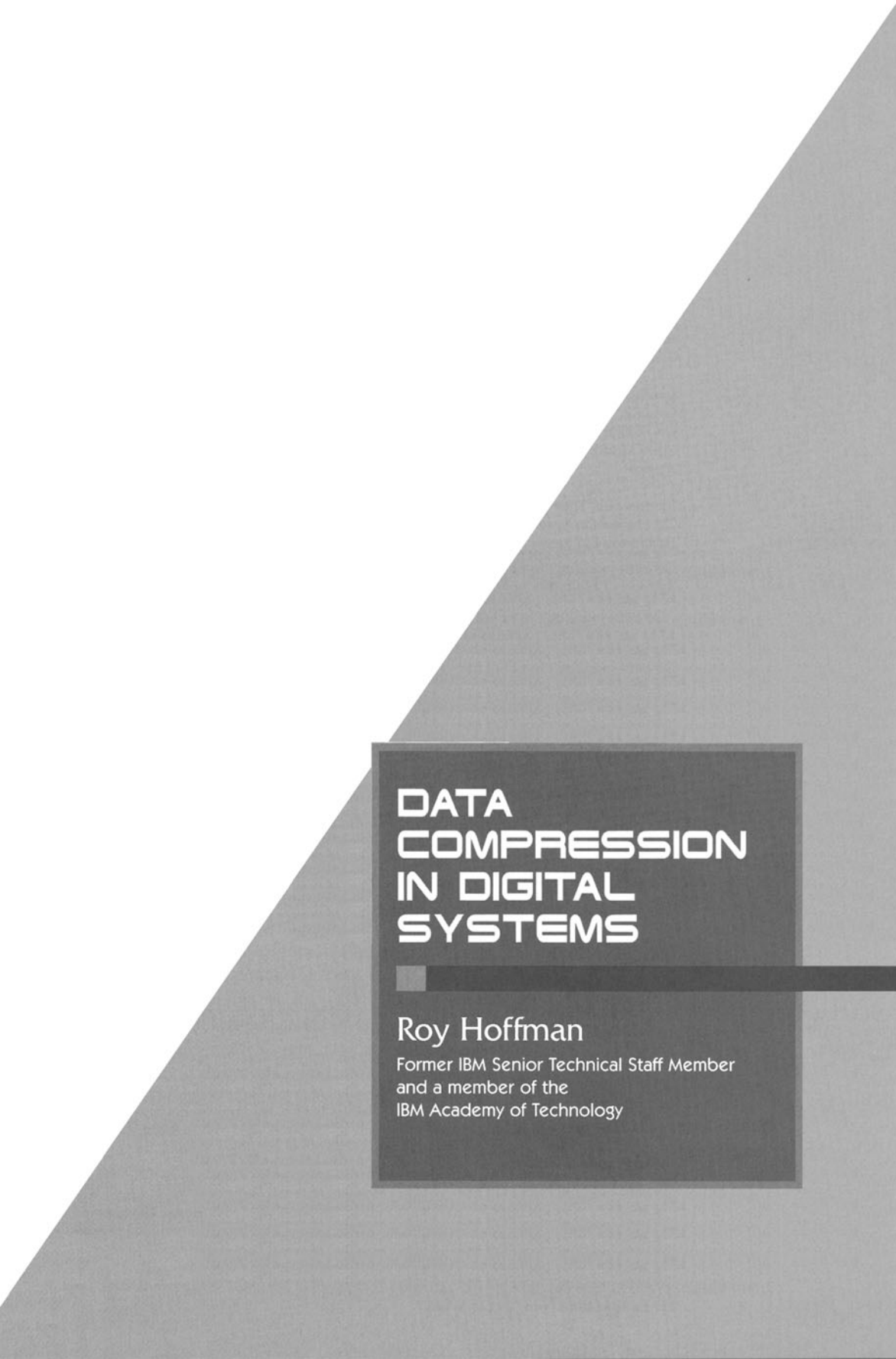


Digital
Multimedia
Standards
Series

DATA COMPRESSION IN DIGITAL SYSTEMS

Roy Hoffman

SNAP-1012
IPR2025-01014



DATA COMPRESSION IN DIGITAL SYSTEMS

Roy Hoffman

Former IBM Senior Technical Staff Member
and a member of the
IBM Academy of Technology

Join Us on the Internet

WWW: <http://www.thomson.com>

EMAIL: findit@kiosk.thomson.com

thomson.com is the on-line portal for the products, services and resources available from International Thomson Publishing (ITP). This Internet kiosk gives users immediate access to more than 34 ITP publishers and over 20,000 products. Through *thomson.com* Internet users can search catalogs, examine subject-specific resource centers and subscribe to electronic discussion lists. You can purchase ITP products from your local bookseller, or directly through *thomson.com*.

