally different delivery systems.

In the world of digital studio installations, both video and audio, connections between pieces of equipment are usually made by dedicated circuits. These circuits carry the bitstream, usually uncompressed, in real time with no significant transit delay. They are deterministic and isochronous—bits go into the circuit at a fixed rate and bits come out at the same fixed rate. For every bit that goes in, one comes out. The data is received by hardware elements designed specifically for the communication circuit being used, and every usable element, be it an audio sample or a video pixel, is processed as it arrives.

In this scenario there is really no concept of a beginning or end; the bitstream may last for a second, or for many years. The transmitting device sends the information it has; the receiving device accepts the information as it receives it. Even when there is no real information to send—if the video is black or the audio silent—as long as the equipment is powered, it will faithfully send the codes for black or silence, and they will be diligently received.

In the world of computers, the usual method of transferring information is a file transfer. One application creates a file and causes it to be stored on some medium. Another application may access the file from its original storage location, or some separate mechanism may cause the file to be transferred to a different storage location before the second application accesses it. The transfer mechanism may move a physical medium like a floppy disk (“sneakernet”), but today it is more likely that the file is transferred over some form of network. The file may be transferred quickly or slowly, perhaps in many parts, and the parts may not arrive in the same order as they were sent. It is the job of the transfer mechanism to ensure that when the job is finished, the received file is the same as the original.

Whatever the mechanism, the process is essentially serial. The file is created, then moved, then accessed. If there is no information to send, no file is created. No access to a file is possible until the transfer is complete.

This is the mechanism used, for example, when downloading an MP3 file from a music distributor. The file is transferred over the Internet from the server to the user’s hard drive. Only when the transfer is complete can the file be accessed by an MP3 player.

The first system described above is really a very specialized form of streaming, but the term today is generally used to refer to transmission of content over a network. What makes streamed data special is that the content may be accessed and used without waiting for the completion of a file transfer. An example of streaming is the sample audio clips offered by on-line vendors of CDs. Many offer the ability to hear a few seconds of one or more tracks before making a purchase decision.

Streaming in this environment is not the same as the isochronous studio interface. We must assume a connection that supports a rate at least close to the data rate of the clip being played. If the clip is encoded at 384 kbits/s and we attempt to stream over a 20 kbits/s link, the best we could offer is one second of audio followed by a 20-second wait, then one more second of audio. In this case, waiting for a complete file transfer and playing the clip from the hard disk is the only practical solution.

But even if the average link speed is sufficient to support approximately real-time transfer, the transfer rate will rarely be constant. Delivery via the Internet generally involves many links and many routers. Transmission is usually "bursty" and packets may not be delivered in the order that they were transmitted. Clearly buffering is needed to permit an acceptable presentation. An intelligent player application will adjust the buffer time according to the characteristics of the incoming data; if the buffer empties (resulting in a break in presentation) the buffer size will be increased to make future interruptions less likely.

Applications for Streaming Media

The usefulness of streaming media is totally dependent on the availability of adequate bandwidth. The comments that follow are based on the environment as seen late in the year 2000. I make some pessimistic predictions below, and I hope I shall be proved wrong, as most commentators are! Time will tell.

In corporate intranets this may not be an issue: 100 Mbits/s Ethernet is starting to replace 10 Mbits/s, star network topology is rapidly replacing loops, and switches are replacing hubs. All of these factors help to increase the bandwidth available to a user and to reduce bottlenecks in the network. Streaming video over corporate networks is viable today, and the situation is continually improving.

Away from this environment the situation is much less encouraging. Audio streaming is useful on the Internet. The standard home connection is a 56 kbits/s modem, often achieving about 40 kbits/s. Modern techniques of audio compression can offer acceptable speech quality at 10 kbits/s or less and reasonable music quality from about 20 kbits/s. The standard connection, and the Internet itself, offer sufficient overhead for these rates to be supported quite consistently.

As mentioned above, e-commerce vendors can offer music samples that genuinely improve the shopping experience. Congressional hearings and shareholder meetings are now routinely "broadcast" over the Internet.

