Multimedia Systems
© Springer-Verlag 1994

[Stei94d]. *Ralf Steinmetz;* **Data Compression in Multimedia Computing: Principles and Techniques;** acm/Springer Zeitschrift "Multimedia Systems", Band 1, Nr. 4, S. 166-172, Februar 1994.

Data compression in multimedia computing – principles and techniques

Ralf Steinmetz

IBM European Networking Center, Vangerowstrasse 18, D-69115 Heidelberg, Germany

Abstract. Integrated multimedia systems process text, graphics, and other discrete media as well as digital audio, and video data. Considerable amounts of graphics, audio and video data in their uncompressed form, especially moving pictures, require storage and digital network capacities that will not be available in the near future. Nevertheless, local, as well as networked, multimedia applications and systems have become realities. In order to cope with these storage and communication requirements in such integrated multimedia systems, compression technology is essential. This papers starts with a brief motivation of the need for compression and subsequently states the essential requirements for these techniques in the scope of multimedia systems and applications. As most of these techniques apply the same principles, namely, the source, entropy, and hybrid coding fundamentals, these are explained in detail. Based on a general framework of the steps encountered in a compression system – data preparation, processing, quantization, and entropy coding – this paper outlines details about the techniques developed by CCITT (H.261, i.e., px64), in the ISO/IEC (JPEG, MPEG) standardization bodies and the proprietary DVI system.

Key words: Multimedia data – Compression – MPEG – JPEG – H.261 – DVI

1 Introduction

A multimedia system is characterized by the computer controlled integrated generation, manipulation, presentation, storage, and communication of independent digital information (Steinmetz 1993). This information is most often coded in continuous – time dependent – media (e.g., audio, video) as well as discrete – time independent – media (e.g., text, graphics). The storage of uncompressed graphics, audio, and video data requires considerable capacity. In the case of uncompressed video, such storage is often not even feasible with today's CD technology. The same is true for multimedia communications; the data transfer of uncompressed video data over digital networks occupies very high bandwidth specially if it is used for point-to-point communications. In order to provide feasible and cost effective solutions, most multimedia systems handle compressed digital video and audio data streams.

Many compression techniques exist (ACM 1989, 1991; Aravind 1993; IEEE 1992a,b; Lu 1993) that are in part competitive and in part complementary. Most of them are used in today's products, while other methods are still being developed or are only partly realized [see also (IS&T/SPIE 1994)]. Whereas fractal image compression (Barnsley and Hurd 1993) might be relevant in the more distant future, today and in the near future the most important compression techniques are Joint Photographic Experts Group (JPEG) – for images see JPEG (1993), Pennebaker and Mitchell (1993) and Wallace (1991) –, H.261 px64, for video see Le Gall (1991) and MPEG (1993) –, Moving Pictures Experts Group (MPEG) [for video and audio (Liou 1991; CCITT 1990)] and proprietary developments include Digital Video Interactive (DVI) – for still images, audio, and video see Harney et al. (1991) – Intel's Indeo, Microsoft's Video for Windows, IBM's Ultimotion Matinee, Apple's QuickTime or DigiCipher II developed by General Instruments and AT&T.

In their daily work, developers and multimedia experts must often understand the most popular techniques very well. Most of today's literature is either superficial or dedicated to one of the compression techniques mentioned, which is described from a very specific point of view. In this article, however, the most relevant techniques (JPEG, H.261, MPEG, DVI) are described in order to clarify their advantages and disadvantages, their common abilities and differences, and their suitability for today's multimedia systems. Therefore this paper presents a detailed overview of various compression schemes, making use of their commonalities to provide the reader with an understanding of their goals, internals, and features.

Note that in this paper the terms *coding* and *compression* are treated as synonyms.

First, a short motivation for the need of compression is given and then some requirements for these techniques are derived. In Sect. 4 source, entropy, and hybrid coding techniques are explained in detail, while Sects. 5–8 provide details on JPEG, H.261, MPEG and DVI respectively.

2 Motivation and requirements

Images require considerably more storage space than text, while audio and video demand even more of data storage. Not only is a huge storage required, but the data rates for the communication of continuous media are also significant, as we illustrate in this section (Blair et al. 1991; Herrtwich and Steinmetz 1991). With specific numbers, we clarify the qualitative transition from simple text to full motion video data and derive the need for compression.