Visit Chapman & Hall's Internet Resource Center for information on our new publications, links to useful sites on the World Wide Web and an opportunity to join our e-mail mailing list. Point your browser to: <http://www.chaphall.com> or <http://www.thomson.com/chaphall/electeng.html> for Electrical Engineering

A service of ITP[®]

Digital
Multimedia
Standards
Series

DATA COMPRESSION IN DIGITAL SYSTEMS

Roy Hoffman

Former IBM Senior Technical Staff Member
and a member of the
IBM Academy of Technology



SPRINGER-SCIENCE+BUSINESS MEDIA, B.V.

Cover design: Curtis Tow Graphics

Copyright © 1997 by Springer Science+Business Media Dordrecht
Originally published by Chapman & Hall in 1997
Softcover reprint of the hardcover 1st edition 1997

All rights reserved. No part of this book covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without the written permission of the publisher.

2 3 4 5 6 7 8 9 10 XXX 01 00 99 98

Library of Congress Cataloging-in-Publication Data

Hoffman, Roy (Roy L.)

Data compression in digital systems / by Roy Hoffman.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-4613-7764-1 ISBN 978-1-4615-6031-9 (eBook)

DOI 10.1007/978-1-4615-6031-9

1. Data compression (Computer science). 2. Digital electronics.

I. Title.

QA76.9.D33H64 1996

005.74'6--dc20

96-9522

CIP

British Library Cataloguing in Publication Data available

Warning and Disclaimer

The views and opinions expressed in this book are solely the author's and should not be attributed to any person or organization.

This book is designed to provide information about applying data compression in digital systems. The information is provided on an *as is* basis, without warranty of any kind. Although every effort has been made by the author and publisher to make this book as complete and as accurate as possible, neither the author nor Chapman & Hall shall be liable to the purchaser or any person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly arising out of use of the information contained in this book.

Some of the clip art contained herein was obtained from CorelDraw™.

To order this or any other Chapman & Hall book, please contact **International Thomson Publishing, 7625 Empire Drive, Florence, KY 41042**. Phone: (606) 525-6600 or 1-800-842-3636. Fax: (606) 525-7778. e-mail: order@chaphall.com.

13

Digital System Design

In Part III of this book, we discovered that many applications and digital systems take advantage of data compression, be it for storage savings, bandwidth reduction, or transmission time savings. Incorporating data compression in a digital system requires careful design to assure it will provide the intended benefits and operate harmoniously with other system functions. In this chapter, we introduce decisions that designers face when choosing to integrate data compression in their digital systems.

Digital systems of varying complexity have been encountered throughout this book. From a physical perspective, there are simple self-contained systems, such as consumer products. There are more complex systems composed of several components, such as computer systems, and there are distributed systems, really a system of many interconnected systems. Table 13.1 provides some examples. Whatever the case, the design process starts (or should start) with the marketplace. Inputs from the marketplace define the need for data compression, if there is one. Marketplace requirements are the foundation for generating system requirements and specifications. They essentially establish the constraints and framework within which system designers must work, including the type of data compression that is acceptable and its impact on and visibility to end users.

Designing a digital system involves making trade-offs. System designers must wrestle with many factors. These include architectural considerations and compatibility with prior implementations for similar products. Also, software and hardware design, size, and packaging considerations. Then there are system performance requirements, application functional requirements, product cost considerations, and the list goes on. When data compression is part of the solution, system designers must first understand the problems it addresses, followed by a

Table 13.1 Digital systems.

Type	Structure	Examples
Simple	Self-containing product	Audio tape player Set-top box Digital camera Multifunction office machine
Complex	Multiple components	Desktop computer Multimedia server
Distributed	Interconnected systems	Radio or television broadcast chain Data communications network Telephone network

close inspection of the wake of interactions created by its introduction. These include additional hardware and software complexities, conflicts with other system functions, requirements for data interchange, and data quality compromises to name a few.

13.1 Data Compression—Do You Need It?

Essentially, the most important question that system designers can ask about data compression is whether it is needed. Are there alternatives that will address the storage, bandwidth, or transmission time reduction it provides—more simply, more economically, or with less disruption? That is, are there, in some sense, better alternatives? Surprisingly (at least in a book dedicated to promoting data compression), sometimes there are. Let us begin with some of the best reasons for using data compression:

- Standards call for data compression
- Competitive products use data compression
- The system cannot operate without data compression
- System specifications are improved
- System cost is reduced

Sometimes the marketplace dictates that data compression must be used; sometimes it does not. Interoperability with a standard that includes data compression

leaves system designers no alternative. Going head-to-head in the marketplace with products that use data compression—and advertise it as an advantage—is almost as compelling a reason. Also, sometimes storage and transmission technology limits are unyielding, and data compression is the only practical solution. This is most evident among current applications where communications bandwidth is insufficient, or outrageously expensive. Then there are the gray areas—the cases where data compression is an option, one that may lead to competitive advantages. For instance, it may provide greater capacity, higher data rates, or more throughput; or it may simply reduce cost, providing yet another form of competitive advantage.