Video is a very different story and has, to date, achieved little more than novelty value over the Internet. High-quality, full-screen, full-refresh-rate video still requires megabits per second. Improvements in compression technology will help, but there is no way we will be able to deliver serious video in a few hundred kilobits per second. While we are confined to connections with even lower bandwidths, Internet streaming video will likely remain postage-stamp sized, and/or have very a low refresh rate.

There are useful applications for streaming video. As mentioned above, corporate intranets can provide a suitable infrastructure. Aside from this, video conferencing and distance learning are attractive applications that can deliver real benefits even today, but the value is greatest where the video is a supplement to other content (audio, faxed documents, prepublished hard copy material, electronic slide presentations, etc.). Generally videoconferencing and distance learning use leased network connections or dial-up ISDN, offering at least 128 kbits/s. Even at this, quality is low, update is slow, and either the video is delayed with respect to the audio, or both are delayed, making conversation difficult.

We hear talk of the "bandwidth explosion." Will the situation improve soon? Personally I think not, even though there is a great deal of unused fiber on long-distance routes. High-speed home connections are available in the form of DSL and cable modems. These certainly offer an improved experience, but mainly because there are so few of them. There is little evidence that the switching infrastructure supporting these connections is growing at anything like the necessary pace. More and more Web sites are offering content that requires more bandwidth. I see an explosion of bandwidth demand and creeping infrastructure bandwidth growth.

The spread of multicasting technology will certainly help with applications such as "live" broadcast. Today, almost all Internet stream-

ing relies on unicast operation—the server must deliver a separate bitstream to each individual user, addressed appropriately, even if thousands of users are watching a live event. When more multicast technology is available, servers will be able to deliver a single bitstream to which multiple users may subscribe.

Standards for Streaming Media

The first standard to achieve widespread adoption was H.261, an ITU Standard approved in 1990. It is intended for use on ISDN circuits and uses multiples of 64 kbits/s. It supports the common image format (CIF) at 352×276 pixels and the quarter-size QCIF at 176×144 pixels. It uses a combination of motion compensation, DCT, and VLC, and can code at rates up to 30 frames/s. Unlike MPEG, H.261 and other streaming media standards permit frame dropping as a means to control bit rate. It is very common in video conferencing systems to see the image freeze when some rapid motion, such as a pan, occurs.

H.263 followed in 1996. Initially, this standard was intended for very low bit rates only (up to 64 kbits/s), but this restriction was removed and the standard is likely to replace H.261. It employs improved coding algorithms and supports QCIF and SQCIF (approximately half-QCIF). As an option, implementations can support higher resolutions up to 16 CIF at $1,408 \times 1,152$ pixels.

More recently, the H.323 series of standards provides higher quality over corporate intranets and to institutions with second-generation Internet connections. Operation is typically in the hundreds of kilobits/s range. The higher quality offered by these standards and bandwidths is particularly attractive to academic and medical conferencing.

In the world of personal computers there is intense rivalry among three companies, Apple, RealNetworks, and Microsoft. The original format, and still very popular, is *Quicktime* by Apple. This format supports a wide range of bit rates and compression schemes—nowadays almost anything can be made into a Quicktime movie. On the CD included with this book you will find commercial stock footage at full resolution, and low bit rate preview versions of the same clips; both are supplied as Quicktime movies. There is a free version of the Quicktime player, but Apple does not seem to offer any free version of the encoder.

RealNetworks range of products including *RealAudio* and *RealVideo* have probably shown the fastest growth in recent years as many web sites have added audio and video elements. As mentioned previously, we now expect to hear audio samples from CDs before we buy them online. The RealNetworks solutions have proven very attractive at the low bit rates available on today's Internet. RealNetworks offer a free basic player, and a trial version of the encoder is also available for download without charge. More sophisticated players, and fully functional encoders, are available for a price.