For reasons of comparability of the different media, the following calculations are based on a typical image format of 640 × 480 pixels on a screen based on some specific assumptions as stated in the individual paragraphs:

– For the representation of *text*, 2 bytes are used for each character (allowing for some Asian language variants). Each character is displayed by 8 × 8 pixels, which is sufficient for the display of ASCII characters.

$$\text{characters per screen page} = \frac{640 \times 480}{8 \times 8} = 4800$$

$$\text{storage required per screen} = 4800 \times 2\ \text{bytes} = 9600\ \text{bytes}$$

$$\text{storage required per screen} \approx 9.4\ \text{KB}\ .$$

– For the presentation of *vector graphics*, a typical still image is composed of 500 lines (Blair et al. 1991). Each line is defined by its horizontal position, its vertical position, and an 8-bit attribute field. The horizontal axis is represented by 10 bits $\lceil \log_2(640) \rceil$, the vertical axis is coded with 9 bits $\lceil \log_2(480) \rceil$.

$$\text{bits per line} = 9\ \text{bits} + 10\ \text{bits} + 9\ \text{bits} + 10\ \text{bits} + 8\ \text{bits}$$
$$= 46\ \text{bits}$$

$$\text{storage required per screen} = 500 \times \frac{46}{8}\ \text{bytes} = 2875\ \text{bytes}$$

$$\text{necessary storage per screen} \approx 2.8\ \text{KB}\ .$$

– In very simple color display modes a single pixel of a *bitmap* can be represented by 256 different colors, therefore 1 byte per pixel is needed.

$$\text{storage required per screen} = 640 \times 480 \times 1\ \text{byte}$$
$$= 307\,200\ \text{bytes}$$

$$\text{storage required per screen} = 300\ \text{KB}\ .$$

The next examples cover continuous media and derive the amount of storage required for 1 s of playback:

– An uncompressed *audio* signal of telephone quality is sampled at 8 kHz and quantized with 8 bits per sample. This leads to a data stream of 64 kbits/s

$$\text{storage required/s} = \frac{64 \times 1024\ \text{bits/s}}{8\ \text{bits/byte}} \times \frac{1\ \text{s}}{1024\ \text{bytes/KB}}$$

$$\text{storage required/s} = 8\ \text{KB}\ .$$

– An uncompressed stereo *audio signal* of CD quality is sampled at a rate of 44.1 kHz and is quantized with 16 bits per sample.

$$\text{data rate} = 2 \times \frac{44\,100}{\text{s}} \times \frac{16\ \text{bits}}{8\ \text{bit/byte}} = 176\,400\ \text{bytes/s}$$

$$\text{storage required/s} = 176\,400\ \text{bytes/s} \times \frac{1\ \text{s}}{1024\ \text{bytes/KB}}$$

$$\text{storage required/s} \approx 172\ \text{KB}\ .$$

– The European phase alternate line (PAL) standard video is defined as 625 lines and 25 frames/s. The luminance and the color difference signals are encoded separately. The resulting digital data streams are transformed using a multiplexing technique (4:2:2). Corresponding to the CCIR 601 studio standard for digital video (CCIR 1982) a sampling rate of 13.5 MHz is used for luminance Y. The sampling rate for chrominance (R-Y and B-Y) is smaller, 6.75 MHz, because it is not as important as the lumininance Y. The result is a uniform 8bit coding of each sample, i.e.,

$$(13.5\ \text{MHz} + 6.75\ \text{MHz} + 6.75\ \text{MHz}) \times 8\ \text{bits}$$
$$= 216 \times 10^6\ \text{bits/s}\ .$$

High definition television (HDTV) doubles the number of lines and uses an aspect ratio of 16/9. This leads to a data rate that is increased by the factor 5.33 compared to today's TV.