When data compression is an option, not a requirement, system designers must weigh its advantages, balancing them against the effect it has on other system functions and the complexity it adds to system design. In working through this process, a list of reasons for *not* using data compression may emerge. That list might include the following:

- More difficult-to-explain system operations and more complex user interfaces
- More complex hardware and software designs due to variable storage sizes, variable data rates, and the need for buffering
- More processing time and/or more VLSI circuitry
- Longer delay times for real-time data operations
- Increased sensitivity to storage and transmission errors
- More complex data recovery with or without errors

With lossy compression algorithms the list grows longer. Two important reasons not to use lossy compression are as follows:

- Loss of audio, image, and video quality
- Loss of VCR-like functions (random accessibility) for video data

Clearly, many very successful products have dealt with all these issues, but they paid a price, for data compression rarely is free. When data compression is optional, system designers sometimes may find it more cost-effective or satisfactory not to use compression, but to add the extra storage or buy the additional bandwidth it would save. Each situation must be carefully analyzed.

13.2 Selecting an Algorithm

Once a genuine need for data compression is established, system designers face the task of selecting a compression algorithm and adapting it to their application. Increasingly, the particular algorithm for an application is predetermined by prevailing official or de facto standards. In any event, system designers must

still make implementation trade-offs, trade-offs that can affect the compression ratio and many other factors which determine how the algorithm will operate. Data compression algorithms frequently are compared just on compression ratio. Compression ratio is important, but a more thorough characterization includes the factors listed in Table 13.2, all of which are considered in this section.

Table 13.2 Compression algorithm characteristics.

Factor	Importance
Compression ratio	Determines storage and bandwidth required
Adherence to standards	Key to widespread adoption of an algorithm
Algorithm complexity, speed, and processing delay	Determines implementation, cost-performance, and application acceptability
Adaptive versus nonadaptive	Key to supporting applications characteristics and needs
Adaptivity and robustness	Defines allowable input data variations
Error tolerance	Key to success in error-prone environments
Encoder/decoder asymmetry	Key to supporting application communications style
Quality	Defines the acceptability of lossy compression
Multiple-coding limitations	Key to repeated usage of lossy compression
Scalability	Required for differing video display resolutions
Progressive resolution	Required for progressive image transmission at various resolutions
Operating bit rate	Defines acceptable range of data transmission speeds
Added function	Key to more application-friendly compression

Compression ratio

Since compressing data is what this is all about, one would think higher compression ratios are always better, but that is not so. As a rule, getting more compression means making bigger investments in algorithm complexity, process-

ing power, and time. However, these trade-offs have limits: If the compression is lossless, the gains in compression and the added investment costs become marginal as the point is reached where all redundancy has been removed. When the compression is lossy, more compression means more information loss, which nearly always leads to lower-quality results. The most practical approach is to shoot for the minimum compression ratio needed, balancing system storage and bandwidth limits and making adjustments when it is not feasible or acceptable to do it all with compression.

Adherence to standards

Standards for data compression algorithms and their importance are focal points of this book. They exist and many applications demand they be used. Nevertheless, the existence of a standard does not rule out innovative implementations. Many standards for data compression algorithms really are nothing more than frameworks that describe the encoded bit stream and how it is to be transmitted between the encoder and decoder. Many examples throughout the book show that system designers are free to follow different paths in creating and sometimes in decoding those bit streams. They may choose to simplify the encoding process in the name of cost savings, to add complexity in the name of speed or transmission bandwidth reduction, and so on. The only real constraint is that the encoded bit stream conforms to the standard. Some may also choose to develop implementations that “enhance” the standard, but this user-beware strategy is fraught with danger because, in effect, it creates a new standard that defies interoperability.

If compressed information must be communicated between systems and between products from different manufacturers, then a standard for compression and decompression that everyone agrees to is the only choice. Many applications, as we have seen, have already settled on an official standard. But standard algorithms are not always the best choice for applications that have control over their environment, such as a self-contained application (described as a closed process in Section 3.4) that can, if it chooses to do so, implement built-in proprietary compression technology. In addition, there are many other reasons why selecting a proprietary algorithm makes good sense, including taking advantage of technology advances the standard algorithms do not offer to gain more compression, or higher-quality results, or finding a simpler implementation, or having a market advantage by being proprietary. There are also hazards in selecting a proprietary algorithm. What happens to your product if the algorithm vendor goes out of business and, therefore, can no longer provide regular updates and improvements or, equally important, fixes when problems arise? A solution gaining in popularity, at least for software, is to require that vendors place a copy of their code in escrow with an agreement that, if the unthinkable should arise, this information be made legally available to their customers [Seby96, WinW96].