The most recent major contender is Microsoft Corporation. It has mounted a major effort to redefine streaming media with the *Advanced Streaming Format* (ASF), offering end-to-end solutions. Microsoft offers a complete range of tools including players, encoders, servers, and supplementary tools at its *Windows Media* site. Microsoft's new Media Player, available at this site and included with the latest operating systems, supports a wide range of coding formats (except Real), and the latest encoder supports MPEG-4 (Version 1 at the time of writing). The CD includes sample Windows Media files, at audio/video bit rates ranging from 28 kbits/s to 2Mbits/s. Microsoft's determination to become a substantial player in the field of streaming media may be deduced from the fact that (at the time of writing) *all* of these tools are available without charge.

Some of the tools mentioned above will be found on the CD-ROM included with this book; others may be obtained from the respective Web sites. In all cases, it would be wise to check the Web sites for the latest versions of these products.

Microsoft Corporation has mounted a major effort to redefine streaming media, with *Advanced Streaming Format* (ASF) offering end-to-end solutions. Microsoft offers a full range of free tools, including encoders, players, and servers, plus a wealth of information at its Web site. Microsoft's new Media Player, available at this site, and included with the latest operating systems, supports a vast range of coding formats (except Real), and the latest encoder supports MPEG-4 (Version 1 at the time of writing).

Some of these tools are on the CD-ROM included with this book, but check out the Web sites for latest versions.

 Update 2003

The technology of streaming media has come a long way since this chapter was written. MPEG-4 implementations have been tried, but largely abandoned as proprietary technology outperformed the MPEG standard. Now things are changing again. The JVT codec, discussed in Chapter 12, is expected to perform well in streaming applications. This codec does require considerable processing power, even for the decoder, but it will likely find be used soon in streaming applications if the licensing terms are attractive. Microsoft is making another bid for the streaming market with Windows Media 9 (see Chapter 13) with a simpler codec, performance believed to be comparable to JVT, and aggressive licensing terms.

The good news is that McGraw-Hill has published *Streaming Media Demystified* by my friend Michael Topic (see bibliography). This should provide an excellent source of information for those who wish to know (much) more about the subject.

DIGITAL VIDEO COMPRESSION

SYN



YOUR ONE-STOP GUIDE TO THE ART AND SCIENCE OF DIGITAL VIDEO COMPRESSION!

CD-ROM WITH ALGORITHMS, ENCODERS, AND PRACTICE CLIPS

Without video compression, video streaming—one of the Internet's great innovations—would be nothing more than a pipe dream. That's why if digitizing video imagery is part of your work or hobby, here's an info-packed volume you need handy at all times: McGraw-Hill's *Digital Video Compression*.

Written by a renowned expert in digital technology, *Digital Video Compression* offers comprehensive coverage on a vast range of digital media technologies and standards. It delivers everything you need to understand the techniques, tricks, and confusing protocols of compression, as well as:

- Fundamentals of compression
- MPEG-2 and MPEG-4 compression standards
- JVT / H.264 / MPEG-4 Part 10—learn the details of the just-released compression standard
- And much more!

Plus, you get an invaluable CD-ROM of compression tools, encoders, and video clips for practice. With *Digital Video Compression*, you're sure to stay out in front of this exploding technology!

COVERAGE INCLUDES

- What Is Compression?
 - An Introduction to Images
 - Entropy Coding
 - Protective Coding
 - Transforms
 - Quantization
 - JPEG
- Motion Compensation
 - MPEG-1
 - MPEG-2
 - MPEG-4
 - JVT / H.264 / MPEG-4 Part 10
- MPEG-7 and MPEG-21
- Pro-MPEG and MPEG Operating Ranges
 - DV Compression
 - Wavelets
 - JPEG2000
- Audio Compression
- Streaming Media

The McGraw-Hill Companies

P/N 142494-6 PART OF

ISBN 0-07-142487-3



5 4 4 9 5



9 780071 424875



6 39785 50689 8

Visit McGraw-Hill Education at:
www.books.mcgraw-hill.com

Cover design: The Gazillion Group

\$44.95 U.S.A. | £29.99 U.K.

Web Graphics/Video