For a video sequence in our example we stick to the same resolution as used in the other examples of 640 × 480 pixels. In Europe we encounter 25 images/s. We encode the luminance and chrominance of each pixel in 3 bytes.

$$\text{data rate} = 640 \times 480 \times 25 \times 3\ \text{bytes/s} = 23\,040\,000\ \text{bytes/s}$$

$$\text{storage required/s} = 23\,040\,000\ \text{bytes/s} \times \frac{1\ \text{s}}{1024\ \text{bytes/KB}}$$

$$\text{storage required/s} = 22\,500\ \text{KB}\ .$$

The storage and throughput requirements of a computer system that processes still images and in particular continuous media are illustrated by these few examples. In the case of video, the processing of uncompressed data streams in an integrated multimedia system leads to secondary storage requirements in the range of at least gigabytes and in the range of megabytes for buffer storage. The throughput in a multimedia system can be as high as 140 Mbits/s, which has to be transferred between different systems. Neither today nor in the near future can technology achieve this kind of data transfer rate with reasonably priced hardware. However, the use of appropriate compression techniques considerably reduces the data transfer rates (Netravali and Haskell 1988; Rabbani and Jones 1991) and fortunately research, development, and standardization have progressed in this area during the last few years (ACM 1991).

Some compression techniques for different media are mentioned often in literature and in product descriptions: JPEG for still image compression and H.261 (px64) for video conferencing. Additionally, some audio coding techniques developed for integrated systems digital networks (ISDNs) and mobile communications can also be used for the compression of speech and music. MPEG is used for video and audio compression, while DVI can be used for still images as well as for continuous media. DVI consists of two different modes for video coding, presentation level video (PLV), and real-time video (RTV).

Compression in multimedia systems is subject to certain constraints. The quality of the coded, and later on, decoded data should be as good as possible. In order to make a cost-effective implementation possible, the complexity of the technique used should be minimal. The processing of the algorithm must not exceed certain time spans.

For each compression technique, there are requirements that differ from those of other techniques – see, for example, the requirements of MPEG (1993a). One can distinguish between requirements of applications in a "dialogue" mode and in a "retrieval" mode. Some techniques like px64 are more suitable for dialogue applications (with the same, symmetric, effort for compression and decompression and a limited delay). Other techniques, like the DVI PLV mode, are optimized for use in retrieval (more time and effort may be expended during offline compression in favor of fast retrieval and decompression).

In a dialogue application, the following requirements based on human perception characteristics must be considered:

- In general, the overall end-to-end delay should not exceed 150 ms. However, the compression–decompression delay should be reduced to approximately 50 ms for *face-to-face* dialogue applications. The *overall* end-to-end delay additionally comprises any delay in the network, in the communication protocol processing at the end system, and in the data transfer from and to the respective input and output devices.

For a retrieval mode application the following demands arise:

- Fast forward and backward data retrieval with simultaneous display should be possible. This allows a search for the information that is needed in these media.
- Random access to single images and audio frames of a data stream should be possible, making the access time less than 0.5 s. This access should be faster than in a conventional CD digital-audio system in order to maintain the interactive character of the application.
- The decompression of images, video, or audio should be possible without the knowledge of other data units. This allows random access and editing.

For both the dialogue and the retrieval mode the following requirements apply:

- In order to support scalable video in different systems, it is necessary to define a format independent of frame size and video frame rate.
- Various audio and video data rates should be supported; usually this leads to different qualities. Thus, depending on specific system conditions, the data rates can be adjusted.
- It must be possible to synchronize audio data with video data as well as with other media.
- In order to make an economical solution possible, coding should be realized using software (as a cheap and low-quality solution) or very large scale integration (VLSI) chips (for high quality).
- It should be possible to generate data on one multimedia system and reproduce this data on another system. The compression technique should be compatible. This compatibility is relevant in the case of tutoring programs available on a CD, for example; it allows different users to read the data on different systems, thus being independent of the manufacturers. As many applications exchange multimedia data using communication networks, the compatibility of compression techniques is required. Standards such as those of the Comité Consultatif Internationale de Telegraph et Telephonie (CCITT), the International Standards Organization (ISO) and the European Computer Manufacturing Association (ECMA) and/or de facto standards are used to reach this compatibility.

This set of requirements is taken into account to a varying extent by the various compression schemes.

In order to clarify these various schemes in the following section the distinction between source coding and entropy coding is first introduced with respect to the hybrid methods discussed later on.

3 Source, entropy, and hybrid encoding

Compression techniques fit into different categories as shown in Table 1. For their use in multimedia systems we can distinguish between *entropy*, *source*, and *hybrid encoding*. Entropy encoding is a "lossless" process while source encoding is a "lossy" process. Most multimedia systems use hybrid techniques that are combinations of the two.