Algorithm complexity, speed, and processing delay time

Algorithm complexity, speed, and processing delay relate to the computational workload and memory requirements that an algorithm imposes, to the need for parallel processing or other efficient processing techniques, and to how fast the algorithm will operate in whatever implementation chosen. These factors decide whether software, DSP, or VLSI hardware logic implementations are suitable for an algorithm and which one(s) will be needed to meet the performance requirements of an application. Two-way communications applications are most sensitive to processing delay time for encoding and decoding, because human interaction places limits on what is acceptable. One-way communications applications and storage applications, where processing delay times are largely irrelevant, are sensitive to processing speed, as throughput determines what level of performance is acceptable.

Adaptive versus nonadaptive

The questions about adaptive algorithms relate to how to design them and when to apply them. Adaptive algorithms continually adjust their parameters to match the data being encoded or decoded. If done properly, more compression will be obtained, as compared to nonadaptive algorithms that operate with static, fixed rules for encoding and decoding. There are some trade-offs to consider when selecting an adaptive algorithm: All adaptive algorithms depend on past history. They have a start-up time, requiring some number of bytes be processed before the algorithm can adjust to the data and the best compression is achieved. The start-up time is a fundamental limitation on the kinds of applications that can use adaptive compression algorithms. Some adaptive algorithms are designed to have shorter start-up times than others. When small, randomly accessed blocks of data are to be compressed, a short start-up time is an advantage, but nonadaptive or semiadaptive algorithms may be still better choices. Examples in Chapters 4 (see Table 4.2) and 6 (see Section 6.9) suggest that a few hundred bytes of data are needed to adapt lossless Lempel-Ziv compression algorithms. Lossy algorithms have start-up times too. For instance, MPEG (see Section 5.5) must encode at least one frame (an I-frame) before motion compensation and interframe prediction can work.

Adaptivity and robustness

Adaptivity and robustness refer to the ability of an algorithm to continually adjust to changes in the data and to produce quality results under adverse conditions. Adaptive algorithms have proven to be superior performers on symbolic data. Many of these algorithms, which were initially designed for text, are robust in the sense that they can handle (or can be extended to handle) graphics and other forms of information. For example, see the discussion of the IBM ADLC

algorithm in Section 9.12. Most algorithms for diffuse data make limited use of this type of adaptivity. For example, the final stage of the JPEG algorithm allows adaptive Huffman and arithmetic entropy coding as options [Penn93]. However, the real opportunity for adaptivity is to deal with the variability of the input characteristics of audio, image, and video data. Diffuse data algorithms can be made more robust by applying adaptivity to varying characteristics such as data statistics, dynamic range, noise, frequency content, pixel-to-pixel correlation, and image resolution. Adaptivity can be applied at many steps of the algorithmic process, including transformation (to adjust the transform block size in transform coding, for example), coefficient selection, quantization step size, motion estimation and compensation, and so on.

Error tolerance

Error tolerance relates to the ability to encode and decode data successfully in the presence of errors that may occur during processing, storage, or transmission. Without question, data compression (coding) schemes are far more sensitive to transmission channel errors than all others. This stems from the fact that, totally apart from the needs of data compression, error control coding and recovery procedures are standard in the processing logic and storage devices used for most digital (computing) systems. These elements handle compressed data with the same aplomb as uncompressed data, with transparent error correction that assures near absolute accuracy, and with little or no increase in running time. Consequently, data compression algorithms intended for computing and storage applications include few if any special error control or recovery provisions.

This is not so for transmission channels. Here, the error rates often are much higher, meaning error recovery must be invoked with greater regularity. Another difference is the user (in this case the compression algorithm), not the transmission channel itself, may be responsible for system error control and recovery. Particularly for real-time transmissions, standard error control and recovery schemes are either far more intrusive or far less effective. For example, a packetized data network typically accomplishes transmission channel error recovery by resending corrupted packets. This form of system error recovery may not be practical for compressed data streams of audio and video data, where the latency of retransmission interferes with real-time playback. In this situation, a much more effective strategy is to limit the effects of a channel error to a small part of the data stream—a few seconds of audio, a small area of an image, or a few frames in a video stream. Section 14.2 will discuss techniques for doing this, but today, as a general rule, each compression algorithm for each application is forced to make additional specific provisions for error handling for each transmission channel. Numerous examples of what has been done are provided in Chapters 5 and 7. The price to be paid, unfortunately, is increased complexity and added bit rate (overhead) for error control and recovery.