Entropy encoding is used for media regardless of their specific characteristics. The data stream to be compressed is considered a simple digital sequence and the semantics of the data are ignored. Entropy encoding is an example of lossless encoding as the decompression process regenerates the data completely. Run length coding is an example of entropy encoding that is used for data compression in file systems, still images as facsimile, or as part of a video or audio coding process.

Source encoding takes into account the semantics of the data. The degree of compression that can be reached by source encoding depends on the data contents. In the case of lossy compression techniques, a one-way relation between the original data stream and the encoded data stream exists; the data streams are similar but not identical. The different source encoding techniques make extensive use of the characteristics of the specific medium. For example, in the case of speech, a transformation of time to frequency domain followed by the encoding of the formants substantially reduces the amount of data. *Formants* are defined as being the maxima of the voice spectrum. In most cases three to five formants are sufficient to reconstruct the original signal in the time domain. The major problem is the correct reproduction of the transitions between individual voice units in the time domain.

Table 1. A rough classification of coding/compression techniques for multimedia systems

Entropy coding	Run-length coding	
	Huffman coding	
	Arithmetic coding	
Source coding	Prediction	DPCM
		DM
	Transformation	FFT
		DCT
	Layered coding	Bit position
		Subsampling
		Sub-band coding
	Vector quantization	
Hybrid coding	JPEG	
	MPEG	
	H.261	
	DVI RTV, DVI PLV	

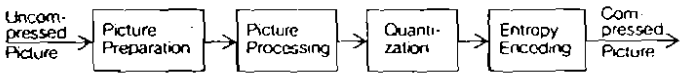


Fig. 1. Major steps of data compression

A content prediction technique can make use, for example, of spatial redundancies within still images. Other techniques transform the spatial domain into a two-dimensional frequency domain by using the discrete cosine transform (DCT). Low frequencies define the "average" color, the information of high frequencies contains sharp edges. Hence, low frequencies are much more important than the higher ones, which is a key feature used in DCT-based compression.

Table 1 shows some examples of coding and compression techniques that are applicable to multimedia applications in relation to the entropy, source, and hybrid coding classification. For the reason of a better and clearer view of the hybrid schemes we will identify in all of them a set of typical processing steps.

Figure 1 shows this typical sequence of operations performed in the compression of still images, video, and audio data streams. The following four steps describe the compression based on the example of the media image.

- *Preparation* includes the analogue-to-digital conversion, generating an appropriate digital representation of the information. A picture is divided into blocks of 8 × 8 pixels, and represented by a fixed number of bits per pixel.
- *Processing* is actually the first step of the compression that makes use of sophisticated algorithms. A DCT can transform from the time to the frequency domain. Interframe coding uses a motion vector for each of the 8 × 8 blocks.
- *Quantization* processes the results of the previous step, which reduces precision. It defines the resolution of the mapping of the real numbers into integers. This can also be considered as the equivalence of the companding μ-law and the A-law that apply to audio data (Jayant and Noll 1984). In the transformed domain, the coefficients are distinguished by their significance, for example, they could be quantized using a different number of bits per coefficient.
- *Entropy* encoding is usually the last step in our reference scheme as shown in Fig. 1. It compresses a sequential digital data stream without loss. For example, a sequence of zeros in a data stream can be compressed by specifying the number of occurrences followed by the zero itself.

Processing and quantization can be iterated several times in "feedback" loops such as in the case of adaptive difference pulse code modulation (ADPCM). After compression, the digital data is combined into a data stream, for example, at the starting point of an image. An identification of the compression technique used may be part of this data stream; the error correction code may also be added. Figure 1 shows the compression process applied to an image; the same principle can be applied to video and audio data.

Decompression is the inverse process of compression. The specific encoders and decoders can function in various ways: symmetric applications, e.g., dialogue applications, should be characterized by more or less the same costs for encoding and decoding. In the case of asymmetric techniques, the decoding process is less costly than the encoding process. This is used for applications in which the data are compressed only once and for which ample time is available; the data are decompressed frequently, and speed is required. An audio-visual tutoring program will be produced once, but it will be used by many students; therefore, it will be decoded many times. In this case, real-time decoding is a fundamental requirement, whereas encoding need not be real-time. This asymmetric processing can be exploited in order to increase the quality of the images.

The understanding of this exploitation as well as the schemes used in multimedia systems requires knowledge of some basics in this area, which are presented in the next section in the form of an overview.

4 Some basic compression techniques

Hybrid compression techniques are a combination of well known algorithms and transformation techniques that can be applied to multimedia systems. For example, all hybrid techniques shown in Table 1 use entropy encoding (in the form of run-length encoding and/or a statistical compression, for more details see also Gonzalez and Woods (1992)).

The simplest compression techniques are based on *interpolation* and subsampling. Here it is possible to make use of the specific physiological characteristics of the human eye or ear. The human eye is more sensitive to changes in brightness than to color changes. Therefore it is reasonable to divide the image into a luminance component (Y) and two chrominance difference signals (U, V) with $Y = 0.30R + 0.59G + 0.11B$, $U = B - Y$, $V = R - Y$ instead of using red, green, and blue (RGB) components. The components U and V are sampled with a lower resolution.

Sampled image, audio, and video data streams often contain sequences of the same bytes; by replacing these sequences with the repetitive byte pattern and providing the number of occurrences, a substantial reduction of data can be achieved. This *run length coding* is indicated by a special flag that does not occur as a part of the data stream itself. This flag byte can also be any other of the 255 different bytes in the compressed data stream. To illustrate such a "byte stuffing" we define the exclamation mark "!" as such a special flag. A single occurrence of this exclamation mark is interpreted as a special flag during decompression. Two consecutive exclamation marks are interpreted as an exclamation occurring within the data. The overall run length coding procedure can be described as follows: If identical bytes occur consecutively at least four times, the number of occurrences is counted. The compressed data contains this byte followed by the special flag and the number of its occurrences. This allows us to compress from 4 to 259 bytes into three bytes only. Remembering that we are compressing at least 4 consecutive bytes the number of occurrences can start with an offset of -4. Depending on the algorithm, one or more bytes can be used to indicate the length. In the following example the character "C" occurs eight times and is "compressed" to three characters "C!8":

Uncompressed data : ABCCCCCCCCDEFGGG

run length coded : ABC!8DEFGGG

Run length encoding is a generalization of *zero suppression*; it assumes that just one symbol appears particularly often in sequences. The blank in text is such a symbol; single blanks or pairs of blanks are ignored. Starting with the sequence of three blanks, they are replaced by an M byte (e.g. an exlamation mark character which does not appear in the data itself) and a byte that specifies the number of blanks of this sequence. Sequences of three to a maximum of 258 bytes can be reduced to two bytes. The number of occurrences can be indicated with an offset of -3. Further variations are "tabulators" used to substitute a specific number of zeros (or blanks), depending on the relative position within a line and the definition of different M bytes to specify a different number of zero bytes (or blanks). The flag M4 byte could, e.g., replace 8 zero bytes ($8 = 2^4$), another M5 byte could substitute a sequence of 16 zero bytes ($16 = 2^5$). An M5 byte followed by an M4 byte would represent 24 zero bytes.

In case of *vector quantization*, see, e.g. Gray (1984), a data stream is divided into blocks of n bytes each ($n > 1$). A predefined table contains a set of patterns. For each block, the table entry with the most similar pattern is identified. Each pattern in the table is associated with an index. Such a table can be multidimensional; in this case the index will be a vector. The decoder uses the same table to generate an approximation of the original data stream. For further details and refinements see, e.g., Gray (1984).

A technique that can be used for text compression substitutes single bytes for patterns that occur frequently. This *pattern substitution* replaces, for instance, the terminal symbols of high-level languages (Begin, End, If). With the use of an escape byte, a larger number of patterns can be considered. This escape byte indicates that an encoded pattern will follow. The next byte is an index used as a reference to one out of 256 words. The same technique can be applied to still images, video, and audio as well. In these media it is not that easy to identify small sets of frequently occurring patterns. It is often better to perform an approximation that looks for the most similar pattern instead of searching for the same pattern in either case leading to the vector quantization described.

Diatomic encoding is a variation based on a combination of two data bytes. This technique determines the most frequently occurring pairs of bytes. According to the analysis of the English language, the most frequently appearing pairs are the following (note: there are blanks included in the pairs "E ", "T ", " A", "S "):

"E ", "T ", "TH", " A", "S ", "RE", "IN", and "HE"

The replacement of these pairs by special single bytes that do not occur anywhere else in the text leads to a data reduction of more than 10%.