Encoder/decoder asymmetry

Encoder/decoder asymmetry describes the expenditure of silicon and time for encoding in relation to the same factors for decoding. The basic idea of asymmetric coding is that a larger investment in encoding will result in fewer bits to be stored or transmitted. Asymmetric algorithms are frequently found in storage or broadcast applications, where encoding is done once and decoding is done many times. Most lossless algorithms are inherently symmetric and there is little that can or should be done to change them. Most lossy algorithms, by their very nature, are asymmetric because deciding what information to remove is frequently the most complex and time-consuming part of the encoding-decoding process. However, some of today's leading lossy algorithms allow trade-offs that will make them more or less asymmetric. Several applications illustrate that intelligently expending more encoding logic and time can result in both higher-quality audio, image, or video data and fewer bits to store or transmit. For example, see the discussion of DBS in Section 8.3 and CD-ROM publishing in Section 10.3.

Quality

Quality refers to the distortion that lossy coding introduces. Because lossy coding involves removing some data, compressed diffuse (speech, audio, image, and video) data may not retain all the quality of the original. As noted in some of the applications described earlier in this book, sometimes lossless compression of diffuse data is required when no loss of quality can be accepted, such as for nonhuman data processing or to meet legal requirements. But what is quality, particularly, what is quality when lossy psychoacoustical and psychovisual models are used for compression? For instance, image quality is a complex psychovisual combination of spatial resolution (detail, sharpness), temporal resolution (frame rate), and the effect of artifacts introduced by the compression process. Unfortunately, there are no good, easily applied quality measures for images or other data subjected to lossy compression.

Standard engineering measures of signal distortion, such as mean-squared error, frequency response, or anything that can be measured by linear instruments, frequently do not correlate well with how humans perceive quality, although, these measures are what engineers often resort to because little else is available. When humans are asked to examine a set of images and make decisions about their quality, they use subjective criteria. Subjective criteria such as whether an image is "visually lossless" are useful, but the downside is they have imprecise definitions. Subjective evaluations (for instance, see Section 12.5), despite their unreliability and time-consuming cost, are the best available tools for judging how objectionable the artifacts created by lossy coding systems are. The types of artifacts produced by these systems include the following:

- Audio distortion due to quantization error (bits/sample)
- Image softness due to low resolution

- Image fringing, blocking, or edge jaggedness
- Loss of image sharpness at edge transitions
- Random noise in images
- Video blurring due to limits in motion estimation

What is known today is that different algorithms create different artifacts, depending on their mode operation. Even a single given algorithm may exhibit different artifacts, depending on the bit rate at which it is operated. The presence of artifacts usually is more apparent at high compression ratios. Also, the objectionability of artifacts is highly dependent on the particular sound or image and the conditions under which it is presented to the listener or viewer.

Quality measures, perhaps, will be easier to develop for audio because we understand (or think we understand) quite well how the human ear perceives sound. But good quality measures for images and video are more difficult to develop. Although our understanding of the human visual system has increased significantly over the past century, it has also revealed many complexities of the visual system. This includes how shape, intensity, color, and motion are perceived. It also includes how we are affected by various media and different techniques to display images (for example, subtractive colors for printers and additive colors for displays), and so on. A simple and yet powerful model for visual processing, one that can be used for optimizing coding systems, remains a goal that still eludes the experts [Netr95].

Multiple-coding limitations

Multiple-coding limitations relate to the loss of quality that may result when data is repeatedly compressed and decompressed with lossy algorithms. Some applications may require that compression-decompression be repeatedly applied during transmission, or for storage on different media, or simply for processing such as when an image is rescaled, clipped, or cropped during document processing. Clearly, successive applications of lossless algorithms do not affect quality, but successive applications of most lossy algorithms cause additional degradation of audio, image, or video quality. Unfortunately, there are no general rules or guidelines that explain when multiple compression-decompression cycles can be safely used. Worse yet, as discussed in Chapter 14, when different algorithms are cascaded, the results can be unpredictable.

Scalability

Scalability, a term often restricted to video coding, may be broadly defined as the generation of bit streams that produce useful results even when only a subset has been decoded. Partial decoding of a video bit stream may be useful if the viewer wants an image of smaller size than the source, or the decoder lacks processing resources to decode the full bit stream, or insufficient channel capacity exists (perhaps momentarily) to carry the full signal. Existing television

standards (NTSC, PAL, and SECAM), like early proposals for HDTV standards, make no provisions to decouple the parameters of encoding from decoding. This means that all receivers must support the same bandwidth, resolution, aspect ratio, and frame rate. With the coming of HDTV, scalability begins to make sense because not every viewer or receiver needs its full resolution and features. A small-screen receiver to be used for casual viewing would be more economical to build if it could decode a subset of the standard. Similar conditions exist for displaying video on computer workstations, where small screens and smaller-yet windows are the norms. Provisions for scalability are incorporated in the emerging U.S. HDTV standard [Chal95]. Essentially, what is required to achieve scalability within the context of the standard are spatial conversion, frame-rate change, and conversion between interlaced and noninterlaced formats [Basi95].

Progressive resolution

Progressive resolution, a form of scalability usually restricted to images, refers to coding and transmitting images at different resolutions or precisions. The idea is to code an image in several passes and sequentially provide these encodings to viewers. The coding may take the form of spectral selection, where image detail is grouped into bands of visual frequency coefficients. It may also be accomplished by successive approximation, where the precision of all image components is limited in early passes, and later passes extend the precision. Either way, transmission begins with an image that can be quickly sent at a low bit rate, allowing the viewer to recognize the image contents as soon as possible. The viewer, perhaps, may choose to halt the transmission of unnecessary detail or may choose to initiate actions for transmission of additional detail. Progressive presentation, besides providing an aesthetically pleasing sequence of images that holds the viewer's attention, allows more efficient browsing through a picture database. This is also known as hierarchical or pyramidal multiresolution coding as described in the Photo CD example that appears in Section 9.9; see also [Rabb91].

Progressive resolution techniques are being actively investigated; many different compression techniques are inherently progressive or can be adapted to operate progressively. However, today there are no strong solutions for managing image compression fidelity, either to satisfy end-user requirements for spatial or amplitude resolution or to meet end-user response time requirements, or to operate within available decompression time limits, or to match available transmission media bandwidth.

Operating bit rate

Operating bit rate refers to the number of bits per second transmitted from the encoder to the decoder. It is usually associated with real-time transmission of audio or video and, sometimes, with image transmission. An ideal compression algorithm would be able to operate over a wide channel-bandwidth range, provid-

ing the highest-quality audio, image, or video sustainable within available bandwidth. The algorithm would be tunable by adjusting a small set of compression parameters to trade off quality for bit rate. Some of today's algorithms, including JPEG, H.261, and MPEG, allow some amount of bit-rate tunability. Others, including nearly all speech coding algorithms, are designed to work only at a given bit rate.

Most compression algorithms are designed to work efficiently within a certain bit-rate range. They may become inefficient or inoperable for transmission rates that fall outside that range—inefficient in the sense that increasing the bit rate will not measurably improve quality, and inoperable in that there is a lower limit below which insufficient transmission bandwidth is available for sending the minimal required information. For example, the H.261 videoconferencing algorithm is designed to operate at not more than 1.92 Mbps and not less than 64 Kbps. At higher rates, there is simply little if any additional information that can be sent to improve picture quality, and at lower rates there is insufficient bandwidth for sending the minimum number of transformed video blocks and motion vectors.

Among those algorithms that can operate at varying bit rates, some require the bit rate be selected and remain fixed for the life of an application, whereas others support dynamically variable bit rates. Examples of the latter include MPEG-2 as implemented for DBS transmission (described in Section 8.3) and for DVD storage (described in Section 9.6). The JPEG hierarchical coding mode also essentially transmits data in this manner.

Added function

This catchall category includes functions that compression algorithms may include to make compression more tolerable for an application. Examples include the sort-order preserving and semiadaptive algorithms developed for database applications (described in Section 6.9).

13.3 Selecting an Implementation Technique

Implementations ranging from software to dedicated special-purpose hardware allow data compression to be adapted to diverse market segments and products, each having differing economics and differing compression needs. Historically, in the computer industry, early compression techniques were executed in software as programs of conventional instructions. In the 1970s, IBM researchers were among the first to define special microcoded CPU instructions and dedicated hardware to speed up data compression techniques such as Huffman coding [Momm74]. By the 1980s, the complexity of compression algorithms had grown, and with more applications demanding faster operation, several VLSI codec chips came to market [Arps88]. In the communications industry, dedicated codec

hardware always has been the preferred implementation, essentially because real-time operations such as speech coding demand it and because computers and software that can operate in real time (or near real time) are relative newcomers in communications applications. In consumer-electronics products, where programmable computing technology is not always a given, hardware-implemented compression is often the method of choice.

The options

The optimal choice for implementation depends on the system requirements. Table 13.3 introduces four cost-performance options available to digital system designers for implementing data compression.

Table 13.3 Data compression implementations.

Option	Performance	Comments
CPU software	Low	Programmable
CPU accelerator	Medium - high	Programmable
DSP chip	Medium	May be programmable
ASIC codec chip	High	Fixed algorithm

CPU software compression is an easy way for computer-based products to introduce new function to the marketplace. For instance, when multiple data formats must be handled, a single hardware solution may be impractical. CPU software offers the cheapest and most flexible solution for many compression applications, as it can handle multiple formats and evolve most quickly to new formats. CPU software also is the right choice in situations calling for occasional usage. For example, when a data file is imported from another system in a heterogeneous computing environment, a software format translator is all that is needed. CPU software is not satisfactory in some situations because of computational complexity, resolution, or rate-of-delivery requirements; there one of the following solutions is often necessary.