Different characters do not have to be coded with a fixed number of bits. The Morse alphabet is based on this idea: frequently occurring characters are coded with shorter strings and seldom occurring characters are coded with longer strings. *Statistical encoding* depends on the frequency of the occurrence of single characters or sequences of data bytes. There are different techniques that are based on these statistical methods. The most prominent are Huffman and arithmetic encoding.

Given the characters that must be encoded together with the probability of their occurrences, the Huffman coding (Huffman 1952) algorithm determines the optimal code using the minimum number of bits given the probability. Hence the length (number of bits) of the coded characters will differ. In a given text, the shortest code is assigned to those characters that occur most frequently. To determine a Huffman code it is useful to construct a binary tree. The leaves of this tree represent the characters that are to be encoded. Every node contains the – relative – probability of the occurrence of one of the characters belonging to this subtree. Zero and 1 are assigned to the edges of the tree.

The following example illustrates this process:

- In Fig. 2 the characters A, B, C, D, and E have the following relative probability of occurrence:

$$p(A) = 10, p(B) = 30, p(C) = 5, p(D) = 8, p(E) = 6 .$$

- The characters with the lowest probabilities are combined in the first binary tree; C and E are leaves. The combined

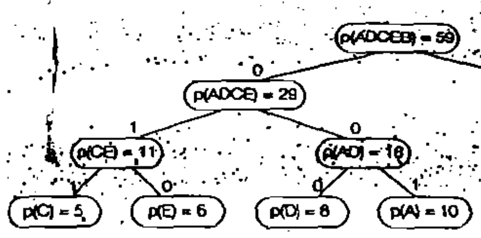


Fig. 2. An example of a Huffman code represented as a binary tree

probability of their root node CE is 11. The edge from node CE to node C is assigned a 1 and the edge from CE to E becomes a 0. This is an arbitrary assignment; therefore, with the same data one can get different Huffman codes.

– The following nodes are left,

$$p(A) = 10, p(B) = 30, p(CE) = 11, p(D) = 8 .$$

Again the two nodes with the lowest probabilities are combined into a binary subtree; the nodes A and D are such leaves and the combined probability of their root AD is 18. The edge from AD to A is assigned a 1 and the edge from AD to D a 0. Note: if there are root nodes of different subtrees with the same probabilities, the trees with the shortest maximal path between their roots and their nodes should be combined. This keeps the code length more constant.

– The following nodes are left:

$$p(AD) = 18, p(B) = 30, p(CE) = 11 .$$

The nodes with the smallest probabilities are AD and CE. They are combined into a binary tree; the combined probability of their root node ADCE is 29. The edge from ADCE to AD is assigned a 0 and the edge from ADCE to CE a 1.

– Two nodes are left:

$$p(ADCE) = 29, p(B) = 30$$

They are combined to a binary tree with the root ADCEB. To the edge from ADCEB to B is assigned a 1, to the edge from ADCEB to CEAD is assigned a 0.

– Figure 2 shows the resulting Huffman code in the form of a binary tree. The result is the following code that is stored in a table:

$$w(A) = 001, w(B) = 1, w(C) = 011, w(D) = 000 ,$$
$$w(E) = 010.$$

If the information of an image can be transformed into a bitstream, then such a table can be used to compress the data without any loss. The most simple form to generate a bitstream is to code the pixels individually and read them line by line. Note that more sophisticated methods are usually applied as described in the remaining part of this paper. Such a stream can be determined for each image or for a set of images. For videos a Huffman table can be used for a single sequence of images, for a set of scenes or even for an entire film clip. The same Huffman table must be available for both encoding and decoding.

Considering run length coding and all the other methods described so far, the coding of identical symbols (bytes) that appear repetitively and consecutively certainly is a major objective in transforming images and videos into a bitstream.

From the information theory point of view, *arithmetic coding* (Langdon 1984; Pennebaker et al. 1988), like the Huffman coding, is optimal. Therefore the length of the encoded data stream is also minimal. Unlike Huffman coding, arithmetic coding does not encode each symbol separately; instead each symbol is coded by considering the previous data. Therefore an encoded data stream must always be read from the beginning. Consequently, random access is not possible. In practice, the average compression achieved by arithmetic and Huffman coding is similar (Storer 1988).

Transform encoding pursues a different approach. Data is transformed into another mathematical domain that is more suitable for compression. The inverse transform must exist and is known on encoding. The most widely known example is the Fourier transform, which transforms data from the time into the frequency domain. The most effective transforms for image compression are the discrete cosine transform (DCT) (Sect. 5.2) and (to some extent) the fast Fourier-transform (FFT).

Unlike transform encoding that transforms all data into another domain, the selective frequency transformation *subband* coding considers a spectral selection of the signal in predefined frequency bands. The number of bands is an important criterion for quality. This technique is well-suited to the compression of speech and often makes use of the FFT for the spectral filtering.

Instead of compressing single bytes or sequences of bytes, a differential encoding can be used. This is also known as *prediction* or *relative encoding*. Consider the example of a sequence of characters whose values are clearly different from zero, but which do not differ much. In this case, the calculation of the difference to the previous values could be profitable for compression. The following explains this technique for different media:

– For still images, the calculated differences between nearby pixels or pixel groups at the edges represent large values, whereas areas with similar luminance and chrominance are characterized by small values. A homogeneous area is characterized by a large number of zeros that could be further compressed using run-length encoding.
– The use of relative coding in the time domain for video can lead to encoding of the differences from the previous image only. The background in newscast and video telephone applications often remains the same for a long time, therefore the "difference" between subsequent images is very small, leading to a large set of zeros. Another very popular technique is *motion compensation* (Puri and Arauind 1991). Blocks of 8×8 or 16×16 pixels in consecutive pictures are compared; for example, consider a car moving from left to right. An area further left in a previous picture would be very similar to the same area of the current picture, where this "motion" can be identified by a motion vector.

- Audio techniques often apply *differential pulse code modulation* (DPCM) to a sequence of *pulse code modulation* (PCM)-coded samples (Jayant and Noll 1984). It is not necessary to store the whole number of bits of each sample. It is sufficient to represent only the first PCM-coded sample as a whole and all following samples as the difference to the previous one.
- The *delta modulation* (DM) is a modification of the DPCM (Jayant and Noll 1984). When coding the differences, it uses exactly one bit, which indicates whether the signal increases or decreases. This leads to an inaccurate coding of steep edges. This technique is particularly profitable if the coding does not depend on 8-bit grid units. If the differences are small, a smaller number of bits is sufficient. Difference encoding is an important feature of techniques used in multimedia systems. Further "delta" methods applied to images are described in the JPEG section on lossless compression.

Most of the compression techniques described so far are based on known characteristics of the data, e.g., sequences of bytes occurring frequently or the probability of the occurrence of certain bytes. An atypical sequence of characters will not be compressed using these methods.

Adaptive compression techniques adapt a particular compression technique to the particular data to be compressed on the fly. This adaptation can be implemented in different ways:

- In one situation, a predefined coding table exists, such as the one invented by Huffman. For each symbol to be encoded, the table contains a corresponding code and a counter in an additional column. This counter is reset to zero at the beginning. For the first symbol to be encoded, the coder takes the code from the table. Then the counter corresponding to this table entry is increased by one. In order to reduce the access time for the individual entries in the table, they are sorted in descending order of the values of the accompanying counters. This procedure leads to the encoding of the most frequently occuring symbols with the shortest codes.
- A prominent adaptive compression technique is *adaptive DPCM* (ADPCM). It is a "follow-on" development of DPCM, which is often subsumed under the acronym DPCM. Here differences are encoded using a small number of bits only (e.g., 4 bits). Therefore either rough transitions will be coded correctly (these bits would represent bits with a higher significance) or small changes are coded exactly (the DPCM encoded values are the less significant bits). In the first case, the resolution of low audio signals would not be sufficient and in the second case, a loss of high frequencies would occur. ADPCM makes an adaptation to this "significance" for a particular data stream as follows.
  The coder divides the value of DPCM samples by a suitable coefficient and the decoder multiplies the compressed data by the same coefficient, i.e., the step size of the signal changes. The value of the coefficient is adapted to the DPCM encoded signal by the coder. In the case of a signal with high frequency, very high DPCM coefficient values occur. The coder determines a high value for the coefficient. The result is a very rough quantization of the DPCM signal in passages with steep edges. Low-frequency portions of such passages are hardly considered at all. For a signal with permanently relatively low DPCM values, i.e., with few portions of high frequencies the coder will determine a small coefficient. Thereby a good resolution of the dominant low-frequency signal portions is guaranteed. If high-frequency portions of the signal suddenly occur in such a passage, a signal distortion in form of a *slope overload* arises. Considering the actually defined step size, the greatest possible change using the existing number of bits will not be large enough to represent the DPCM value with an ADPCM value. The transition of the signal will be *faded*. It is possible to insert a change of the coefficient explicitly that is adaptively adjusted to the data during compression. Alternatively, the decoder is able to calculate the coefficients itself from an ADPCM encoded data stream. An audio signal with frequently changing frequency portions of extreme high or low frequencies is not very suitable for such an ADPCM encoding. In the G.700 series of standards, CCITT has standardized a version of the ADPCM technique using 32 kbits/s for telephone applications that is based on 4 bits per sample and an 8 kHz sampling rate.