Special-purpose CPU accelerators allow the flexibility of software programmability to be retained while providing vastly improved performance. These accelerators may take the form of specialized instructions added to the regular CPU chip, but specially designed coprocessor chips can be far more powerful. As learned in Section 9.11, both are available for multimedia computing applications where data compression and other audio, image, and video data operations require many times the performance of a regular CPU.

The DSP (digital signal processor) chip is a 1990s phenomenon finding its way into more and more applications. The primary advantage of a DSP chip is the specialized dataflow which allows high performance execution of real-time (signal) processing algorithms with far less expense and energy consumption than any CPU chip intended for general-purpose computing. Today's DSPs cost as little as \$10, or less, and they are embedded in many portable products. They also offer the right cost-performance for high-volume, low-end home and small office environment applications such as audio products, modems, telephones, and FAX machines. They are used for implementing many diffuse data compression algorithms discussed in this book. (DSPs are not promoted as engines for implementing lossless symbolic data compression, although it is technically feasible.) The market for DSPs is about equally split between programmable chips that may be downloaded with various algorithms and fixed-function, algorithm-specific chips used for embedded applications where programmability is not required.

ASIC (application-specific integrated circuit) codec chips are appropriate when high-performance data compression solutions are required. In particular, ASIC codec chips may provide the only viable implementation of highly complex, computationally intense algorithms intended for real-time data compression applications. They also may be needed for simpler algorithms in applications where general-purpose computing logic is not available. When compared to the alternatives, ASIC codec chips can also be the lowest-cost solution for these applications. Data compression codec chips are available for almost all speech and audio algorithms, for JPEG and other image compression algorithms, and for video algorithms, including H.261 and MPEG. Chips also are available for many lossless data compression algorithms including Lempel-Ziv and binary arithmetic coding.

Choosing an option

Choosing an implementation for data compression is easy when the digital system design or the application itself dictates the correct choice. When a product does not include a general-purpose CPU, DSPs and ASIC codecs are the only choices. The same is true when software data compression performance is woefully inadequate. But there are gray areas, especially when a product does feature a CPU, one seemingly having enough performance to do software compression. In these situations, specialized data compression hardware could still be a wise choice. The issue is having a balanced design. One possible decision-making approach is to try to quantify the ratio between the operations-per-second devoted to compression versus those available for noncompression-related work. The larger this ratio, the more justified is hardware support for compression. Consider how this classification scheme might apply a hypothetical product built around a modern microprocessor: It would likely reveal that speech could be processed without special hardware. The same could be true for still-image processing if no hard real-time requirements are imposed. However, it would likely point to the need for compression hardware for high-quality video processing.

Future directions

To appreciate what future data compression implementations may be, consider what has happened in the past. Computer CPUs have become microprocessors—smaller, cheaper, and faster—and far more pervasive in products from all industries, even low-cost, mass-produced products. Microprocessors are now able to execute nearly all symbolic data compression algorithms at speeds that satisfy most applications and at costs that are affordable for scores of products. Simultaneously, compression algorithms have grown more complex—handling speech, audio, and video data types with real-time requirements. Where only a few short years ago it was inconceivable that *any* of these real-time algorithms could be executed in software, as a rule today's microprocessors have advanced to execute nearly *all* these algorithms with at least marginally acceptable performance.

Some type of compression hardware assist is necessary for many of today's algorithms, but time and technology march on. One can assert, with visionary clarity, that someday soon, all of *today's* data compression algorithms will operate quite adequately and cost-effectively in software on microprocessors of the future. Those microprocessors will somehow include the processing logic needed for executing the basic data manipulations on which today's algorithms are based. But it would be shortsighted to assume that all future implementations of data compression will be in software. They will not. Data compression algorithms, particularly those for video (and probably speech, audio, and maybe still images too) are going to grow still more complex. Continuing demand for higher-quality video images, transmitted at ever-lower bandwidth and stored evermore compactly, assure that algorithm development must continue. Thus, leading-edge algorithms will undoubtedly continue to require specialized hardware implementations.

13.4 Where To Do Data Compression

Location, location, location: With data compression, as in real estate, location is everything. As asserted throughout this book, from a technology perspective, the optimum location for data compression is close to the source or producer of data, and for decompression, close to the end user or consumer of data. As illustrated in Figure 13.1, this produces the maximum benefit by reducing bandwidth and minimizing storage (and processing time) for all hardware and software elements throughout the system. Clearly, some systems are so simple that no other choice is possible; the digital speech products described in Section 9.13 are a classic example. For almost all systems that handle video data, short of a hugely expensive investment in networks and storage, this is the only practical way to support the data rates needed for real-time display.