Apart from the whole set of basic compression techniques described in this section, some other well-known techniques are also being used today:

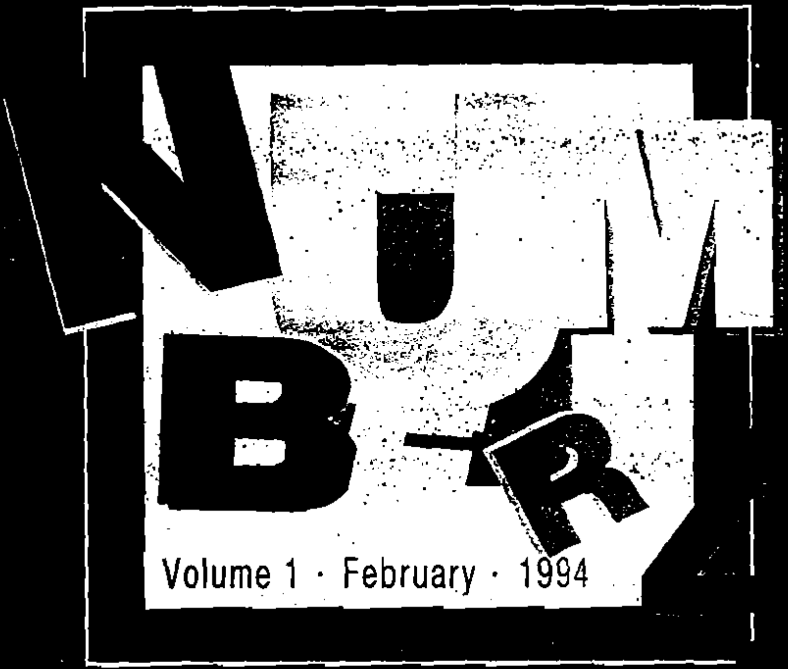
- Video compression techniques often use color look-up tables (CLUT) to achieve data reduction. For instance in Lamparter and Effelsberg (1991) and Lamparter et al. (1992) this technique is used in distributed multimedia systems.
- A simple technique for audio is silence suppression; in this case data are only encoded if the volume level exceeds a certain threshold. This can be seen as special case of run-length encoding.

CCITT incorporated some of the basic audio coding schemes into the G.700 series of standards: G.721 defines the PCM coding for a quality of 3.4 kHz over 64 Kbits/s channels. Finally G.728 defines 3.4 kHz quality over 16 Kbits/s channels. Atal et al. (1993) provide a more detailed description of various audio coding techniques.

In the following sections the most relevant work in the standardization bodies concerning image and video coding is outlined. In the framework of ISO/IECJTC1/SC2/WG8 four subgroups were established in May 1988: the Joint Photographic Experts Group (JPEG) working on coding algorithms for still images, Joint Bilevel Expert Group (JBIG) working on the progressive processing of "bilevel" coding algorithms, Computer Graphics Expert Group (CGEG) working on coding principles and Moving Pictures Experts Group (MPEG) working on the coded representation of motion video. The next section presents the results of the JPEG activities.

Note to the reader: For literature references and biography of the author see Part 2 of this article which will be puslihed in the next issue.

Multimedia SYSTEMS



Volume 1 · February · 1994

Association for Computing Machinery

Springer International