Deciding where to place compression and decompression can be far more complex for most digital systems, particularly, digital computer systems. Figure 13.2 shows the work flow for a database computer system. A human interacts

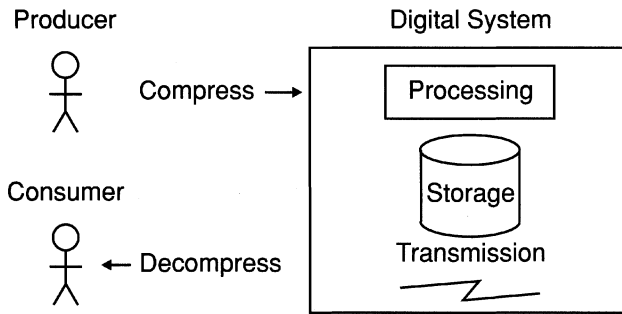


Figure 13.1 Optimal compression for a simple digital system.

with a workstation (WS) connected to a network, which, in turn, is connected to a server computer. Data enters the server through an I/O channel interface (IFAC). It passes through several software components, beginning with communications (COMM), through an application program (APPC), and then is passed to a database management system (DBMS), all of which run in the main storage (MSTG). Data is then routed through I/O channel interfaces to a disk subsystem where it enters a buffer (BUFF) and, subsequently, is passed through yet another interface to disk storage. Data is returned through these same elements in the reverse direction. Along this path there are many sites where it is possible to do compression/decompression functions (indicated by arrows), in hardware or in software.

This situation is quite different from earlier examples for several reasons. To begin with, the data rates and the amount of data for commercial database management applications are much lower; all components are quite able to operate without compression, but there are system cost and performance advantages when it is used. Another difference is that somewhere in this path, data must be processed—not simply passed along. At minimum, the application program, the DBMS code, and, usually, both must examine the data and make logical decisions based on its values. So, the strategy of compressing data when the producer generates it, and leaving it compressed until the consumer needs it, simply does

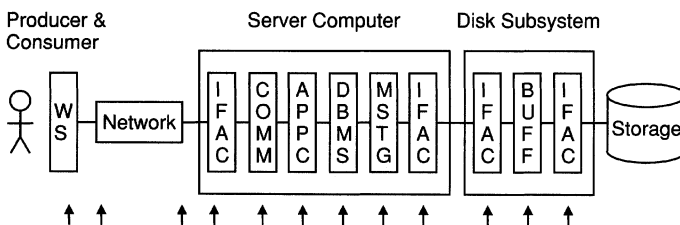


Figure 13.2 Work flow into and out of a database computer.

not work. At least this is true for data processing as we know it today because, generally, the technology to execute program logic on compressed data does not exist. (There are exceptions, such as the compressed DBMS index operations discussed in Section 6.9.) Finally, there are marketplace factors that come into play. A system such as the one shown in Figure 13.2 typically is assembled from components from several sources—a workstation from a manufacturer of computer peripherals, a network from the telecommunications industry, a server from a computer manufacturer, software from several suppliers, and so on. Each manufacturer may be attracted to data compression, using it in their product for the marketing advantages it can offer and choosing to compress and decompress at the product interfaces. Thus, data may be compressed and decompressed several times within this system, a situation that could be interpreted as wasted motion (or processing cycles), but there are valid reasons. It must be noted, from a technology viewpoint, that this is acceptable if the compression is lossless (which it is in business-oriented database computer systems).

In summary, the “rule” that data compression should be placed close to the source of data and decompression placed close to where data is used does not always apply. Sometimes marketplace decisions outweigh technical decisions, and in these situations data compression/decompression can be placed anywhere that adequate computing and storage resources are available and wherever the marketplace makes it profitable!

13.5 Summary

This chapter examined decisions that designers face when choosing to integrate data compression in their digital systems. We learned the following:

- Marketplace requirements define the need for data compression, the type of compression that is acceptable, and limits within which system designers must work.
- To assure data compression can be effectively integrated, system designers must understand not only the problems data compression addresses but also its interactions with other system functions.
- Essentially, the most important question that system designers can ask about data compression is whether it is needed or whether other alternatives may provide a better choice.
- Sometimes the marketplace or the limitations of storage and transmission technology demand that data compression be used; in other situations it is an option, providing competitive advantage through improved system cost-performance.
- System designers must deal with the costs for using data compression including its effect on other system functions and the complexity it adds to system design.
- All too frequently, the compression ratio is the only parameter used to evaluate compression algorithms; actually, there are many other characteristics that may be far more important in many system designs.

- Four cost-performance options are available for doing data compression, ranging from software to dedicated special-purpose hardware. The optimal choice for implementation depends on the system requirements.
- Software compression is becoming more powerful, but most leading-edge algorithms require some type of hardware assist and will continue to do so.
- The optimum location for data compression is close to the source or producer of data, and for decompression, close to the end user or consumer of data. But this cannot always be achieved for a variety of technical and marketing reasons.